

# Diploma Thesis

## Development of a lattice based blind signature scheme

Frank Nauheimer

June 11, 2007



Darmstadt University of Technology  
Department of Computer Science  
Cryptography and Computeralgebra  
Prof. Dr. Johannes A. Buchmann

Advisor: Roberto Samarone dos Santos Araújo



# Ehrenwörtliche Erklärung

Hiermit versichere ich, die vorliegende Diplomarbeit ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus den Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den 11.6.2007

Frank Nauheimer



# Danksagung

Als erstes möchte ich mich an dieser Stelle bei Prof. Dr. Johannes Buchmann und Roberto Samarone dos Santos Araújo bedanken, die mich bei der Anfertigung betreuten und mir diese Diplomarbeit letztendlich erst ermöglichten. Ebenso danke ich Prof. Dr. Jintai Ding und Axel Schmidt für die hilfreichen Diskussionen, Martin Döring für die Hilfe bei technischen Fragen, meinen Eltern, die mich das ganze Studium über unterstützt haben und Janina, die mich immer zu motivieren wußte.



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	A blind signature . . . . .	2
1.2	Hard problems and quantum computing . . . . .	2
1.3	Existing blind signature schemes . . . . .	3
1.3.1	Blind RSA signature scheme . . . . .	3
1.3.2	Blind ElGamal signature scheme . . . . .	4
1.3.3	Blind Schnorr signature scheme . . . . .	5
1.3.4	Other blind signatures . . . . .	5
1.4	Organization . . . . .	5
<b>2</b>	<b>Lattice based signature schemes</b>	<b>7</b>
2.1	Formal basics about lattices . . . . .	7
2.1.1	Lattices . . . . .	7
2.1.2	Qualities of lattices . . . . .	8
2.1.3	Hard problems in lattices . . . . .	8
2.2	Lattice based signature schemes . . . . .	9
2.2.1	Idea of a lattice based signature scheme . . . . .	9
2.2.2	Goldreich, Goldwasser and Halevi (1996) . . . . .	11
2.2.3	Miccianco (2001) . . . . .	11
2.2.4	NTRUSIGN (2003) . . . . .	11
2.3	Security . . . . .	14
<b>3</b>	<b>Proposals</b>	<b>15</b>
3.1	General idea for a lattice blind signature scheme . . . . .	15
3.2	First proposal - adding a random lattice vector . . . . .	16
3.2.1	Idea . . . . .	16
3.2.2	Security . . . . .	17
3.3	Second proposal - auxiliary perturbing . . . . .	20
3.3.1	Idea . . . . .	20
3.3.2	Issues . . . . .	21
3.3.3	Security . . . . .	22

3.4	Third proposal - multipoint assignment . . . . .	25
3.4.1	Idea . . . . .	25
3.4.2	Alternative unblinding . . . . .	27
3.4.3	Issues . . . . .	28
3.4.4	Security . . . . .	28
3.5	Fourth proposal - blinding matrix . . . . .	29
3.5.1	Idea . . . . .	29
3.5.2	The blinding matrix . . . . .	30
3.5.3	Issues . . . . .	31
3.5.4	Security . . . . .	31
3.6	Fifth proposal - a blinding matrix and vector . . . . .	34
3.6.1	Idea . . . . .	34
3.6.2	Issues . . . . .	35
3.6.3	Security . . . . .	35
<b>4</b>	<b>Implementation and results</b>	<b>37</b>
4.1	Implementation details . . . . .	37
4.1.1	Blinding . . . . .	37
4.1.2	Unblinding . . . . .	38
4.2	Used parameters . . . . .	39
4.3	Pretest . . . . .	39
4.4	Efficiency . . . . .	40
4.5	Results . . . . .	40
4.6	Evaluation . . . . .	41
<b>5</b>	<b>Conclusion and future prospects</b>	<b>43</b>
5.1	On this work . . . . .	43
5.2	Future prospects . . . . .	43
<b>A</b>	<b>Implemented methods</b>	<b>45</b>
A.1	createpublic . . . . .	45
A.2	invert . . . . .	45
A.3	createpermute . . . . .	48
A.4	multiply . . . . .	49
A.5	print . . . . .	50
<b>B</b>	<b>Example results</b>	<b>51</b>

# List of Figures

2.1	Signing of a hash value - example in $\mathbb{R}^2$ . . . . .	10
3.1	Blinding operation of the first proposal - example in $\mathbb{R}^2$ . . . . .	16
3.2	Unblinding operation of the first proposal - example in $\mathbb{R}^2$ . . . . .	17
3.3	Blinding operation of the second proposal - example in $\mathbb{R}^2$ . . . . .	20
3.4	Unblinding operation of the second proposal - example in $\mathbb{R}^2$ . . . . .	21
3.5	Example for a invalid signature calculation in $\mathbb{R}^2$ . . . . .	23
3.6	Blinding operation of the third proposal - example in $\mathbb{R}^2$ . . . . .	25
3.7	Unblinding operation of the third proposal - example in $\mathbb{R}^2$ . . . . .	27
3.8	Blinding operation of the fourth proposal - example in $\mathbb{R}^2$ . . . . .	29
3.9	Blinding operation of the fifth proposal - example in $\mathbb{R}^2$ . . . . .	34



# List of Tables

2.1	Basic steps of lattice based signature schemes . . . . .	10
3.1	Steps of a blinding . . . . .	16
3.2	Steps of the first proposal . . . . .	18
3.3	Steps of the second proposal . . . . .	22
3.4	Steps of the third proposal . . . . .	26
3.5	Steps of the fourth proposal . . . . .	30
3.6	Steps of the fifth proposal . . . . .	35
4.1	$p$ = public polynomial, $m$ = message polynomial, $m'$ = blinded message polynomial, $s'$ = corresponding signature for $m'$ , $s''$ = unblinded $s'$ . . . . .	40



# Chapter 1

## Introduction

Digital signatures are used to ensure authentication. The common digital signature schemes are based on asymmetric cryptography with underlying hard problems like *integer factorization* or *discrete logarithm*.

Blind signature schemes are a special form of digital signatures and important for different cases, for example in electronic payment schemes like David Chaums ECash system [Cha83]. Another example is the scenario of electronic voting, whose importance is increasing continuously (an example can be found in [Ohk99]). The features of a blind signature scheme play an important role in both scenarios which depend on *anonymous and secure authentication*. The existing blind signature schemes (see section 1.3 for details) are only modified versions of common digital signature schemes, so they share the same underlying hard problems.

These hard problems will become weak with the development of *quantum computers* (see section 1.2 for details). To avoid the uselessness of classic cryptography in the *post quantum era*, it will be important to find alternative methods based on other hard problems. In the case of blind signatures no alternatives are existing yet. Promising candidates are probably lattice signature schemes which are based on solving hard problems in a lattice (see section 2.1 for details).

This diploma thesis deals with the development of a new blind signature scheme based on lattice signature schemes. The content of the first chapter will clarify the details of the motivation.

## 1.1 A blind signature

A digital signature scheme is used to *sign* documents in such a way that the validity of an authentic signature can be proved by everyone, but it is impossible to forge a valid signature on other documents. This requirement is called *non-forgeability*:

- After getting  $x$  signatures, it is infeasible for the receiver to compute  $x + 1$  signatures.<sup>1</sup>

Blind signature schemes are a special form of signature schemes because they include an additional requirement:

A signer can *sign* a document without knowing its content.

This requirement could be achieved by giving the signer a document which is encrypted or disturbed in some way. The *unlinkability* is an additional security requirement of a digital blind signature:

- No one can derive a link between one of the messages which the signer has received and a valid blind signature, except the signature requester.<sup>2</sup>

A formal definition of digital signatures and digital blind signatures can be found in [JLO97].

## 1.2 Hard problems and quantum computing

A quantum computer is a computation device, which uses the quantum properties of particles to perform operations on data. Instead of using bits to represent data, the quantum computer uses qubits. These qubits allow to use not only binary states, but also the *superposition* of these states. Dedicated quantum algorithms can use this *superposition* or the *entanglement* attributes of these qubits to solve problems with much less operations than a 'classic' computer.

The security of RSA is based on the problem of large integer *factorization*. There is no known algorithm for solving this problem in a polynomial time on an ordinary computer. The *Shor's algorithm* for quantum computers can solve integer *factorization* in a polynomial time. This algorithm can also be used for solving the *discrete logarithm* problem, where public key ciphers like the ElGamal encryption and the Diffie-Hellmann key exchange based upon.

---

<sup>1</sup>definition in dependence on [JLO97]

<sup>2</sup>definition in dependence on [KC99]

Examples of blind signature schemes based on *factorization* or *discrete logarithm* can be found in 1.3.

The development of quantum computers is at the moment in its beginning. There are some positive experiments with a very small number of qubits. It is not sure, if it is even possible to build quantum computers with a significant higher number of qubits, because there are problems to keep the components of the computer in a coherent state. According to the prevailing opinion, it is only a question of time, that one of the technical different candidate models for quantum computing will lead to success. It also seems, that a quantum computing device can't solve every existing problem in polynomial time, which is only solvable in non-polynomial time in the case of 'classic' computing.

## 1.3 Existing blind signature schemes

In this section, common blind signatures are introduced. The RSA signature scheme is based on the problem of integer factorization, while the ElGamal and the Schnorr signature schemes are based on the discrete logarithm problem.

### 1.3.1 Blind RSA signature scheme

One simple, secure and efficient existing blind signature scheme is a modified version of the RSA <sup>3</sup> signature. The traditional RSA signature  $s$  is computed by exponentiating the hash value  $h$  of a message  $m$  with the secret exponent  $d$ , all mod a public modulus  $N$ :

$$s(h(m)) \equiv (h(m))^d \pmod{N}$$

The validity of a signature can be verified by calculating the exponentiation of the signature with the public key  $e$ , all mod the public modulus  $N$  and comparing it to the original hash of the message  $m$ .

$$(s(h(m)))^e \equiv h(m) \pmod{N}$$

The blind version of RSA only adds a random value  $r$ , which character has to be  $\gcd(r, N) = 1$ . The signer receives

$$h'[m] \equiv h[m] * r^e \pmod{N}$$

---

<sup>3</sup>publicly described in 1977, named by its authors surnames Rivest, Shamir and Adleman

without any knowledge about the value of  $r$ . The computation of the blinded signature  $s'$  is analogue to the traditional way:

$$s' \equiv (h'(m))^d \pmod{N}$$

The author of  $m$  can extract the valid signature  $s$  out of  $s'$ :

$$s \equiv s' * r^{-1} \pmod{N}$$

### 1.3.2 Blind ElGamal signature scheme

The blinded version of the ElGamal signature scheme is based on the ElGamal variant I.3 in [HMP94] and has been proposed by Carmenisch, Pivetaeu and Stadler in [CPS94].

A trusted authority chooses large, suitable primes  $p$  and  $q$  with  $q|(p-1)$  and an element  $\alpha \in \mathbb{Z}_{p-1}$

The signer chooses a random  $s_N \in \mathbb{Z}_q$  and computes

$$p_N \equiv \alpha^{s_N} \pmod{p}$$

He keeps  $s_N$  secret and publishes the value  $p_N$ .

The signer also generates a random number  $\tilde{k} \in \mathbb{Z}_q$  and computes

$$\tilde{r} \equiv \alpha^{\tilde{k}} \pmod{p}$$

Now he sends  $\tilde{r}$  to the author of a message  $m$ , who wants to receive a signature his message.

The author chooses two random numbers  $a, b \in \mathbb{Z}_q$  and computes

$$r \equiv \tilde{r}^a \alpha^b \pmod{p}$$

and

$$\tilde{m} \equiv am\tilde{r}r^{-1} \pmod{q}$$

He sends  $\tilde{m}$  to the signer, who is able to calculate the signature  $\tilde{s}$  for the blinded message  $\tilde{m}$  by computing

$$\tilde{s} \equiv s_N\tilde{r} + \tilde{k}\tilde{m} \pmod{q}$$

The signature  $\tilde{s}$  will now be sent to the owner of the message, who can unblind this signature by calculating

$$s \equiv \tilde{s}r\tilde{r}^{-1} + bm \pmod{q}$$

The verification of the signature can be done by proofing the following equation

$$\alpha^s \equiv p_N^r r^m \pmod{p}$$

### 1.3.3 Blind Schnorr signature scheme

The Schnorr signature scheme [Sch90] can also be turned into a blind signature scheme [Poi96]. A trusted authority chooses large, suitable primes  $p$  and  $q$  with  $q|(p-1)$  and an element  $g$  of  $(\mathbb{Z}/p\mathbb{Z})^*$  of order  $q$ .

The signer generates a pair of keys,  $x \in \mathbb{Z}/q\mathbb{Z}$  and

$$y \equiv g^{-x} \pmod{p}$$

where  $y$  is a public value. He also generates a random  $k \in \mathbb{Z}/q\mathbb{Z}$  and computes

$$r \equiv g^k \pmod{p}$$

which is sent to the author of a message  $m$ . The message author generates two random values

$$\alpha, \beta \in \mathbb{Z}/q\mathbb{Z}$$

for calculating

$$r' \equiv r g^{-\alpha} y^{-\beta} \pmod{p}$$

and  $e' = H(m, r')$ , where  $H$  is a public hash function. After this, he sends a challenge  $e$  to the signer

$$e = e' + \beta$$

The signer returns a value  $s$  such that

$$g^s y^e \equiv r \pmod{p}$$

One can calculate  $s' \equiv s - \alpha \pmod{q}$ , so it is easy to proof, if  $(e', s')$  is a valid signature of  $m$  by doing the following equation

$$e' = H(m, g^{s'} y^{e'} \pmod{p})$$

### 1.3.4 Other blind signatures

The blind signature schemes above are three common blind signature schemes. There are a lot of different schemes, which are in most cases basing on the integer factorization or discrete logarithm problem. Examples can be found in [CPS94], [HMP94], [Poi96] or [JL96].

## 1.4 Organization

This thesis is organized as following: The second chapter leads into details, variants and security issues of lattice based signature schemes. The research

## *CHAPTER 1. INTRODUCTION*

---

process and the proposals for the new blind signature scheme are shown in the third chapter. The following chapter contains details about the implementation of a blind lattice signature scheme and its evaluation. The last chapter deals with the conclusion on this work and the prospects for future work.

# Chapter 2

## Lattice based signature schemes

This chapter deals at first with the mathematical basics of lattices. The underlying idea and the different versions of lattice signatures are presented in the second section. After this, the security of the existing lattice signature schemes will be discussed in the third section.

### 2.1 Formal basics about lattices

#### 2.1.1 Lattices

**Definition 1:** A lattice.

A lattice  $L(B)$  is the set of all possible integral linear combinations of the vectors  $\vec{b}_i$  from the vector set  $B = \{\vec{b}_1, \dots, \vec{b}_n\}$  in  $\mathbb{R}^n$ :

$$L(B) = \left\{ \sum k_i \vec{b}_i : k_i \in \mathbb{Z} \text{ for all } i \right\}$$

The vector set  $B$  is called the basis of  $L(B)$ . A vector  $\vec{v}$  is called a lattice vector or lattice point if it belongs to the lattice  $L$ . Typically the basis  $B$  for a lattice in  $\mathbb{R}^n$  is given as a  $n \times n$  matrix, which is non-singular. The columns of the matrix  $B$  represent the basis vectors  $\vec{b}_i$ , so

$$L(B) = \{Bv : v \in \mathbb{Z}\}$$

would be the definition of the lattice  $L(B)$ . This representation of a lattice basis allows to show the infinite number of different bases for the same lattice. If we take a vector  $\vec{b}_i$  of our basis  $B$ , we can produce another basis by adding any integral linear combination of the other vectors  $\{\vec{b}_1, \dots, \vec{b}_n\}$ .

### 2.1.2 Qualities of lattices

**Definition 2:** Orthogonality defect. (orth-defect)

The orthogonality defect of the non-singular  $n \times n$  matrix  $B$  is defined as

$$\text{orth} - \text{defect}(B) = \frac{\prod_i \|\vec{b}_i\|}{\det(B)}$$

where  $\|\vec{b}_i\|$  is the Euclidean norm of the  $i$ 'th column in  $B$ .

**Definition 3:** The dual lattice.

In a basis matrix  $B$  of a lattice  $L(B)$  in  $\mathbb{R}^n$ , the columns  $\vec{b}_i$  are the basic lattice vectors. The *dual lattice* of  $L$  is the lattice, which is spanned by the rows of matrix  $B^{-1}$ . These rows are called  $\vec{b}'_i$  in the following.

**Definition 4:** The dual orthogonality defect.

The *dual orthogonality defect* of a matrix of a real non-singular matrix  $B$  is defined as:

$$\text{dual} - \text{orth} - \text{defect}(B) = \frac{\prod_i \|\vec{b}'_i\|}{\det(B^{-1})}$$

where  $\|\vec{b}'_i\|$  is the Euclidean norm of the  $i$ 'th row in  $B^{-1}$ . Using the properties between  $B$  and the inverse matrix  $B^{-1}$ , this definition can also be denoted by:

$$\text{dual} - \text{orth} - \text{defect}(B) = \prod_i \|\vec{b}'_i\| \det(B)$$

### 2.1.3 Hard problems in lattices

**The Closest Vector Problem (CVP).**

If there is given a vector  $\vec{v} \in \mathbb{R}^n$  and a lattice basis  $B$  in  $\mathbb{R}^n$ , it is difficult to find the lattice point in  $L(B)$ , which is the closest to  $\vec{v}$  in a prior defined norm. The CVP was shown to be  $NP$  hard, so there is no polynomial-time algorithm for solving the CVP.

**The Smallest Basis Problem (SBP).**

If there is given a basis  $B$  for a lattice in  $\mathbb{R}^n$ , it is difficult to find the *smallest* basis  $B'$  (in some norm) for the same lattice. The important norm in the case of cryptographic usage is the size of the *orthogonality defect*. There is currently no polynomial-time algorithm for solving the SBP. <sup>1</sup>

---

<sup>1</sup>further informations on cryptographic relevant hard problems in lattices are given in [GGH96]

## 2.2 Lattice based signature schemes

### 2.2.1 Idea of a lattice based signature scheme

It is easy for any given basis to find a vector, which is close to a lattice point. This could be done by choosing a lattice point and adding a small error vector. From this new point, it is not trivial to return to the original lattice point only with the knowing of a random lattice basis. So this operation can be seen as a form of one-way computation.

This one-way computation also offers a trapdoor functionality. Different bases for the same lattice in  $\mathbb{R}^n$  seem to differ in the difficulty of the ability to find the closest lattice point for a random vector, thus to solve the CVP.

The difference between those bases is set in different values of the *dual orthogonality defect*. The basis which allows a good approximation of the CVP has a significant small dual orthogonality defect and is called the *reduced basis*. After the determination of the lattice dimension  $n$ , the reduced basis is chosen uniformly, while the non-reduced basis is derived from the original reduced basis by using a randomised transformation.

The input of the function will be the sum of a lattice point  $\vec{v}$  and a error vector  $\vec{e}$ . The length of  $\vec{e}$  is upper-bounded by a formerly specified threshold  $\tau$ . With the usage of the *reduced basis*, it is possible to approximate the solution of finding the lattice point which is at most  $\tau$  away from the given vector.

In the case of a lattice based signature, a message could be represented as a hash value, which is a point in  $\mathbb{R}^n$ . The reduced basis - now called  $R$  - has to be kept secret, because it contains the trapdoor information. The non-reduced basis - now called  $B$  - is the public element of this signature scheme.

It contains the following steps:

Step 1: The *signer* hashes his message  $m$  with a special form of hash function, which delivers  $h(m)$ , a point in  $\mathbb{R}^n$ .

Step 2: With the use of the secret basis  $R$ , the *signer* can calculate the closest lattice point to the hash value  $h(m)$ . This point of the lattice  $L(B)$  is the signature of the message  $m$ , called  $s(m)$ . The *signer* will now send the message  $m$  and its signature  $s(m)$  to the receiver.

Step 3: The receiver has also to calculate the hash value of  $m$ .

Step 4: He can verify the validity of  $s(m)$  by proofing the distance between  $h(m)$  and  $s(m)$  with the public basis  $B$ . If the distance is smaller than the specified threshold  $\tau$  the signature is valid.

The steps are shown as an overview in table 2.1.

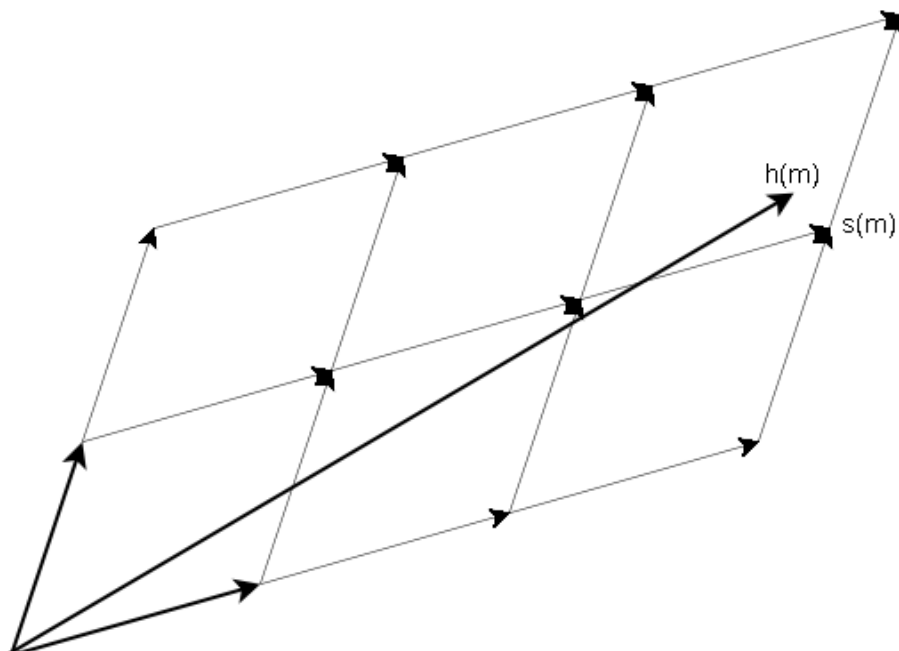


Figure 2.1: Signing of a hash value - example in  $\mathbb{R}^2$

step	signer		receiver
1	$h(m)$ out of $m$		
2	$s(m)$ out of $h(m)$ with $R$	$\longrightarrow$	$m$ and $s(m)$
3			$h(m)$ out of $m$
4			verifies $s(m)$ with $B$

Table 2.1: Basic steps of lattice based signature schemes

A very important point is the special nature of lattice based signature schemes:

If two vectors  $\vec{v}, \vec{v}'$  in  $\mathbb{R}^n$  are significant closer to each other than the threshold  $\tau$ , it is very likely that a signature on  $\vec{v}$  could also be a valid signature for  $\vec{v}'$ . So the signature scheme has to avoid, that a signature on a message could also be used to sign another message. This could be done by a strong hash function, that will hash two similar messages <sup>2</sup> far apart in  $\mathbb{R}^n$ .

---

<sup>2</sup>with only a slightly difference

### 2.2.2 Goldreich, Goldwasser and Halevi (1996)

In 1996, the first proposal of a lattice based public-key cryptosystem was released by Oded Goldreich, Shafi Goldwasser and Shai Halevi in [GGH96]. Their idea for a digital signature design is simply called *GGH* in the following.

The first part of GGH is the generation of the basis by the *GENERATE* algorithm. By an input of  $n$ , it generates two bases  $R$  and  $B$  of the same full-rank lattice in  $\mathbb{Z}^n$  and the threshold  $\sigma^3$ , which is a positive real number.  $R$  will be the private basis with a low dual-orthogonality defect and  $B$  will be the public basis with a high dual-orthogonality defect. Both are  $n \times n$  matrices, where the basis vectors of the lattice are the columns of the matrices. To generate the private basis, it is possible to choose random  $n$  vectors in  $\mathbb{Z}^n$  and mix them to generate the public basis. The public key is now the tuple  $(B, \sigma)$  and the trapdoor information is the private basis  $R$ .

The *GGH* scheme was shown as secure for high dimensions at the time, but it was not very efficient with a keysize of  $O(n^3 \log n)$  and a computation time of  $O(n^2 \log n)$ . With the existence of very good competitors like RSA and especially modern hybrid schemes, the development of lattice based cryptography was not in the focus of research.

### 2.2.3 Micciancio (2001)

In 2001, Daniele Micciancio developed a way to increase the performance with the use of the Hermite Normal Form (HNF)<sup>4</sup> in a GGH-like trapdoor function, of which GGH is a special case. This improvement allows to reduce the key size from  $O(n^3 \log n)$  to  $O(n^2 \log n)$  and also the computation time from  $O(n^2 \log n)$  to  $O(n \log n)$  without taking an influence to the security of the scheme. Any attack on Micciancios scheme can be provably transformed into an equally attack against the original GGH scheme. Further informations on this scheme are given in [Mic01].

### 2.2.4 NTRUSIGN (2003)

In 2003, NTRU Cryptosystems introduced NTRUSIGN, a signature scheme which is based on their own public-key cryptosystem NTRUENCRYPT. NTRUSIGN uses the same form of NTRU lattices like NTRUENCRYPT.

The ring  $R = \mathbb{Z}[X]/(X^N - 1)$  is the underlying mathematics of the NTRU lattice.

---

<sup>3</sup>the threshold  $\sigma$  means the same as threshold  $\tau$

<sup>4</sup>the public basis  $B$  is the HNF of private basis  $R$

The matrix

$$\begin{pmatrix} r_0 & r_1 & \dots & r_{N-1} \\ r_{N-1} & r_0 & \dots & r_{N-2} \\ \vdots & \vdots & \ddots & \vdots \\ r_1 & r_2 & \dots & r_0 \end{pmatrix}$$

is associated with any element of ring  $R$

$$r = \sum_{i=0}^N r_i X^i \in R$$

and is part of the matrix, which rows generates the convolution modular lattice  $L_R$ :

$$\begin{pmatrix} 1 & 0 & \dots & 0 & r_0 & r_1 & \dots & r_{N-1} \\ 0 & 1 & \dots & 0 & r_{N-1} & r_0 & \dots & r_{N-2} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 & r_1 & r_2 & \dots & r_0 \\ 0 & 0 & \dots & 0 & q & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 & 0 & q & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0 & 0 & 0 & \dots & q \end{pmatrix}$$

The basic operations of the polynomials are addition and convolution multiplication, where the convolution multiplication  $*$  of two polynomials  $f$  and  $g$  is defined by taking the coefficient  $X^k$  in  $f * g$  to equal

$$(f * g)_k \equiv \sum_{i+j \equiv k \pmod{N}} f_i * g_j \quad (0 \leq k \leq N)$$

Quoted from [NTRU03], the basic operations of NTRUSIGN are as follows:

*Key Generation* - requires a source of (pseudo)random bits.

1. INPUT: Integers  $N, q, d_f, d_g, B \geq 0$ , and the string  $t =$  'standard' or 'transpose'.
2. GENERATE  $B$  PRIVATE LATTICE BASES AND ONE PUBLIC LATTICE BASIS: Set  $i = B$ . While  $i \geq 0$ :
  - (a) Randomly choose  $f, g \in R$  to be binary<sup>5</sup> with  $d_f, d_g$  ones, respectively.

---

<sup>5</sup>a convolution is binary, if one of the polynomials contains only coefficients from the set  $\{0, 1\}$

(b) Find small  $F, G \in R$  such that  $f * G - F * g = q$ . Sections 3 to 4.2 in [NTRU03] give more details.

(c) If  $t = \text{'standard'}$ , set  $f_i = f$  and  $f'_i = F$ . If  $t = \text{'transpose'}$ , set  $f_i = f$  and  $f'(i) = g$ . Set  $h_i = f_i^{-1} \bmod q$ . Set  $i = i - 1$ .

3. PUBLIC OUTPUT: The input parameters and  $h = h_0 \equiv f_0^{-1} * f'_0 \bmod q$ .

4. PRIVATE OUTPUT: The public output and the set  $f_i, f'_i, h_i$  for  $i = 0 \dots B$ .

*Signing* - requires a hash function  $H : \mathcal{D} \rightarrow R$  on a digital document space  $\mathcal{D}$ . Signing also requires a norm function  $\|\cdot\| : R^2 \rightarrow \mathbb{R}$  and a 'norm bound'  $N \in \mathbb{R}$ . For  $(s, t) \in R^2$  we define  $\|(s \bmod q, r \bmod q)\|$  to be the minimal value of  $\|(s + k_1q, r + k_2q)\|$  for  $k_1, k_2 \in R$ .

1. INPUT: A digital document  $D \in \mathcal{D}$  and the private key  $f_i, f'_i, h_i$  for  $i = 0 \dots B$ .

2. Set  $r = 0$ .

3. Set  $s = 0$ . Set  $i = B$ . Encode  $r$  as a bit string. Set  $m_0 = H(D\|r)$ , where ' $\|$ ' denotes concatenation. Set  $m = m_0$ .

4. PERTURB THE POINT USING THE PRIVATE LATTICES: While  $i \geq 1$ :

(a) Set  $x = \lfloor -(1/q)m * f'_i \rfloor$ ,  $y = \lfloor (1/q)m * f_i \rfloor$ ,  $s_i = x * f_i + y * f'_i$ .

(b) Set  $m = s_i * (h_i - h_{i-1}) \bmod q$ .

(c) Set  $s = s + s_i$ . Set  $i = i - 1$ .

5. SIGN THE PERTURBED POINT USING THE PUBLIC LATTICE:

Set  $x = \lfloor -(1/q)m * f'_0 \rfloor$ ,  $y = \lfloor (1/q)m * f_0 \rfloor$ ,  $s_0 = x * f_0 + y * f'_0$ ,  $s = s + s_0$ .

6. CHECK THE SIGNATURE:

(a) Set  $b = \|(s, s * h - m_0 \bmod q)\|$ .

(b) If  $b \geq \mathcal{N}$ , set  $r = r + 1$  and go to step 3.

7. OUTPUT: The triplet  $(D, r, s)$ .

*Verification* - requires the same hash function  $H$ , norm function  $\|\cdot\|$  and 'norm bound'  $\mathcal{N} \in \mathbb{R}$ .

1. INPUT: A signed document  $(D, r, s)$  and the public key  $h$ .

2. Encode  $r$  as a bit string. Set  $m = H(D\|r)$ .

3. Set  $b = \|(s, s * h - m \bmod q)\|$ .

4. OUTPUT: 'valid' if  $b < (\mathcal{N})$ , 'invalid' otherwise.

where the norm function  $\|\cdot\| : R^2 \rightarrow \mathbb{R}$  is the *centered Euclidean norm*. The *centered Euclidean norm* of a polynomial  $r$  is calculated the following way:

$$\|r\| = \sqrt{\sum_{i=0}^{N-1} r_i^2 - (1/N)(\sum_{i=0}^{N-1} r_i)^2}$$

## 2.3 Security

At the EUROCRYPT 2006 Phong Q. Nguyen and Oded Regev presented their paper [HPP06] about the cryptanalysis of GGH and NTRU signature. They presented a new key recovery attack on these signature schemes, which uses the knowledge that a list of known pairs (*messages, signatures*) will leak some information about the lattice and its secret key. They introduced a problem they call the *hidden parallelepiped problem (HPP)*: Many given random points over an unknown n-dimensional parallelepiped, recover the parallelepiped or an approximation thereof.

They used this problem to recover the secret key by collecting pairs of messages and signatures. In case of NTRUSIGN-251 with standard parameters they needed only about 90.000 pairs to recover the secret key. They also attacked the GGH signature scheme successfully, using a number of signatures which is roughly quadratic in the lattice dimension.

In november 2006 Phong Q. Nguyen published a new paper [Ngu06], where he shows an efficiency increasing of this attack. With the use of existing symmetries of the NTRUSIGN parallelepipeds the number of needed signatures could be significantly decreased. In case of NTRUSIGN-251 only about 400 signatures are needed to recover the secret key.

This attack works only on NTRUSIGN without perturbation techniques. The use of perturbations is recommended in [NTRU03] and is part of the engineering specification, also shown in 2.2.4.

# Chapter 3

## Proposals

In this chapter our proposals to accomplish a new blind signature scheme based on lattices are presented. Also the problems related to these proposals are discussed.

The initial idea and the first proposal are suggestions of Prof. Dr. Johannes Buchmann which are shown in the first two sections. The following two sections show the proposals derived from the evaluation process of their predecessors. Sections five and six show two proposals based on a different idea suggested by Prof. Dr. Jintai Ding.

### 3.1 General idea for a lattice blind signature scheme

The following steps contain the basic idea of converting a lattice signature scheme into a blind lattice signature scheme:

Step 1: The message in lattice signature schemes is always represented by a hash value, which is a vector  $h(m) \in \mathbb{R}^n$ .

Step 2: A possible option to blind this message is to move it to another position in  $\mathbb{R}^n$  by an abstract blinding function  $p : \mathbb{R}^n \rightarrow \mathbb{R}^n$  to obtain a new vector  $h'(m)$ .

Step 3: This new vector  $h'(m)$  will now be sent to the *signer*. The signer calculates the signature  $s'(m)$  for  $h'(m)$  and sends it back to the author.

Step 4: The author uses the unblinding function, which is a reverse of  $p$  called  $p^{-1} : \mathbb{R}^n \rightarrow \mathbb{R}^n$ , to calculate the valid signature  $s(m)$  by computing  $p^{-1}(s'(m))$ .

The steps are shown as an overview in table 3.1.

step	Blinder		Signer
1	$h(m)$ out of $m$		
2	$p(h(m))$	$\longrightarrow$	$h'(m)$
3	$s'(m)$	$\longleftarrow$	$s'(m)$ for $h'(m)$
4	$p^{-1}(s'(m)) = s(m)$		

Table 3.1: Steps of a blinding

## 3.2 First proposal - adding a random lattice vector

The first proposal is based on the idea of adding a vector as a blind factor.

### 3.2.1 Idea

The first proposal contains a quite simple idea. The person who demands the sign from the *signer*, is now called the *blinder* in the following.

Step 1: The *blinder* hashes his message  $m$ .

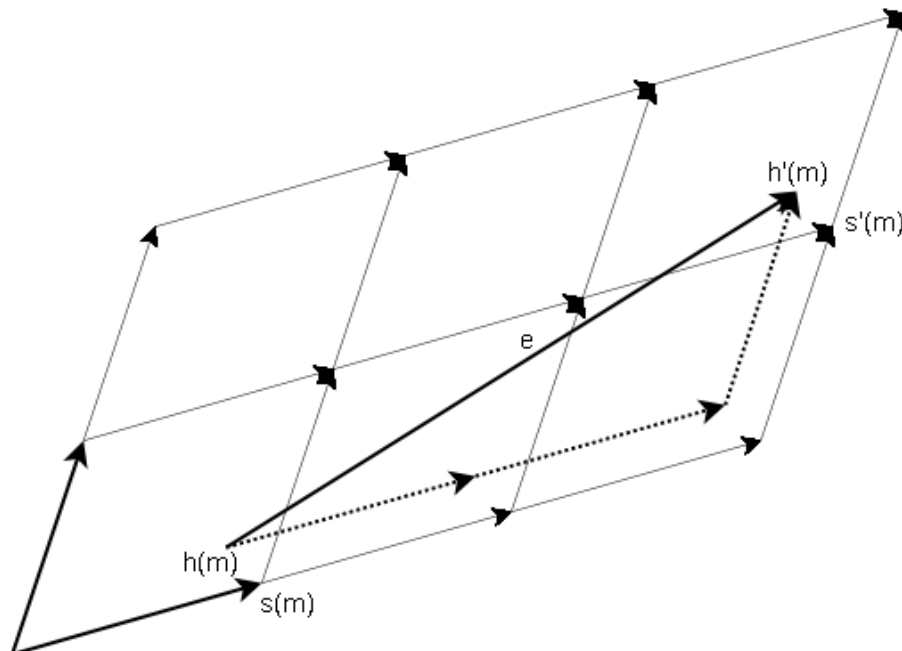


Figure 3.1: Blinding operation of the first proposal - example in  $\mathbb{R}^2$

Step 2: The result  $h(m)$  is a point in  $\mathbb{R}^n$ . He chooses a random linear

combination  $\vec{e}$  of the basis vectors of the lattice  $L(B)$ . This linear combination, the *blinding vector* must have integral coefficients and has to be chosen uniformly distributed.

The *blinder* adds the random vector  $\vec{e}$  to  $h(m)$  and obtains a new point in  $\mathbb{R}^n$ , which is now called the *blinded hash*  $h'(m)$ . This blinded  $h'(m)$  is the value, which will be sent to the *signer*.

Step 3: The *signer* calculates the correct signature  $s'(m)$  for the received  $h'(m)$  and sends it to the *blinder*.

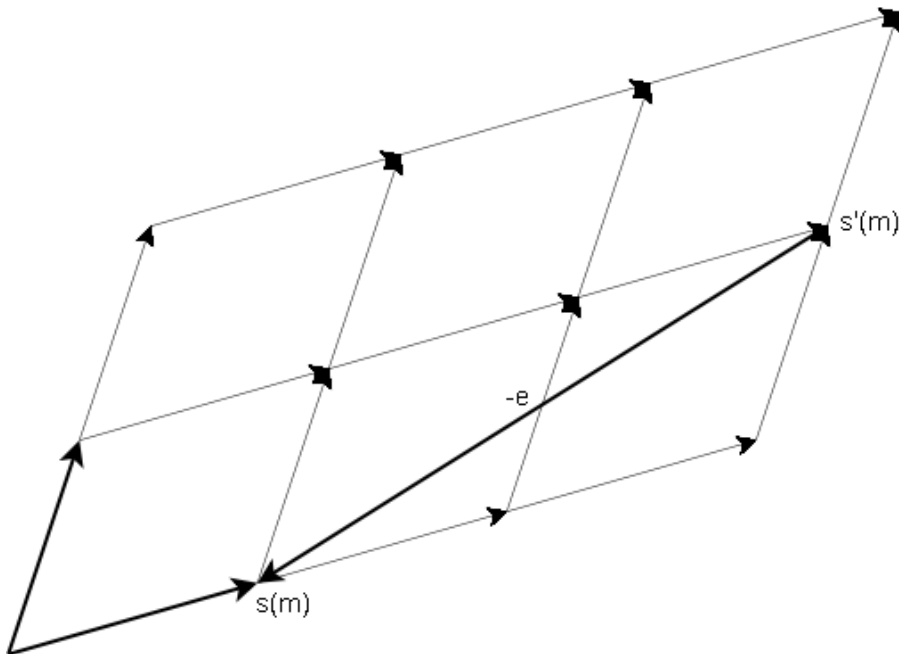


Figure 3.2: Unblinding operation of the first proposal - example in  $\mathbb{R}^2$

Step 4: Now the *blinder* is able to calculate the signature  $s(m)$  for the original hash value  $h(m)$  by subtracting the blinding vector  $\vec{e}$  from the signature  $s'(m)$  of the blinded hash value  $h'(m)$ .

The steps are shown as an overview in table 3.2.

### 3.2.2 Security

Two different attacks on this proposal were developed on its evaluation process.

step	Blinder		Signer
1	$h(m)$ out of $m$		
2	$h(m) + \vec{e}$	$\longrightarrow$	$h'(m)$
3	$s'(m)$	$\longleftarrow$	$s'(m)$ for $h'(m)$
4	$s'(m) - \vec{e} = s(m)$		

Table 3.2: Steps of the first proposal

**Attack 1**

There is an possible attack against this proposal, which enables the signer to link between the blinded and the unblinded messages. The signer can save all the values, that he received in step 2. As the result, he gets a list of blinded values

$$\{h'_1(m), \dots, h'_n(m)\}$$

in which  $n$  is the number of hash values.

If the signer also receives the final messages signed with the unblinded signatures, he can also build a list of those original hash values:

$$\{h_1(m), \dots, h_n(m)\}$$

In the case, that the *signer* is not the receiver of the signed messages, he maybe can see them somewhere in the complete process, so the building of this list can not be excluded.

Now, the signer can find possible linkings by taking one item of a list and calculating the difference between it and all the items of the other list:

$$\{h_1(m) - h'_1(m), \dots, h_1(m) - h'_n(m)\}$$

The result is a list of vectors.

If one of this vectors is a lattice vector, the attacker may have found a link. But for all the vectors, which are no lattice vectors, he can surely say, that the corresponding pair of hash values is not linked.

The chance, that the difference from a randomly chosen pair is accidently a lattice vector, is very small. So the attacker has a very good chance to find all the links by simply comparing his two lists.

The complexity of this attack depends only on the number of values. There is a very high chance to find the first link in  $O(n)$ .

**Attack 2**

Another similar attack is thinkable:

### 3.2. FIRST PROPOSAL - ADDING A RANDOM LATTICE VECTOR

---

The *signer* can save the pairs of blinded hashes and their corresponding signatures

$$\{(h'_1(m), s'_1(m)), \dots, (h'_n(m), s'_n(m))\}$$

to calculate the length of the error vectors belonging to each pair:

$$\{\|h'_1(m) - s'_1(m)\|, \dots, \|h'_n(m) - s'_n(m)\|\}$$

The *signer* can also save the pairs of unblinded hashes and their corresponding signatures

$$\{(h_1(m), s_1(m)), \dots, (h_n(m), s_n(m))\}$$

to calculate the length of the error vectors belonging to each unblinded pair, too:

$$\{\|h_1(m) - s_1(m)\|, \dots, \|h_n(m) - s_n(m)\|\}$$

Now the *signer* can compare the two error vector lists to find matches. If an error vector from the first list is different to one from the second, he can exclude a possible linking between them.

The complexity of this attack depends only on the number of values. There is a very high chance to find the first link in  $O(n)$ .

### 3.3 Second proposal - auxiliary perturbing

A possible countermeasure against the attacks for the first proposal would probably be the *perturbing* of the hash value. Perturbing means to displace the correct value by a little error value.

#### 3.3.1 Idea

In this proposal, the idea is to add a small error vector, which is not a lattice vector to prevent the attacker from finding links with the method shown above.

Step 1: The *blinder* hashes his message  $m$  and receives  $h(m)$ .

Step 2: He adds a uniformly randomly chosen lattice vector  $\vec{e}$  to  $h(m)$ .

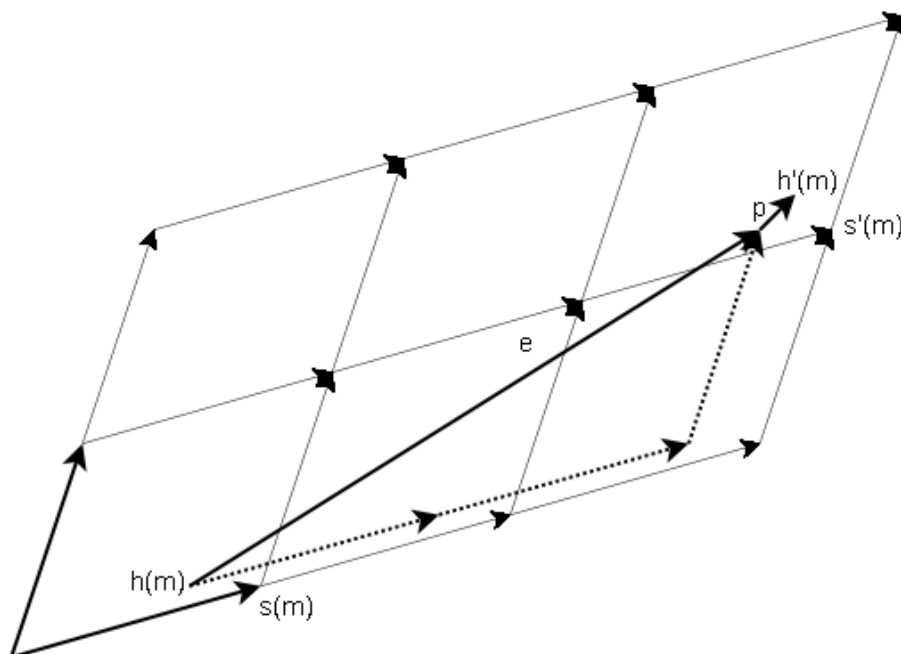


Figure 3.3: Blinding operation of the second proposal - example in  $\mathbb{R}^2$

The *blinder* has now to choose a perturbing vector  $\vec{p}$ . This vector  $\vec{p}$  has to be

- chosen uniformly randomly in its direction
- chosen uniformly randomly in its length, limited by a previously chosen or a given upper bound.

$\vec{p}$  will now be added to the result from step 2, so the *blinder* will receive the final blinded value  $h'(m)$ :

$$h'(m) = h(m) + \vec{v} + \vec{p}$$

Step 3: The blinded value  $h'(m)$  will be sent to the *signer*.

Step 4: The *signer* calculates the correct signature  $s'(m)$  for the received  $h'(m)$  and sends it to the *blinder*.

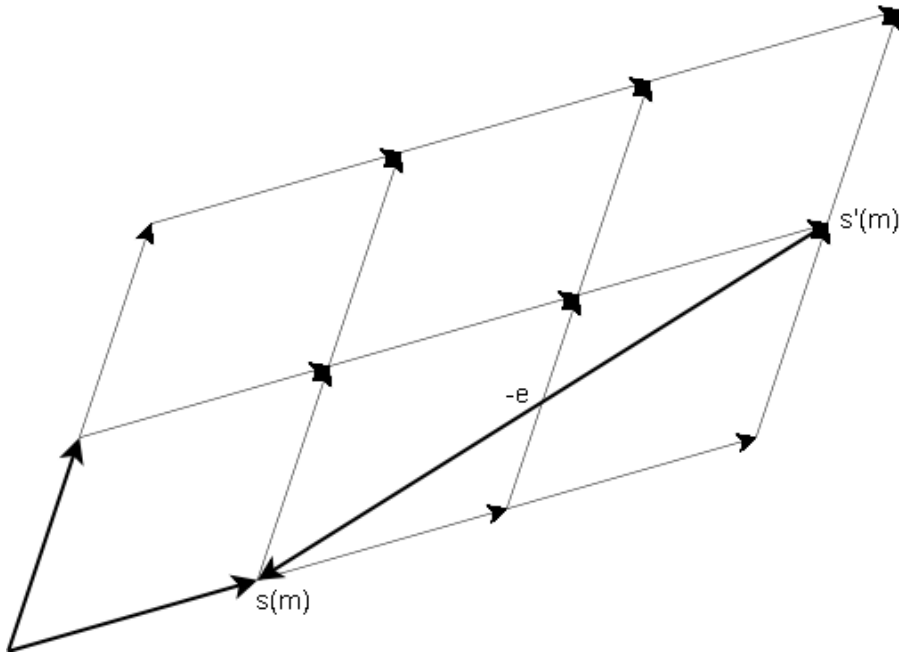


Figure 3.4: Unblinding operation of the second proposal - example in  $\mathbb{R}^2$

Step 5: Now the *blinder* is able to calculate the signature  $s(m)$  for the original hash value  $h(m)$  by subtracting only the blinding vector  $\vec{e}$  (and not the perturbing vector  $\vec{p}$ ) from the signature  $s'(m)$  of the blinded hash value  $h'(m)$ .

The steps are shown as an overview in table 3.3.

### 3.3.2 Issues

There is an issue in constructing the perturbation function: If a hash value  $h(m)$  is near the border of two adjacent areas with different corresponding signatures, it is thinkable that adding the perturbation vector  $\vec{p}$  can cause validity problems. The addition of  $\vec{p}$  can move the point out of the valid

step	Blinder		Signer
1	$h(m)$ out of $m$		
2	$h(m) + \vec{e}$		
3	$h(m) + \vec{e} + \vec{p} = h'(m)$	$\longrightarrow$	$h'(m)$
4	$s'(m)$	$\longleftarrow$	$s'(m)$ for $h'(m)$
5	$s'(m) - \vec{e} = s(m)$		

Table 3.3: Steps of the second proposal

signature area, so the signer would calculate a different signature  $s'(m)$  than expected by the blinder.

When the blinder receives the  $s'(m)$  from the signer, he calculates  $s(m)$  by subtracting  $\vec{e}$  from it. The blinder has no knowledge about the invalidity of his calculated signature.

A way to minimize this failure, would be to choose a very small upper bound for the length of the perturbing vector  $\vec{p}$ . Changing its direction in a way, that holds up the right signature, is not possible at all, because the blinder doesn't know which direction he has to choose.

Unfortunately a small upper bound will be an advantage for the possible attack shown below.

### 3.3.3 Security

There is similar attack to the one for the first proposal thinkable.

The signer can save all the values, that he received in step 3. As the result, he gets a list of blinded values

$$\{h'_1(m), \dots, h'_n(m)\}$$

where  $n$  is the number of hash values.

The *signer* can also build a list of those original hash values

$$\{h_1(m), \dots, h_n(m)\}$$

As a difference to the attack 1 for the first proposal, the signer has now to build two other lists. At first, he can take all the calculated blinded signatures  $s'(m)$  and build a list

$$\{s'_1(m), \dots, s'_n(m)\}$$

And he can construct a list of all the signature values  $s(m)$  according to hash values  $h(m)$  in the form

$$\{s_1(m), \dots, s_n(m)\}$$

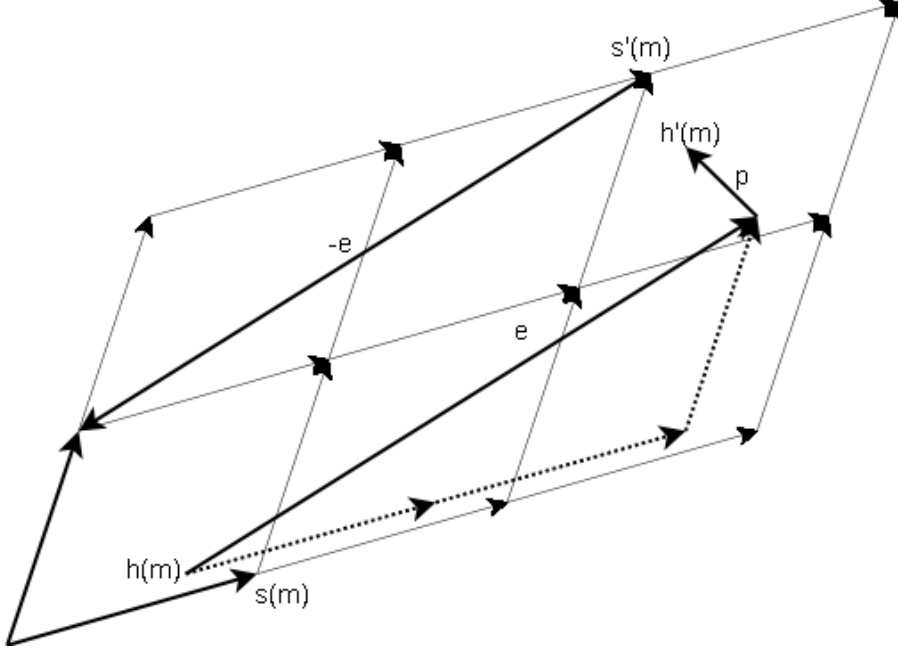


Figure 3.5: Example for a invalid signature calculation in  $\mathbb{R}^2$

Now he can execute the following calculation

$$\|(h_i(m) - h'_j(m)) - (s_i(m) - s'_j(m))\|$$

In the case, that the signer has chosen randomly the matching pair of unblinded and blinded values, the result of this calculation will be  $\|\vec{p}\|$ , because

$$h_i(m) - h'_j(m) = \vec{e} + \vec{p} \quad \text{and} \quad s_i(m) - s'_j(m) = \vec{e}$$

so it is

$$\|(h_i(m) - h'_j(m)) - (s_i(m) - s'_j(m))\| = \|\vec{e} + \vec{p} - \vec{e}\| = \|\vec{p}\|$$

For a not matching pair, the result will also be a length. If the values of this length are set in a different codomain, an attack will be possible.

We can do the following equation

$$\|(h_i(m) - h'_j(m)) - (s_i(m) - s'_j(m))\| = \|(h_i(m) - s_i(m)) - (h'_j(m) - s'_j(m))\|$$

The terms  $\|h_i(m) - s_i(m)\|$  and  $\|h'_j(m) - s'_j(m)\|$  contain both a hash value minus the corresponding valid signature value. This fact will ensure, that the value of this terms will be smaller or equal than the threshold  $\tau$ :

$$\|h_i(m) - s_i(m)\| \leq \tau \wedge \|h'_j(m) - s'_j(m)\| \leq \tau$$

So it is

$$\|(h_i(m) - s_i(m)) - (h'_j(m) - s'_j(m))\| \leq 2\tau$$

As explained in 3.3.2, the upper bound of  $\|\vec{p}\|$  has to be very small to minimize the chance of getting a invalid signature value. It has to be so small, that the chance of leaving the corresponding area around a lattice point with radius  $\tau$  is smaller than a acceptable plausibility. So  $\|\vec{p}\|$  has to be significantly smaller than the threshold  $\tau$ . This fact allows a possible attack.

The attacker can exclude many non-corresponding pairs of blinded and unblinded values, if the result of the inequation is

$$\|\vec{p}\| \leq \|(h_i(m) - h'_j(m)) - (s_i(m) - s'_j(m))\| \leq 2\tau$$

Increasing the length of  $\vec{p}$  will ensure the security of the blinding, but will produce unacceptable failures in the scheme by generating invalid signatures.

### 3.4 Third proposal - multipoint assignment

In the third proposal, the main idea is to calculate and send a set of lattice points instead of sending only one.

#### 3.4.1 Idea

To avoid the issues in security and construction of the second proposal, the basic idea is now to give the *blinder* more than one lattice point, so he can choose always the valid one.

Step 1: The *blinder* hashes his message  $m$  and receives  $h(m)$ .

Step 2: He adds a uniformly randomly chosen lattice vector  $\vec{e}$  to  $h(m)$ .

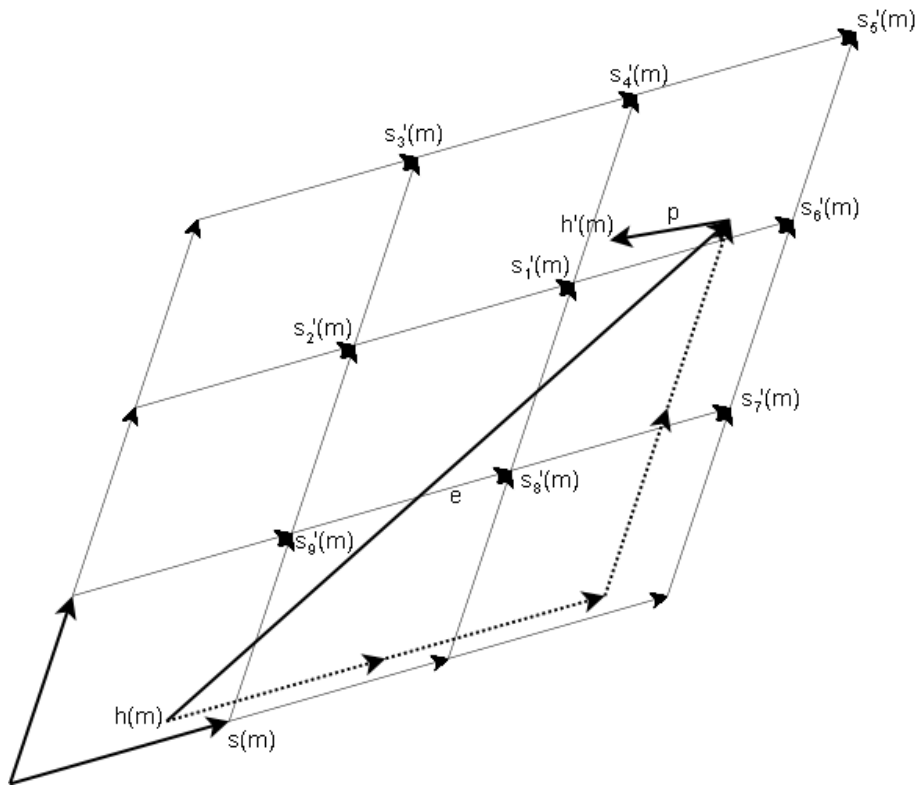


Figure 3.6: Blinding operation of the third proposal - example in  $\mathbb{R}^2$

Step 3: The *blinder* has now to choose a perturbing vector  $\vec{p}$ . This vector  $\vec{p}$  has to be

- chosen uniformly randomly in its direction

- chosen uniformly randomly in its length, limited by the upper bound  $2\tau$ <sup>1</sup>.

$\vec{p}$  will now be added to the result from step 2, so we receive the final blinded value  $h'(m)$ :

$$h'(m) = h(m) + \vec{v} + \vec{p}$$

The blinded value  $h'(m)$  will be sent to the *signer*.

Step 4: The *signer* calculates the correct signature  $s'(m)$  for the received  $h'(m)$ . He also has to find the adjacent lattice points, which are the next to the signature  $s'(m)$ . The number of adjacent lattice points depends on the dimension of the lattice. See the issues section 3.4.3 for more details.

The result is a list of lattice points

$$\{s'_1(m), \dots, s'_i(m)\}$$

The complete list will now be sent to the *blinder*.

Step 5: The *blinder* has to calculate the distances between the value  $h'(m) - \vec{p}$  and all the items from the received list

$$\|h'(m) - \vec{p} - s'_i(m)\|$$

The smallest result indicates the lattice point, which is the valid blinded signature value  $s'(m)$  for the blinded hash without the *perturbation*.

Step 6: Now the *blinder* is able to calculate the signature  $s(m)$  for the original hash value  $h(m)$  by subtracting only the blinding vector  $\vec{e}$  (and not the perturbing vector  $\vec{p}$ ) from the signature  $s'(m)$  of the blinded hash value  $h'(m)$ .

The steps are shown as an overview in table 3.4.

step	Blinder		Signer
1	$h(m)$ out of $m$		
2	$h(m) + \vec{e}$		
3	$h(m) + \vec{e} + \vec{p} = h'(m)$	$\longrightarrow$	$h'(m)$
4	$\{s'_1(m) \dots s'_i(m)\}$	$\longleftarrow$	$\{s'_1(m) \dots s'_i(m)\}$ out of $h'(m)$
5	$s'(m)$ out of $\{s'_1(m) \dots s'_i(m)\}$		
6	$s'(m) - \vec{e} = s(m)$		

Table 3.4: Steps of the third proposal

---

<sup>1</sup>the selection of  $2\tau$  as upper bound is resulting from the considerations made in 3.3.3

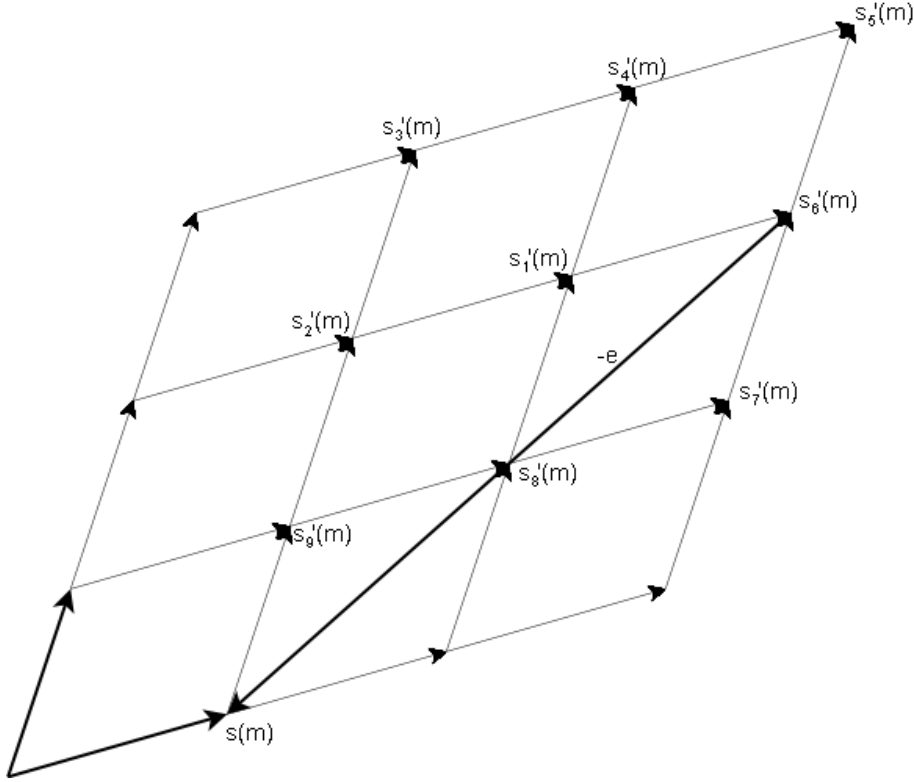


Figure 3.7: Unblinding operation of the third proposal - example in  $\mathbb{R}^2$

### 3.4.2 Alternative unblinding

For step 5 as shown in 3.4.1, there is a possible calculation alternative for the unblinding.

The *blinder* could use the list from step 4

$$\{s'_1(m), \dots, s'_i(m)\}$$

to calculate possible signature candidates by subtracting the blinding vector  $\vec{e}$  from all the items of the received list to build a new list:

$$\{s'_1(m) - \vec{e}, \dots, s'_i(m) - \vec{e}\}$$

After this, the *blinder* can check the validity of the list signature candidates to find the correct signature value.

It is not known, which method is more efficient. There are no thoughts on an efficiency comparison of these methods, because of the performance issues shown in 3.4.3.

### 3.4.3 Issues

The reason why this proposal is not usable, is caused by the effectiveness of the blinding. The number of adjacent lattice points, which the *signer* has to calculate, is simply too high.

A point in one dimension has two direct neighbours. For an additional dimension, the *signer* has to calculate not only the neighbours of the primary point, but also those of his neighbours. For every following additional dimension, the *signer* has to calculate all of the new neighbours of the former calculated points.

So it is easy to see that the number of points grows exponential by the dimension. The number of items in the list of possible signature lattice points in a lattice  $L$  of  $\mathbb{R}^n$  is  $3^n$ .

With already a small number of dimensions - for example 100 - the number of calculated lattice points grows up to  $3^{100} \approx 10^{47}$ , which is almost impossible to handle efficiently.

### 3.4.4 Security

It is very likely that the high number of exchanged lattice points will weaken the security of the lattice signature algorithm itself. As shown in [HPP06], a by far smaller number of message hash/signature pairs could be used to break NTRU.

## 3.5 Fourth proposal - blinding matrix

The fourth proposal was initialized by Prof. Dr. Jintai Ding. His idea was to use a multiplication instead of an addition as process of blinding.

### 3.5.1 Idea

As the first proposal, the *blinder* uses a special value to force the blinding of his message hash. Instead of a blinding summand, the *blinder* uses a blinding matrix as blinding factor.

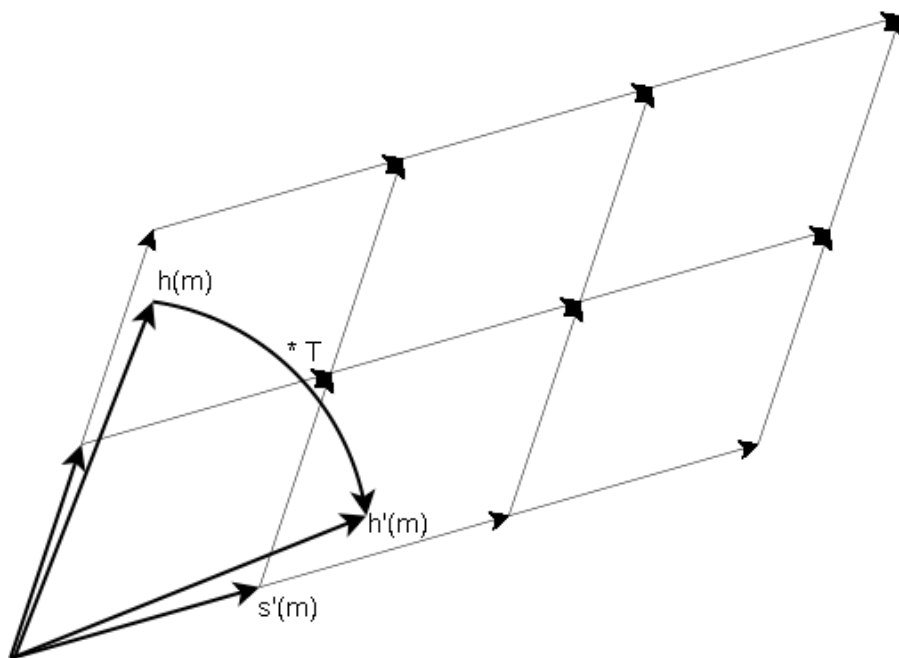


Figure 3.8: Blinding operation of the fourth proposal - example in  $\mathbb{R}^2$

The multiplication with this matrix  $T$  has to fulfil the following conditions:

- It has to map lattice points on lattice points:  $T : L = L$
- It has to preserve the length of a vector.

Step 1: The *blinder* hashes his message  $m$  and receives  $h(m)$ .

Step 2: He multiplies  $h(m)$  by the formerly chosen blinding matrix<sup>2</sup>  $T$  and obtains a new point in  $\mathbb{R}^n$ , which is now called the *blinded* hash  $h'(m)$ . The blinded value  $h'(m)$  will be sent to the *signer*.

<sup>2</sup>further details in 3.5.2

Step 3: The *signer* calculates the correct signature  $s'(m)$  for the received  $h'(m)$  and sends it to the *blinder*.

Step 4: Now the *blinder* is able to calculate the signature  $s(m)$  for the original hash value  $h(m)$  by multiplying the blinded signature  $s'(m)$  of the blinded hash value  $h'(m)$  by the inverse of the blinding matrix  $T$ , called  $T^{-1}$ .

The steps are shown as an overview in table 3.5.

step	Blinder		Signer
1	$h(m)$ out of $m$		
2	$h(m) * T = h'(m)$	$\longrightarrow$	$h'(m)$
3	$s'(m)$	$\longleftarrow$	$s'(m)$ for $h'(m)$
4	$s'(m) * T^{-1} = s(m)$		

Table 3.5: Steps of the fourth proposal

## 3.5.2 The blinding matrix

### Specifications

As seen in 3.5.1 the multiplication with this matrix  $T$  has to fulfil the following specifications:

1. It has to map lattice points on lattice points:  $T : L = L$
2. It has to preserve the length of a vector.

Both specifications are necessary to get valid signatures in the unblinding process.

### A candidate matrix

A promising blinding matrix candidate will be the following product:

$$T = B^{-1}PB$$

where  $P$  is a special permutation matrix,  $B$  is a public basis and  $B^{-1}$  its inverse.

The encapsulation of a permutation matrix in the basis and its inverse will ensure the first specification.

Proof:

A lattice point could be written as  $XB \in L$ , where  $X$  is a random coefficient matrix, so

$$XBT = xPB \in L$$

Some special conditions of the public NTRU-basis could be useful to ensure the second specification. The public basis

$$\begin{pmatrix} 1 & 0 & \dots & 0 & r_0 & r_1 & \dots & r_{N-1} \\ 0 & 1 & \dots & 0 & r_{N-1} & r_0 & \dots & r_{N-2} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 & r_1 & r_2 & \dots & r_0 \\ 0 & 0 & \dots & 0 & q & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 & 0 & q & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0 & 0 & 0 & \dots & q \end{pmatrix}$$

could be divided into quarters of the same size. The rows of the upper right one contain only the coefficients of the public polynomial in different permutations:

$$\begin{pmatrix} r_0 & r_1 & \dots & r_{N-1} \\ r_{N-1} & r_0 & \dots & r_{N-2} \\ \vdots & \vdots & \ddots & \vdots \\ r_1 & r_2 & \dots & r_0 \end{pmatrix}$$

Each vector in a row of the matrix has the same length. If the special permutation matrix  $P$  will only permute the rows, the length of the vectors inside will not change.

### 3.5.3 Issues

It is not clear, if the blinding/unblinding process with matrix  $T$  will deliver a valid signature. The results of the testings are shown in 4.5.

### 3.5.4 Security

#### Improvements to previous proposals

The signer can save all the values, that he received in step 2. As a result, he gets a list of blinded values

$$\{h'_1(m), \dots, h'_n(m)\}$$

where  $n$  is the number of hash values.

The *signer* can also build a list of those original hash values

$$\{h_1(m), \dots, h_n(m)\}$$

Then he can take all the calculated blinded signatures  $s'(m)$  and build a list

$$\{s'_1(m), \dots, s'_n(m)\}$$

And he can construct a list of all the signature values  $s(m)$  according to hash values  $h(m)$  in the form

$$\{s_1(m), \dots, s_n(m)\}$$

Now he can execute the following calculation

$$\|(h_i(m) - s_i(m)) - (h'_j(m) - s'_j(m))\|$$

In the case that the signer has chosen randomly the matching pair of unblinded and blinded values, the result of this calculation will be  $\leq 2\tau$ , because

$$\|h_i(m) - s_i(m)\| \leq \tau \wedge \|h'_j(m) - s'_j(m)\| \leq \tau$$

so it is

$$\|(h_i(m) - s_i(m)) - (h'_j(m) - s'_j(m))\| \leq 2\tau$$

For a not matching pair, the result will also be a length. If the values of this length are set in a different codomain, an attack will be possible.

The fact, that the terms  $\|h_i(m) - s_i(m)\|$  and  $\|h'_j(m) - s'_j(m)\|$  contain both a hash value minus the corresponding valid signature value will ensure, that the value of this terms will always be smaller or equal than the threshold  $\tau$ :

$$\|h_i(m) - s_i(m)\| \leq \tau \wedge \|h'_j(m) - s'_j(m)\| \leq \tau$$

So it is

$$\|(h_i(m) - s_i(m)) - (h'_j(m) - s'_j(m))\| \leq 2\tau$$

There seems to be no possibility to differ a matching pair from a random pair by comparing their error vectors.

### Attack

As said in 3.5.2, the blinding operation should preserve the length of a vector. This fact will allow a simple attack:

The signer can save all the values that he received in step 2. As a result, he gets a list of blinded values

$$\{h'_1(m), \dots, h'_n(m)\}$$

where  $n$  is the number of hash values.

The *signer* can also build a list of original hash values

$$\{h_1(m), \dots, h_n(m)\}$$

and lists of their lengths,

$$\{\|h'_1(m)\|, \dots, \|h'_n(m)\|\}$$

and

$$\{\|h_1(m)\|, \dots, \|h_n(m)\|\}$$

Now he can compare an item from one length-list with all items of the other list to find possible links. If he find two identical values, it is a very high chance that these values are linked. If two compared items are different, it is sure that they are not linked.

## 3.6 Fifth proposal - a blinding matrix and vector

The fifth proposal is only a slight modification of the fourth one, which should avoid the security issues shown in 3.5.4.

### 3.6.1 Idea

As a combination of the first and fourth proposal, the blinder uses both a matrix as a blinding factor and a vector as a blinding summand.

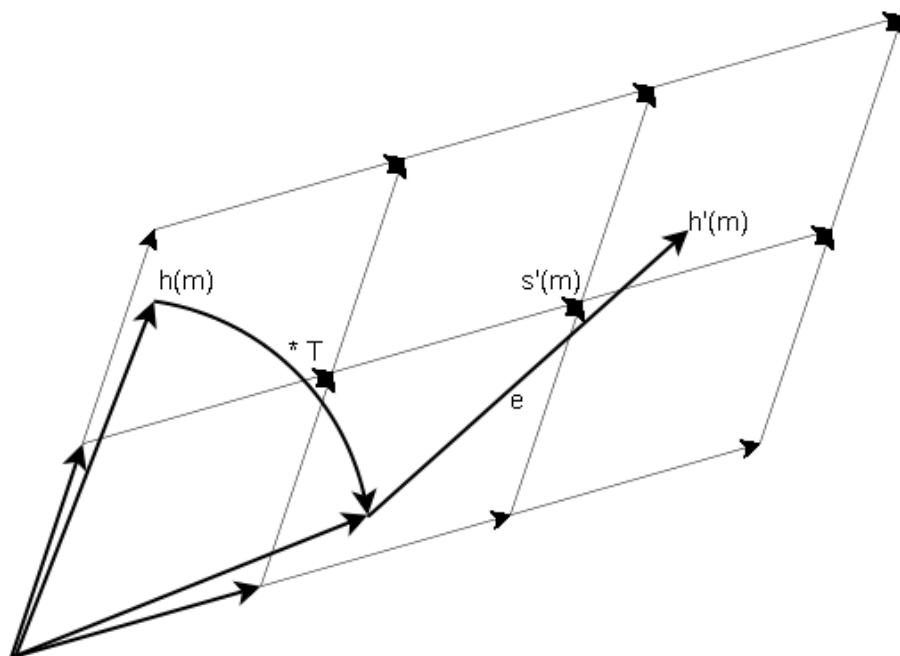


Figure 3.9: Blinding operation of the fifth proposal - example in  $\mathbb{R}^2$

Step 1: The *blinder* hashes his message  $m$  and receives  $h(m)$ .

Step 2: He multiplies  $h(m)$  by the formerly chosen blinding matrix  $T$ .

Step 3: The *blinder* chooses a random linear combination  $\vec{e}$  of the basis vectors of the lattice  $L(B)$ . This linear combination, the *blinding vector* must have integral coefficients and has to be chosen uniformly distributed. He adds the random vector  $\vec{e}$  to  $h(m)$  and obtains a new point in  $\mathbb{R}^n$ , which is now called the *blinded* hash  $h'(m)$ .

Step 4: The *signer* calculates the correct signature  $s'(m)$  for the received  $h'(m)$  and sends it to the *blinder*.

### 3.6. FIFTH PROPOSAL - A BLINDING MATRIX AND VECTOR

---

Step 5: Now the *blinder* can remove the added lattice vector from the blinded signature by calculating  $s'(m) - \vec{e}$ .

Step 6: After this, he is able to calculate the signature  $s(m)$  for the original hash value  $h(m)$  by multiplying the result from step 5 with the inverse of the blinding matrix  $T$ , called  $T^{-1}$ .

The steps are shown as an overview in table 3.6.

step	Blinder		Signer
1	$h(m)$ out of $m$		
2	$h(m) * T$		
3	$h(m) * T + \vec{e} = h'(m)$	$\longrightarrow$	$h'(m)$
4	$s'(m)$	$\longleftarrow$	$s'(m)$ for $h'(m)$
5	$s'(m) - \vec{e}$		
6	$(s'(m) - \vec{e}) * T^{-1} = s(m)$		

Table 3.6: Steps of the fifth proposal

#### 3.6.2 Issues

This proposal will have the same issues as the previous one.

#### 3.6.3 Security

##### Improvements to previous proposals

As shown in 3.5.4 the multiplication with the blinding matrix should be a countermeasure against attacks using the length of the error vector.

The supplemental addition of a lattice vector should be a countermeasure against the attack also shown in 3.5.4. The changing of the message vectors length by a random lattice point summand disturbs the attack which is based on finding a link between blinded and original message by comparing their length.



# Chapter 4

## Implementation and results

This chapter presents the implementation details of a blind signature scheme and the corresponding results of the testing process. Instead of using the fifth proposal, the fourth has been chosen for implementation. This decision should simplify the implementation process and reduces the number of possible errors. A successful implementation could easily be extended to the fifth proposal.

The trunk version (build 1113) of the FlexiProvider [Flexi] contains an implementation of the NTRUSIGN scheme in Java. This NTRUSIGN implementation builds the basis for the blinded version. Java version 1.5.0 is used for compiling.

### 4.1 Implementation details

In this section the implementation details of the blinding and unblinding process are presented.

#### 4.1.1 Blinding

The FlexiProviders NTRUSIGN contains a data type called *Polynomial* which contains the coefficients of a polynomial and the corresponding degrees. The public key polynomial is saved after its generation in a *Polynomial publicH*. This *publicH* is used to call the method *createpublic* (shown in A.1) which creates the upper right quarter of the public matrix that is described in 3.5.2. The resulting matrix is saved as a two-dimensional array of type double and is named *publicmatrix*.

It is also necessary to invert this matrix. An efficient inversion algorithm is published by Tao Pang in [Pang06] and implemented here as the method

*invert* (shown in A.2). The inverted matrix is also stored in a two-dimensional array of type double and is named *invpublicmatrix*.

The third matrix which is required for the blinding is called the random permutation matrix. This matrix is generated by the method *createpermute* (shown in A.3) which needs only the size as an argument. It creates a unit matrix and permutes its rows and columns pseudo-randomly with the use of *Math.random()*.

The multiplication of the blinding is realized with the method *multiply* (shown in A.4) which allows matrix/matrix and matrix/vector multiplications. Analogue to the two-dimensional double array matrix representation, vectors are represented as a standard double array.

The coefficients of the message polynomial *i* have been extracted and stored into the double array *coefficients*.

The complete blinding is done by the first command of the following code

```
c = multiply(publicmatrix,multiply(permutationmatrix,
    multiply(invpublicmatrix, coefficients)));
for (int j=0; j<degree;j++){
    c[j]=((c[j]+q)%q); //mod q
    blind[j]=(int)(c[j]+ 0.5)%q; //rounding to integer
}
i = Polynomial.newFrom(blind); //building the polynomial
```

where *c* and *blind* are temporary arrays. *c* contains the unrounded double values and *blind* contains the rounded integer values.

After the rounding process, the array *blind* is used to build the blinded *Polynomial i*, which is necessary for the signing process.

### 4.1.2 Unblinding

The signature of the blinded message *Polynomial i* is stored in *Polynomial s*.

Inverting the *permutationmatrix* from the blinding process is necessary for the unblinding operation. This could be done by transposing. The result is saved as a two-dimensional array of type double and is named *invpermutationmatrix*.

The unblinding operation is analogue to the blinding operation:

```
c = multiply(publicmatrix,multiply(invpermutationmatrix,
    multiply(invpublicmatrix, coefficients)));
for (int j=0; j<degree;j++){
    c[j]=((c[j]+q)%q); //mod q
```

```
        unblind[j]=(int)(c[j]+ 0.5)%q; //rounding to integer
    }
s = Polynomial.newFrom(unblind); //building the polynomial
```

There is also a method *print* (shown in A.5) for controlling the operations by plotting matrices on the console.

## 4.2 Used parameters

The parameters that are used for the tests are recommended in [NTRU02], except the value of *perturbationBases*, which is changed from 1 to 0 for the experimental runnings:

```
private static final int N = 251;

private static final int q = 128;

private static final int df = 73;

private static final int dg = 71;

private static final int perturbationBases = 0;

private static final int normBound = 310;
```

As shown in 2.2.4,  $N$  means the degree of the used polynomials,  $q$  is the modul,  $df$  and  $dg$  are the number of ones in the binary polynomials  $f$  and  $g$  and the *normBound*<sup>1</sup> is the upper bound for signatures to be valid.

## 4.3 Pretest

Before the main testing, a simple pretest could be done. The substitution of the pseudo-random permutation matrix by a simple rotation matrix will allow a conclusion on the programs correctness. The rotation matrix means a unit matrix, which rows or columns are rotated. Such a matrix will keep the norm of a polynomial exactly the same.

The small errors which are produced by imprecise double entries of the inverted public matrix *invpublicmatrix* are outrounded in the conversion from

---

<sup>1</sup>the *normBound* is called threshold  $\tau$  in the proposals

the vector in the array of type double back to the polynomial with integer coefficients.

The unblinding process with the inverted rotation matrix leads to a valid signature for the original message. This is at least a strong hint, that the program is working correctly.

## 4.4 Efficiency

The program was executed on an Intel PIII-M with 1,13Ghz. The total running time is between 30 and 45 seconds, whereof the major part is used for the generation of the key pair. The blinding/unblinding process itself is very efficient and needs only a small fraction of the whole computation time.

## 4.5 Results

The calculated norm  $\|(s, s * h - m \text{ mod } q)\|$  for the unblinded signature is around 3 (up to 4) times higher than the average norm for a valid signature, which is around 210.<sup>2</sup>

degree	$p$	$m$	$m'$	$s'$	$s''$
$x^8$	87	105	21	-6	121
$x^7$	32	4	38	-10	121
$x^6$	12	114	46	5	127
$x^5$	53	114	75	4	0
$x^4$	100	123	115	-1	3
$x^3$	5	13	95	-19	15
$x^2$	66	22	126	6	117
$x$	73	114	96	8	123
1	59	114	111	3	3

Table 4.1:  $p$  = public polynomial,  $m$  = message polynomial,  $m'$  = blinded message polynomial,  $s'$  = corresponding signature for  $m'$ ,  $s''$  = unblinded  $s'$

This norm value is also higher than the norm bound with a value of 310. So the unblinding process every time leads to an invalid signature. This problem continues, even if the permutation matrix contains the unit matrix with only two rows permuted.

---

<sup>2</sup>the average signature norm for the recommended parameters is also published in [NTRU02]

The table 4.1 shows some typical example results with small parameters. The calculated norm  $\|(s, s * h - m \bmod q)\|$  for the blinded message and its valid signature is around 36. The norm for the original message and the unblinded signature is around 96, which is nearly 3 times higher. The detailed example run and the used parameters are shown in appendix B.

## 4.6 Evaluation

The problem of generating valid signatures in the implementation of the fourth proposal is probably founded in the use of the *centered Euclidean norm*. The considerations on the blinding matrix in 3.5.2 have been made with the *Euclidean norm* as the deciding criterion, so the changes by the permutation matrix are disturbing the *centered Euclidean norm* too much.

The blinding should work on the GGH scheme, which is described in 2.2.2. Unfortunately the GGH scheme is already broken. (see 2.3 for more details)

As shown in 4.3 the blinding with a rotation matrix instead of a pseudo-random permutation matrix is working correctly on *NTRUSign*, but such a blinding is at least insecure in two ways:

1. There are only  $N$  different rotation matrices.
2. An attack comparing the norm values is possible.

Maybe there is another permutation matrix existing which disturbs the norm values in a convenient way.



# Chapter 5

## Conclusion and future prospects

This chapter presents the conclusion of the thesis and the research directions that could be taken from it.

### 5.1 On this work

A lot of aspects have to be considered on the construction of a blind lattice signature scheme. The scheme has to produce valid signatures in the unblinding process and also has to satisfy the security requirements on a blind signature scheme (shown in 1.1). Every proposal seem to be promising at first, but they all had at least one serious issue in security or functionality. There are different attacks on proposals one, two and four (shown in 3.2.2, 3.3.3 and 3.5.4), which allow an attacker to break the property of unlinkability. The third proposal seems to be secure against these attacks, but has a lack in efficiency. The proposals four and five (which are based on a blinding matrix) did not generate valid signatures in the NTRUSIGN test environment. Anyhow, there are a few good ideas which could still be interesting for future work.

The changes on the FlexiProvider [Flexi] for the adaptation of the fourth proposal were not very extensive. The familiarization with the NTRUSIGN part of the FlexiProvider was more difficult than the implementation itself. Probably the modified FlexiProvider could be easily adjusted for the implementation of future proposals.

### 5.2 Future prospects

Although the original GGH scheme and NTRUSIGN without perturbations are already broken (details in 2.3), the idea of a digital blind signature based

on hard lattice problems is still interesting.

The fifth proposal, introduced in 3.6.1, seems to be a promising candidate. The residual problem is to find a special blinding matrix that changes the norm value in a sufficient small way and guarantees an adequate randomness. A blinding matrix which satisfies these requirements could easily substitute the matrix candidate from 3.5.2 in the implementation of the fourth proposal (shown in 4).

There are also other proposals thinkable which base on other blinding techniques than a blinding vector summand or a matrix blinding factor. The need for alternative blind signature schemes will become increasing importance if the quantum computing research gains progress. This work can be seen as basis for future attempts on constructing a blind lattice signature scheme.

# Appendix A

## Implemented methods

### A.1 `createpublic`

The *createpublic* method creates the polynomial quarter of the NTRU matrix.

```
public static double[][] createpublic (Polynomial publicH){
    int degree = publicH.degree()+1;
    double publicmatrix[][] = new double [degree] [degree];
    int[]publicarray = new int[degree];
    for (int j=0; j<degree;j++){
        publicarray[j]= publicH.getCoefficient(j);
    }
    for (int j=0; j<degree;j++){
        for (int h=0; h<degree;h++){

            publicmatrix[j][h]= publicarray[(j-h+degree)%degree];

        }
    }
    return publicmatrix;
}
```

### A.2 `invert`

This *invert* method is used for matrix inversions and was published in [Pang06].

```
public static double[][] invert(double a[][]) {
    int n = a.length;
    double x[][] = new double[n][n];
```

## APPENDIX A. IMPLEMENTED METHODS

---

```
        double b[][] = new double[n][n];
        int index[] = new int[n];
        for (int i=0; i<n; ++i) b[i][i] = 1;

// Transform the matrix into an upper triangle
    gaussian(a, index);

// Update the matrix b[i][j] with the ratios stored
    for (int i=0; i<n-1; ++i)
        for (int j=i+1; j<n; ++j)
            for (int k=0; k<n; ++k)
                b[index[j]][k]
                    -= a[index[j]][i]*b[index[i]][k];

// Perform backward substitutions
    for (int i=0; i<n; ++i) {
        x[n-1][i] = b[index[n-1]][i]/a[index[n-1]][n-1];
        for (int j=n-2; j>=0; --j) {
            x[j][i] = b[index[j]][i];
            for (int k=j+1; k<n; ++k) {
                x[j][i] -= a[index[j]][k]*x[k][i];
            }
            x[j][i] /= a[index[j]][j];
        }
    }
    return x;
}

// Method to carry out the partial-pivoting Gaussian
// elimination. Here index[] stores pivoting order.

public static void gaussian(double a[][],
    int index[]) {
    int n = index.length;
    double c[] = new double[n];

// Initialize the index
    for (int i=0; i<n; ++i) index[i] = i;

// Find the rescaling factors, one from each row
    for (int i=0; i<n; ++i) {
```

```
    double c1 = 0;
    for (int j=0; j<n; ++j) {
        double c0 = Math.abs(a[i][j]);
        if (c0 > c1) c1 = c0;
    }
    c[i] = c1;
}

// Search the pivoting element from each column
int k = 0;
for (int j=0; j<n-1; ++j) {
    double pi1 = 0;
    for (int i=j; i<n; ++i) {
        double pi0 = Math.abs(a[index[i]][j]);
        pi0 /= c[index[i]];
        if (pi0 > pi1) {
            pi1 = pi0;
            k = i;
        }
    }
}

// Interchange rows according to the pivoting order
int itmp = index[j];
index[j] = index[k];
index[k] = itmp;
for (int i=j+1; i<n; ++i) {
    double pj = a[index[i]][j]/a[index[j]][j];

    // Record pivoting ratios below the diagonal
    a[index[i]][j] = pj;

    // Modify other elements accordingly
    for (int l=j+1; l<n; ++l)
        a[index[i]][l] -= pj*a[index[j]][l];
}
}
}
```

### A.3 createpermute

The *createpermute* method creates a pseudo-random permutation of the unit matrix.

```
public static double[][] createpermute(int degree){
int[][] pm = new int[degree][degree];
double[][] permutationmatrix = new double[degree][degree];
for(int i = 0; i < degree; i++)
    for(int j=0; j < degree;j++)
        if(i==j) pm[i][j] = 1;
int swapper, swap1, swap2;
for( int i = 0; i < degree; i++ ) {
    if( Math.random() < 0.5 )
        swapper = 0;
    else
        swapper = 1;

    swap1 = (int)(Math.random()*degree);
    swap2 = (int)(Math.random()*degree);

    if(swapper==0){ //swap rows
int tmp[] = new int[degree];

    for(int j = 0; j < degree; j++)
        tmp[j] = pm[swap1][j];
    for(int j = 0; j < degree; j++)
        pm[swap1][j] = pm[swap2][j];
    for(int j = 0; j < degree; j++)
        pm[swap2][j] = tmp[j];
    }
else{ //swap columns
int tmp[] = new int[degree];
for(int j = 0; j < degree; j++)
    tmp[j] = pm[j][swap1];
for(int j = 0; j < degree; j++)
    pm[j][swap1] = pm[j][swap2];
for(int j = 0; j < degree; j++)
    pm[j][swap2] = tmp[j];
    }
}
```

```
    for(int a = 0; a < degree; a++)
    for(int b=0; b < degree;b++)
    permutationmatrix[a][b]=pm[a][b];
    return permutationmatrix;
}
```

## A.4 multiply

These *multiply* methods are for matrix/matrix and matrix/vector multiplications.

```
public static double[][] multiply(double a[][],double b[][]) {
    if (a.length!=b.length) System.out.println("multiplication size error");
    int size = a.length;
    double c[][] = new double[size][size];

    for (int h = 0; h < size; h++) {
    for (int j = 0; j < size; j++) {
    c[h][j] = 0.0;
    for (int k = 0; k < size; k++) {
    c[h][j] += a[h][k] * b[k][j];
    }
    }
    }
    return c;
}
```

```
public static double[] multiply(double a[][],double b[]) {
    if (a.length!=b.length) System.out.println("multiplication size error");
    int size = a.length;
    double c[] = new double[size];

    for (int h = 0; h < size; h++) {
    c[h] = 0.0;
    for (int k = 0; k < size; k++) {
    c[h] += a[h][k] * b[k];
    }
    }
    return c;
}
```

## A.5 print

The *print* method allows matrix printing on the console.

```
public static void print(double[][] matrix){
    for (int zeile = 0; zeile < matrix.length; zeile++){
        System.out.print("Zeile " + zeile + ": ");
        for ( int spalte=0; spalte < matrix[zeile].length; spalte++ )
            System.out.print( matrix[zeile][spalte] + " ");
        System.out.println();
    }
}
```

# Appendix B

## Example results

This example run is executed with the following parameters:

```
private static final int N = 9;  
  
private static final int q = 128;  
  
private static final int df = 5;  
  
private static final int dg = 3;  
  
private static final int perturbationBases = 0;
```

The random public polynomial

$$87x^8 + 32x^7 + 12x^6 + 53x^5 + 100x^4 + 5x^3 + 66x^2 + 73x + 59$$

creates the public matrix:

$$\begin{pmatrix} 59 & 87 & 32 & 12 & 53 & 100 & 5 & 66 & 73 \\ 73 & 59 & 87 & 32 & 12 & 53 & 100 & 5 & 66 \\ 66 & 73 & 59 & 87 & 32 & 12 & 53 & 100 & 5 \\ 5 & 66 & 73 & 59 & 87 & 32 & 12 & 53 & 100 \\ 100 & 5 & 66 & 73 & 59 & 87 & 32 & 12 & 53 \\ 53 & 100 & 5 & 66 & 73 & 59 & 87 & 32 & 12 \\ 12 & 53 & 100 & 5 & 66 & 73 & 59 & 87 & 32 \\ 32 & 12 & 53 & 100 & 5 & 66 & 73 & 59 & 87 \\ 87 & 32 & 12 & 53 & 100 & 5 & 66 & 73 & 59 \end{pmatrix}$$

The inverse of the public matrix contains very small rational entries around  $3 * 10^{-3}$ , so the printing of this matrix would be too large in its measure.

*APPENDIX B. EXAMPLE RESULTS*

---

The message polynomial:

$$105x^8 + 4x^7 + 114x^6 + 114x^5 + 123x^4 + 13x^3 + 22x^2 + 114x + 114$$

The pseudo-random permutation matrix

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

and its inverse:

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

The blinded message polynomial (rounded)

$$21x^8 + 38x^7 + 46x^6 + 75x^5 + 115x^4 + 95x^3 + 126x^2 + 96x + 111$$

and the corresponding signature polynomial:

$$-6x^8 - 10x^7 + 5x^6 + 4x^5 - x^4 - 19x^3 + 6x^2 + 8x + 3$$

The norm value calculated for the blinded message and its signature is 36, 21.

The unblinded signature polynomial (rounded):

$$121x^8 + 121x^7 + 127x^6 + 3x^4 - 15x^3 + 117x^2 + 123x + 3$$

The norm value calculated for the original message and the unblinded signature is 96, 02.

# Bibliography

- [Cha83] D. Chaum: *Blind signatures for untraceable payments* Advances in Cryptology - Crypto '82, Springer-Verlag (1983), 199-203.
- [CPS94] J. L. Carmenisch, J.-M. Piveteau and M.A. Stadler *Blind signature based on the discrete logarithm problem* EUROCRYPT '94, Perugia, Italy
- [Flexi] <http://www.flexiprovider.de>
- [GGH96] Oded Goldreich, Shafi Goldwasser and Shai Halevi: *Public-Key Cryptosystems from Lattice Reduction Problems*. Electronic Colloquium on Computational Complexity - Reports Series 1996
- [JL96] Wen-Shenq Juang and Chin-Laung Lei *Blind Threshold Signatures Based on Discrete Logarithm* National Taiwan University, Taipei, Taiwan, R.O.C., 1996
- [JLO97] Ari Juels, Michael Luby and Rafail Ostrovsky *Security of Blind Digital Signatures* RSA Laboratories, Digital Fountain, UCLA, 1997
- [KC99] Moonsang Kwon and Yookun Cho, *Randomization Enhanced Blind Signature Schemes Based on RSA* IEICE Trans. Fundamentals, Vol.E82, No.1 January 1999
- [HMP94] P. Horster, M. Michels and H. Petersen: *Meta-ElGamal signature schemes* Proc. of the 2nd ACM conf. on Comp. and Comm. security, Fairfax, Virginia, 1994
- [HPP06] Phong Q. Nguyen and Oded Regev: *Learning a Parallelepiped: Cryptanalysis of GGH and NTRU Signatures* EUROCRYPT '06, Best Paper Award

## BIBLIOGRAPHY

---

- [Mic01] Daniele Micciancio: *Improving Lattice Based Cryptosystems Using the Hermite Normal Form* Cryptography and Lattices Conference - CaLC 2001. March 29-30, 2001, Providence, Rhode Island. Lecture Notes in Computer Science 2146. Springer-Verlag, pp. 126-145
- [Ngu06] Phong Q. Nguyen *A Note on the Security of NTRUSIGN* École normale supérieure and CNRS, France, 2006
- [NTRU02] Jeffrey Hoffstein, Nick Howgrave-Graham, Jill Pipher, Joseph H. Silverman, William Whyte: *NTRUSign: Digital Signatures Using the NTRU Lattice - Preliminary Draft 2* April 2002
- [NTRU03] Jeffrey Hoffstein, Nick Howgrave-Graham, Jill Pipher, Joseph H. Silverman, William Whyte: *NTRUSign: Digital Signatures Using the NTRU Lattice* CT-RSA 2003 Proceedings
- [Ohk99] Iyako Ohkubo, Fumiaki Miura, Masayuki Abe, Atsushi Fujioka and Tatsuaki Okamoto: *An Improvement on a Practical Secret Voting Scheme*. ISW, Springer, pp. 225-234, 1999
- [Poi96] David Pointcheval and Jaques Stern *Provably Secure Blind Signature Schemes* Advances in Cryptology - Proceedings of ASIACRYPT '96, M. Y. Rhee and K. Kim Eds. Springer-Verlag, LNCS 1163, pages 252-265, 1996
- [Sch90] C.P. Schnorr: *Efficient Identification and Signatures for Smart Cards* Crypto '89, LNCS 435, pages 235-251, Springer Verlag, 1990
- [Pang06] Tao Pang: *An Introduction to Computational Physics, 2nd Edition* Cambridge University Press, 2006