Proposal of a new efficient public key system for encryption and digital signatures

Gerold Grünauer
Email: geroldgruenauer@web.de

10/01/2007

Abstract

In this paper a new efficient public key cryptosystem usable for both encryption and digital signatures is presented. Due to its simple structure this public key cipher can be implemented easily in every software or hardware device, making the cryptosystem available for circumstances where the implementation of an alternative like RSA, El Gamal / Diffie - Hellmann, etc. is too complicated. Furthermore the construction on the closest and shortest vector problem using a new homomorph "almost" linear one-way function gives not only strong evidence of the ciphers security, but may be also the base for a new class of cryptographic primitives based on lattice problems. Therefore this cipher and its construction is a good alternative to cryptosystems based on the integer factoriziation problem or the discrete logarithm.

1 Introduction

In the last decades a lot of public key cryptosystems have been proposed, but nevertheless only a few cryptosystems (namely RSA, El Gamal / Diffie-Hellman, DSA, ...) are used in practise. However, these cryptosystems are very hard to implement (especially for unskilled new programmers or in smart cards or comparable environments). Furthermore all those public key systems are based on number theoretical problems like the integer factorization problem or the discrete logarithm. Should any number theoretical problem be solved the security of these ciphers is in danger.

In order to cicumvent these drawbacks a lot of people have been searching for faster and sometimes easier implementable alternative cryptographic algorithms. Worth to mention here are for example the cryptosystems based on the knapsack problem (see [1] for the history of knapsack type cryptosystems). These public key encryption algorithms would have been faster than any system involving number theoretical constructions. However, all known ciphers of that type have either been broken and / or are completely impractical because of too large keys / ciphertext.

Other proposed public key systems are based on problems which are hardly known and researched by the cryptographic community. The FAPKC (Finite Automata Public Key Cryptosystem, see [2]) is an example for such a cipher. This promising system (as stated by [4] in Ch. 19.10) has been completely broken after 10 years. The multivariate polynomial equation problem was also used as a base for new public key algorithms but is now successively researched and can be solved quite well by now (see [3] for example).

However, despite of all those failures above, a new class of promising public key algorithms arises from the well researched lattice problems. In 1996 M. Ajtai and C. Dwork presented a new probabilistic public key encryption algorithm([5]) for which they could proof that breaking a random instance of the cryptosystem is as hard as solving the worst-case of the ciphers base problem, the unique shortest vector problem. A lot of other systems followed like the GGH system (see [6]), or a construction of O. Regev (see [7]). The NTRU system (see [8] for a complete reference of papers concerning NTRU) finally was the breakthrough for lattice based cryptosystems. This cryptosystem is not only very fast but has (the first time for a lattice based encryption algorithm) also an acceptable keysize (growing only linear with the security parameter t). Nevertheless, not only the NTRU system but all lattice based cryptographic constructions lack of a secure digital signature algorithm.

In this paper, this gap is closed, by presenting a new secure complete efficient cryptosystem featuring both digital signature generation and public key encryption. The new cryptosystem has for the security parameter n a keysize of $\mathcal{O}(n)$ and for encryption, decryption, signature generation and verification a running time of $\mathcal{O}(n^2)$. Furthermore a new cryptographical "almost" linear homomorphic one-way function is presented, which can be used to build other primitives like hash functions, etc.

2 Background

2.1 Lattices and cyclic lattices

Let $B = \{b_1, ..., b_n\}$ be a set of n linearly independent vectors. This set B, called basis, generates the lattice $L(B) = \{\sum_{i=0}^{n} x_i \vec{b_i} \mid x_i \in Z\}$, therefore all linear combinations of the vectors in B. The basis of a lattice is not unique. The p norm is defined by the formula: $\|x\|_p = \sqrt[p]{\sum_{i=0}^{n} x_i{}^p}$.

A cyclic basis consists of cyclically shifted versions of a base vector b (with components $b_1, ..., b_n$), for example:

$$B = \left\{ \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}, \begin{pmatrix} b_n \\ b_1 \\ \vdots \\ b_{n-1} \end{pmatrix}, \begin{pmatrix} b_{n-1} \\ b_n \\ \vdots \\ b_{n-2} \end{pmatrix}, ..., \begin{pmatrix} b_2 \\ b_3 \\ \vdots \\ b_1 \end{pmatrix} \right\}$$

Such a cyclic basis can be constructed by polynomial multiplication:

$$B = \{b(x)x^i \, mod \, x^n - 1 \mid 0 \leq i < n\}$$

The lattice is now generated by ordinary polynomial multiplication in $GF(p^n)$, where n denotes the dimension of the lattice:

$$L = \{b(x)v(x) \, mod \, x^n - 1 \mid v(x) \in GF(p^n)\}$$

Every vector $b(x)$ generates a cyclic basis (if the polynomyial $x^n - 1$ is irreducible in $GF(p^n)$). Hence polynomial multiplication of $b(x)$ with another arbitrary polynomial / vector can be seen as a basis transformation or as the selection of a specific lattice vector.

2.2 Lattice problems

From lattices as defined above, a lot of NP-hard problems arise. The security of the new cryptosystem is based on the actual hardness of these problems:

- Shortest vector problem (SVP): given a basis B of the lattice $L(B)$, find $\vec{v} \in L(B)$ such that $\|\vec{v}\|_p$ is minimal.
- Closest vector problem (CVP): given a basis B of the lattice $L(B)$ and a lattice vector $\vec{w}$, find another lattice vector $\vec{v}$ such that $\|\vec{w} - \vec{v}\|_p$ is minimized or $\vec{v}$ is closest to $\vec{w}$.

It has been shown that it is computationally not harder to solve the shortest vector problem than the closest vector problem ([9]). Furthermore it is known that approximating the closest vector problem concerning the L_p-Norm to a better factor than $\sqrt[p]{log(n)}$ is NP-hard (see [10], [11]).

However, NP-hardness is no proof for a problems actual hardness. For example most knapsack-type cryptosystems can be generally effective attacked by lattice reduction techniques, although the subset sum problem is NP-hard. Even hard high density subset sum instances (like the one produced by the Chor-Rivest cryptosystem) can be directly solved. On the other hand, problems for which NP-hardness could not have been proven up to date, cannot be solved in practise (like the integer factorization problem for example). Therefore the "hardness" of the problems used by the new cryptosystems are measured in this paper by the average running time of the best known solving algorithm and by the average reduction factor (i.e. how "close" is the approximated vector to the actual solution).

3 The new cryptosystem

First the new "almost" linear homomorphic one-way function is described, afterwards the new signature scheme and the public key encryption is defined. An example with artificial small parameters can be found at the very end of this section.

3.1 "Almost" linear homomorphic one-way function

A homomorphism is a function that preserves the original group structure:

$$f(a+b) = f(a) + f(b)$$

Exponentiation for example is a homomorphism:

$$x^{a+b} = x^a x^b$$

Moreover exponentiation in suitable discrete subgroups is also an one-way function and therefore an appropriate base for cryptsystems (like El Gamal, Diffie-Hellman, ...).

Linear functions (without constant term) are always homomorphisms in any dimension, but not one-way functions. In this paper I disclose a new type of "almost" linear homomorphism, i.e. a linear function which introduces a small error for every addition or subtraction (mutliplication can be viewed as a series of additions and therefore a much bigger error is introduced). This new function combines all positive properties and constructions shown in the cryptosystem described by Ajtai and Dwork ([5], A public key cryptosystem with worst-case / average-case equivalence) and in an algorithm for "Public key distribution using approximately linear functions" by R.C. Merkle ([12]).
Let $f(x,g) = Round((x * g\, Mod\, p) * \frac{q}{p})\, Mod\, q$ be a similiar approximately linear function as defined in [12]. Then the following facts are valid:

- $f(a+b,g) = (f(a,g) + f(b,g) \pm [0;0.5])\ Mod\ q$
- $f(a*b,g) = (f(a,g) * b \pm [0;0.5] * b)\ Mod\ q$

$[0;0.5]$ means, that you have to add a random error from the intervall $[0;0.5]$. Given a hard, random instance of the subset sum problem $a = \{a_1, a_2, ..., a_n\}$ the private key of a key exchange system would be a random number $g\ Mod\ p$. The public key is then defined as $b = \{b_1 = f(a_1,g), ..., b_n = f(a_n,g)\}$. A shared approximate key is then computed by Bob as

$$s = \vec{x} \cdot a\, Mod\, p$$

and

$$Key \approx \vec{x} \cdot b\, Mod\, q$$

, where $\vec{x}$ denotes a binary vector. Alice can then compute her shared key from s and her private key g:

$$Key \approx f(s,g) \approx f(\vec{x} \cdot a, g) \approx \vec{x} \cdot \{f(a_1,g), ..., f(a_n,g)\}$$

If an attacker now tries to find a non binary vector $\vec{c}$ such that

$$s = \vec{c} \cdot a\, Mod\, p$$

he can also compute

$$Key \approx \vec{c} \cdot b\, Mod\, q$$

but gets a much bigger error (probable an error much bigger than q) and therefore no information about Key. [1] Therefore the introduced error makes the "almost" linear function to be one-way. The error introduced by this function is proportional to $\|\vec{x}\|_1$.

It has been shown by R. Merkle ([12]) that, given p, q and some $y_i = f(x_i, g)$, it is NP-complete to find g. Glenn Durfee, however, concludes - although he didn't break the specified scheme - in an addendum of [12] that "...the scheme [and hence the above defined homomorphic one-way function] is insecure for practical key sizes, and uncompetitive with existing methods for (even only potentially) secure key sizes."
As a result, a better homomorphic one-way function for practical cryptography has to be found.
In [5] Ajtai and Dwork defined a similiar function for probabilistic encryption:

Let $\vec{x_i}$, $\vec{y}$ and $\vec{g}$ be n-dimensional real vectors, such that:

$$f(\vec{x_i}, \vec{g}) = (\vec{x_i} \cdot \vec{g})\, Mod\, 1 \approx 0$$

$$f(\vec{y}, \vec{g}) = (\vec{y} \cdot \vec{g})\, Mod\, 1 \approx 0.5$$

A random point $\vec{p}$ of the lattice spanned by $\{\vec{x_1}, ..., \vec{x_n}\}$ is used to encrypt a 0 bit, a point from the lattice $\{\vec{x_1}, ..., \vec{x_n}\} + \vec{y}$ (i.e. $\vec{y}$ is alway added to the choosen lattice point) is used to encrypt a 1 bit. These random points are chosen by multiplication of the base with a small vector (hence, the error introduced by the function f is small). Alice can, with her private key $\vec{g}$, reveal the encrypted bit by computing $s = f(\vec{p}, \vec{g}) \approx \{0; 0.5\}$

These two attempts of homomorphic functions can be combined to a more general construction:

$$f(\vec{x}, \vec{g}) = (\vec{x} \cdot \vec{g}\ Mod\ p) * \frac{q}{p}\ Mod\ q \approx r * \frac{q}{p}\ Mod\ q$$

for some integer r. Given a basis B for a random lattice of $GF(p^n)$, a new key exchange system can be now established[2]: Choose a random vector $\vec{g}$ of $GF(p^n)$ and compute the new vector $\vec{y} = f(B, \vec{g})$, i.e. $y_i = f(\vec{b_i}, \vec{g})$ that is, the i-th component of $\vec{y}$ is the new one-way function of the i-th basis vector $\vec{b_i}$ of B. Now Bob chooses a random lattice vector $\vec{x}$ and computes the associated shared key k. Alice can do the same by computing $k = f(\vec{x}, \vec{g})$.
This function, nevertheless, is not more efficient than the above described cryptosystems. By combining this one-way functions with cyclic lattices, however,

[1] It is pretty easy to find a relative big $\vec{c}$. It is , however, very hard to find the original binary vector, because you would have to solve the subset sum problem.

[2] As the observing reader might have seen by now, this version of the encryption algorithm is complete yet and can be hence broken easily and is only listed here to demonstrate the principles, see 3.2 for additional infos and improvements

it is possible to drastically enhance the effiency of the new function:

x and g are now no longer seen as vectors but as polynomials of $GF(p^n)$. The new (and final) "almost" linear homomorphic one-way function is given by:

$$y = f(x, g) = Truncate(x * g \; Mod \; x^n - 1)$$

where $y = Truncate(x)$ ($x \in GF(p^n)$, $y \in GF(q^n)$) truncates every coefficient x_i of x by the following formula: $y_i = (x_i * \frac{q}{p}) \; Mod \; q$.
This function can be seen as a complete basis transformation and therefore a public key for the above described key-system can be generated with only one polynomial multiplication and truncation. Furthermore by applying the new function $f(x, g)$ on a randomly selected lattice point/polynomial, n - shared keys are computed in paralell.

Using this new function, both a signature scheme and an encryption scheme are constructed in the next section. The security of this new function will be analysed in section 4.

3.2 The new signature scheme

The new signature scheme uses as it's base the same zero knowledge proof used by discrete logarithm signature schemes (as described in [4], Chapter: 20.4):

Let α be a generator of a discrete subgroup of order q and $y = \alpha^x$ (with x the private key). Then a new signature is computed by:
$s_1 = \alpha^r$ (r is a random integer), $s_2 = Hash(s_1 \| m) * x + r \; Mod \; q$.
The signature is verificated by checking: $s_1 y^{Hash(s_1 \| m)} = \alpha^{s_2}$.

As the observing reader may have noticed in the above section 3.1, it is hard to compute exactly the same g, such that $y = f(x, g)$, but it is easy to "upsample" the truncated polynomial y (filling the deleted and missing information with random data) and compute: $g' = y' * x^{-1} \; Mod \; x^n - 1$. It is clear that $y = f(x, g')$.
Therefore the public key for the new scheme is computed by applying the one-way function $f(x, g)$ to two different polynomials x_1 and x_2. Now the above strategy to find an alternative private key no longer works, because if you choose g' such that $y_1 = f(x_1, g')$, the second equation won't be satisfied: $y_2 \neq f(x_2, g')$[3] . A polynomial g' that satisfies both equation can be only found by solving the lattice problem related with the one-way function f (more about that topic can be find in the section 4).
The new signature scheme can be now defined:

[3]You would have to choose as filling data exactly this data, which has been deleted by the truncation process.

Key Generation [4]

1. Choose two random and public polynomials x_1, x_2 from $GF(p^n)$.
2. Choose a random secret polynomial g from $GF(p^n)$.
3. Compute the public key: $P = \{y_1 = f(x_1, g);\ y_2 = f(x_2, g)\}$

The public key consists of: $x_1, x_2, P = \{y_1, y_2\}$
The private key consists of: x_1, x_2, g

Signature generation

1. Choose a random secret polynomial r from $GF(p^n)$.
2. Compute: $R = \{r_1 = f(x_1, r);\ r_2 = f(x_2, r)\}$
3. Choose a polynomial e_1 with $\|e_1\|_1 = t$ and e_2 such that $\|e_2\|_1 = 2t$ [5]
   These polynomials must be dependent on $Hash(R\|m)$.
4. Compute: $s = (e_1 * g + e_2 * r)\ Mod\ x^n - 1$
5. Check the new signature with the algorithm below.
   Go to step 1, if failure.

The signature for m consists of: $R,\ s$

Signature verification

1. Recompute e_1 and e_2 with the step 3 of signature generation.
2. Compute: $U = \{u_1 = f(x_1, s);\ u_2 = f(x_2, s)\}$
3. Compute: $V = \{v_i = ((e_1 * y_i + e_2 * r_i)\ Mod\ x^n - 1) \in GF(q^n) \| i \in \{1;\ 2\}\}$
4. Compute: $\Delta = U - V = \{\Delta_i = (u_i - v_i) \in GF(q^n) \| i \in \{1;\ 2\}\}$
5. Represent the coefficients of Δ as integers (e.g. 250 Mod 256 = -6).
6. Check the error coefficients of the error polynomial Δ:
   If $|\Delta_{i,j}| > \sigma_1$, reject the signature $(0 \le j < n)$.
7. Check the complete error polynomial:
   Check the distribution of the error coefficients with respect to the gaussian normal distribution (See remarks below).
8. Accept the signature.

Remarks

The quantity q should be approximately the squareroot of p ($q \approx \sqrt{p}$). p should be a primenumber and $\gcd(p, q)$ shall be 1 (e.g. p = 65521, q = 256). The lattice dimension n must be choosen, such that $x^n - 1$ has no low weight factors (i.e. $\|Factor\|_1 \ge p$), hence n must be a prime too (e.g. n = 1021, n = 2039). The verification constants σ_1 and the weight factor t must be choosen with care, because they determine the signature schemes security against forging attacks with approximated secret keys g'[6] and against a direct lattice approximation attack against s[7]. These constants are determined by taking the variance of a

[4]The public key generation procedure given here is also be used for the encryption scheme.

[5]The weight of the second polynomial must be greater, because now the signature error is dominated by a randomly choosen error

[6]This attack tries to recover or approximate the secret key

[7]this attack tries to generate a valid signature without any knowledge about the secret key, read more in section 4

single coefficient error value $Var_{SingleError} = \sum_{i=0}^{q-1} (i\frac{q}{p} - 0.5)^2 * \frac{1}{q} \approx \frac{1}{12}$:

$$\sigma_1 = \sqrt{(\|e_1\|_1 + \|e_2\|_1) * Var_{SingleError}} * c = \sqrt{\frac{3t}{12}} * c$$

The larger t gets, the more information you can host in the zero knowledge proofs decision polynomials e_1 and e_2 (only the information in e_1 is important for birthday attacks), but on the other hand the easier it becomes to approximate a single signature s [8]. An appropriate choice for t may be $20 \leq t \leq 30$. The constant c should be 4.

The decision polynomials e_1 and e_2 shall consists of -1, 0 or 1 for each coefficient.

In step 7, the error polynomials shall be checked against the gaussian normal distribution[9]: Count the values greater σ, 1.5σ, 2σ, ... and compare these values with the expected values. The tolerance for these values shall be choosen such that for example every second computed signature is rejected (these values can be choosen for a particular implementation by generating 1000 random signatures and observing the specified values).

The suggested parameters are:

n = 1021, p = 65521, q = 256, t = 16, c = 3.5

n = 2039, p = 65521, q = 256, t = 16, c = 3.5

No known attacks exists today that can practically attack any of these two specified parameter sizes. The key size is 2000 Byte, the signature size 4000 Byte or 4000 Byte (for the second parameter set) and 8000 Byte.

The security of the scheme against attacks, where the attacker has known signatures at his hand, are discussed in section 4.

3.3 The new encryption scheme

The new encryption scheme is based on the key-exchanging algorithm of El Gamal /Diffie Hellman. In a nutshell we choose a random lattice point out of the two cyclic lattices x_1 and x_2 and compute the corresponding approximated key point. The message - transformed into a polynomial of $GF(q^n)$ by suitable error correction methods - is then added to the approximated key point. The legitimate recipient can compute, using his private polynomial g, the another approximated key and subtract it from the message. The errors introduced here can be decoded using the error correction scheme. The error tolerance must be choosen such that an approximated private key cannot decrypt a message in practise.

Key Generation [10]

[8] You would first compute the signature until step 4, then you would solve the lattice problem to find a suitable s, see more about that in 4

[9] See the appendix for a concrete method

[10] The public key generation procedure given here is also be used for the signature scheme.

1. Choose two random and public polynomials x_1, x_2 from $GF(p^n)$.
2. Choose a random secret polynomial g from $GF(p^n)$.
3. Compute the public key: $P = \{y_1 = f(x_1, g);\ y_2 = f(x_2, g)\}$

The public key consists of: $x_1, x_2, P = \{y_1, y_2\}$
The private key consists of: x_1, x_2, g

Encryption

1. Choose two random polynomials e_1 and e_2 with $\|e_i\|_1 \leq t$ and each coefficient either -1, 0 or 1.
2. Compute: $c_1 = (e_1 * x_1 + e_2 * x_2)\ Mod\ x^n - 1 \in GF(p^n)$
   $c_2{}' = (e_1 * y_1 + e_2 * y_2)\ Mod\ x^n - 1 \in GF(q^n)$
3. Represent the message M using suitable error correction methods as a polynomial m in $GF(q^n)$ (see Remarks).
4. Compute: $c_2 = c_2{}' + m \in GF(q^n)$

The encrypted message consists of: c_1 and c_2

Decryption

1. Compute: $c_2{}' = f(c_1, g)$
2. Compute: $m = c_2 - c_2{}' \in GF(q^n)$
3. Correct all errors and recompute the Message M.

Remarks

As with the signature scheme, the parameter t must be choosen carefully. In the encryption scheme the normal deviation is computed as $\sigma = \sqrt{\frac{2t}{12}}$ because the two "decision polynomials" e_1 and e_2 are weighted equally. A possible parameter choice might be:

n = 1021, p = 65521, q = 256, t = 681
n = 2039, p = 65521, q = 256, t = 1359
i.e. you choose e_1 and e_2 completely at random with each coefficient be either -1, 0 or 1.

As noted above, a suitable error correction scheme must be used to correct errors introduced by the "almost" linear function f. First of all the message M (with error correction data) is transformed into a polynomial by taking the data (for n = 1021, p = 65521, q = 256, t = 681) as a polynomial of $GF(3^n)$ and transforming each coefficient to $GF(256)$ by computing: $m_i' = m_i * \frac{256}{3}\ Mod\ 256$. This transformation can be reversed by using the same truncation function as used by the one-way function f. Now it must be observed, that the truncation operation will revert this, as long as the absolute error introduced is not bigger than $\frac{256}{3}/4 = 2\sigma$. Hence every coefficient is for the above first parameter choice transformed correctly with a probability of approximately 95.45% (because, it is transformed correctly if the error is smaller than two times the standard deviation of this error). A bit probability math shows you that one error would occur with $P(1) = 36.2\%$, two errors with $P(2) = 12.9\%$, ..., $P(8) = 3.33 * 10^{-1}$ (i.e. all 3 million blocks the errors cannot be detected if the error detection algorithm is constructed such that up to 8 errors can be detected). Moreover an error adds or subtract always 1 to the coefficient (this fact can be used to construct the checksum - see the appendix for such a method). Considering todays lattice reduction and other cryptanalytical techniques, the best choice is to use an error correction scheme, that can detect or correct up to 3 errors.

As a result the the remaining errors must be corrected by an ordinary error correction scheme (Reed - Muller codes cannot be used here, because the word size is only 3 - in this case). A possibility would be, to split the message into some blocks of k coefficients: k = 16 would imply that every second block is corrupted. Now insert into each block an error detecting checksum and split up the message such that only some of the blocks are required to reconstruct the message. Another possibility would be to use a Golay - Code for each block, ... Choose a method, which doesn't repeat a block (like the repetition code) and choose a method which can correct as most as much errors as in practice can appear. Otherwise the system is weakened against approximated key and direct lattice attacks (but on the other hand is strengthened against "Response attacks"[11]). In appendix B a possible solution is presented.

3.4 Example

As I have myself often experienced, a simple example with artificially small parameters is worth a thousand words. Hence here an example of the digital signature algorithm and the encryption algorithm is included:

Parameters

The following parameters are used: n = 3, p = 100, q = 10

Key generation

1. $x_1 = 17x^2 + 50x + 26,\ x_2 = 19x^2 + 14x + 4$
2. $g = 71x^2 + 74x + 94$
3. $f(x_1, g) = Tr(\begin{pmatrix} 17 & 50 & 26 \\ 50 & 26 & 17 \\ 26 & 17 & 50 \end{pmatrix} * \begin{pmatrix} 94 \\ 74 \\ 71 \end{pmatrix}) = Tr(\begin{pmatrix} 44 \\ 31 \\ 52 \end{pmatrix}) = \begin{pmatrix} 5 \\ 3 \\ 4 \end{pmatrix}$

   $f(x_2, g) = Tr(\begin{pmatrix} 19 & 14 & 4 \\ 14 & 4 & 19 \\ 4 & 19 & 14 \end{pmatrix} * \begin{pmatrix} 94 \\ 74 \\ 71 \end{pmatrix}) = Tr(\begin{pmatrix} 6 \\ 61 \\ 76 \end{pmatrix}) = \begin{pmatrix} 8 \\ 6 \\ 1 \end{pmatrix}$

Hence the public key is given by: $y_1 = 4x^2 + 3x + 5,\ y_2 = x^2 + 6x + 8,\ x_1$ and x_2. Now lets sign a message:

Signature generation (t = 1)

1 and 2. Because the computation of step 1 and 2 are actually the same as the key generation procedure, let here $r = g$ and $r_1 = y_1$ and $r_2 = y_2$. Hence: $r_1 = 4x^2 + 3x + 5,\ r_2 = x^2 + 6x + 8,\ r = 71x^2 + 74x + 94$

[11]These attacks tries to corrupt a choosen ciphertext in a way, such that your response - decryption succeded/failed - can be used to deduce the key.

3. Suppose the message $Hash(r_1\|r_2\|m)$ delivers the two polynomials: $e_1 = x^2,\ e_2 = x^2 - 1$
4. $s = \begin{pmatrix} 71 & 74 & 94 \\ 74 & 94 & 71 \\ 94 & 71 & 74 \end{pmatrix} * \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} + \begin{pmatrix} 71 & 74 & 94 \\ 74 & 94 & 71 \\ 94 & 71 & 74 \end{pmatrix} * \begin{pmatrix} -1 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 54 \\ 68 \\ 17 \end{pmatrix}$
5. See below.

The signature is now given by: $s = 17x^2 + 68x + 54$, r_1 and r_2.

Signature verification

1. $e_1 = x^2,\ e_2 = x^2 - 1$
2. $f(x_1, s) = Tr(\begin{pmatrix} 17 & 50 & 26 \\ 50 & 26 & 17 \\ 26 & 17 & 50 \end{pmatrix} * \begin{pmatrix} 17 \\ 68 \\ 54 \end{pmatrix}) = Tr(\begin{pmatrix} 60 \\ 57 \\ 10 \end{pmatrix}) = \begin{pmatrix} 1 \\ 6 \\ 6 \end{pmatrix}$

   $f(x_2, s) = Tr(\begin{pmatrix} 19 & 14 & 4 \\ 14 & 4 & 19 \\ 4 & 19 & 14 \end{pmatrix} * \begin{pmatrix} 17 \\ 68 \\ 54 \end{pmatrix}) = Tr(\begin{pmatrix} 46 \\ 51 \\ 46 \end{pmatrix}) = \begin{pmatrix} 5 \\ 5 \\ 5 \end{pmatrix}$
3. $v_1 = \begin{pmatrix} 4 & 3 & 5 \\ 3 & 5 & 4 \\ 5 & 4 & 3 \end{pmatrix} * \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} + \begin{pmatrix} 4 & 3 & 5 \\ 3 & 5 & 4 \\ 5 & 4 & 3 \end{pmatrix} * \begin{pmatrix} -1 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 5 \\ 6 \end{pmatrix}$

   $v_2 = \begin{pmatrix} 1 & 6 & 8 \\ 6 & 8 & 1 \\ 8 & 1 & 6 \end{pmatrix} * \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} + \begin{pmatrix} 1 & 6 & 8 \\ 6 & 8 & 1 \\ 8 & 1 & 6 \end{pmatrix} * \begin{pmatrix} -1 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 4 \\ 6 \\ 5 \end{pmatrix}$
4. $\Delta_1 = \begin{pmatrix} 1 \\ 6 \\ 6 \end{pmatrix} - \begin{pmatrix} 1 \\ 5 \\ 6 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$

   $\Delta_2 = \begin{pmatrix} 5 \\ 5 \\ 5 \end{pmatrix} - \begin{pmatrix} 4 \\ 6 \\ 5 \end{pmatrix} = \begin{pmatrix} 1 \\ -1 \\ 1 \end{pmatrix}$
5. $\sigma_1 \approx \sqrt{\frac{3*1}{12}} * 4 = \sigma * 4 = 0.5 * 4$, due to the small parameters p and q, however, $\sigma \approx 0.6$.
   No error coefficient is bigger the the treshold.
6. 4 coefficient are smaller than $\sigma = 0.6$, 2 are bigger as the standard deviation: This is predicted by normal distribution.
7. The signature is accepted.

Next, the encryption and decryption procedure is shown:

Encryption

1. $e_1 = -x + 1,\ e_2 = -x^2 + 1$

2. $c_1 = \begin{pmatrix} 17 & 50 & 26 \\ 50 & 26 & 17 \\ 26 & 17 & 50 \end{pmatrix} * \begin{pmatrix} 1 \\ -1 \\ 0 \end{pmatrix} + \begin{pmatrix} 19 & 14 & 4 \\ 14 & 4 & 19 \\ 4 & 19 & 14 \end{pmatrix} * \begin{pmatrix} 1 \\ 0 \\ -1 \end{pmatrix} = \begin{pmatrix} 99 \\ 19 \\ 82 \end{pmatrix}$

   $c_2' = \begin{pmatrix} 4 & 3 & 5 \\ 3 & 5 & 4 \\ 5 & 4 & 3 \end{pmatrix} * \begin{pmatrix} 1 \\ -1 \\ 0 \end{pmatrix} + \begin{pmatrix} 1 & 6 & 8 \\ 6 & 8 & 1 \\ 8 & 1 & 6 \end{pmatrix} * \begin{pmatrix} 1 \\ 0 \\ -1 \end{pmatrix} = \begin{pmatrix} 4 \\ 3 \\ 3 \end{pmatrix}$

3. M = 012 Mod 3: $m = 0 * \frac{10}{3}x^2 + 1 * \frac{10}{3}x + 2 * \frac{10}{3} = 0x^2 + 3x + 7$

4. $c_2 = \begin{pmatrix} 4 \\ 3 \\ 3 \end{pmatrix} + \begin{pmatrix} 0 \\ 3 \\ 7 \end{pmatrix} = \begin{pmatrix} 4 \\ 6 \\ 0 \end{pmatrix}$

The encrypted message consists of: $c_1 = 82x^2 + 19x + 99$ and $c_2 = 4x^2 + 6x + 0$

Decryption

1.$c_2' = f(c_1, s) = Tr(\begin{pmatrix} 82 & 19 & 99 \\ 19 & 99 & 82 \\ 99 & 82 & 19 \end{pmatrix} * \begin{pmatrix} 94 \\ 74 \\ 71 \end{pmatrix}) = Tr(\begin{pmatrix} 43 \\ 34 \\ 23 \end{pmatrix}) = \begin{pmatrix} 4 \\ 3 \\ 2 \end{pmatrix}$

2. $m = \begin{pmatrix} 4 \\ 6 \\ 0 \end{pmatrix} + \begin{pmatrix} 4 \\ 3 \\ 2 \end{pmatrix} = \begin{pmatrix} 0 \\ 3 \\ 8 \end{pmatrix}$

3. $M = 0 * \frac{3}{10}x^2 + 3\frac{3}{10}x + 8\frac{3}{10} = 0x^2 + 1x + 2 \; Mod \; 3$

4 Cryptanalysis of the new cryptosystem

In this section the security of the new scheme shall be discussed. First of all, these attacks (found by the author) could be used against the new scheme:

- **Direct attack against the private key g:** Try to approximate the polynomial g by lattice reduction techniques or using brute force (or using a technique to circumvent the one-way function f)
- **Attack against a single signature without knowledge of other signatures**: Try to approximate a single value s_2 (this is a bit easier as approximating g due to the allowed error tolerance).
- **Attack against the private key with knowledge of other signature**: Try to deduce the private key using the error information given by each signature.
- **Attack against a single encrypted message:** Try to find the polynomials e_1 and e_2 and recover with that the message-key.
- **Attack against the private key using a message response attack:** Use corrupted message and the others response (I could decrypt / not decrypt) to deduce the private key.

Now each possible attack is introduced and it's feasibility analysed. The following list of attacks may not be complete. It gives, however, insight information about how the suggested parameters were choosen and what has to be considered, when attempting to change these parameters. It must be further noted, that there may be more efficient possibilities to implement the following attacks.

4.1 Basic security considerations

The first security consideration to be made here is: n (the dimension of the cyclic lattice and polynomials) must be a prime number. Otherwise a trivial factorization for every modulus exists and therefore the lattice problems arising from the cryptosystem can be splitted into smaller ones.

Moreover, all polynomials that shall be choosen at random, must be choosen with an appropriate method having either a good random source or a starting seed of at least 128 bit. Otherwise a brute force attack could be possible.

Although brute force attacks exist, they are for the rest of this paper not considered a thread, because they are practically infeasible for dimension $n > 20$ for all parts of the scheme (signature generation, key generation, encryption).

4.2 Direct attack against the private key g

The public key is computed by applying the one-way function two times to x_1 and x_2 each. The first observation, that can be made is: All $x_i' = a*x_i \ Mod \ x^n - 1$ are equivalent (because $f(a * x, g) = Truncate(x * a * g \ Mod \ x^n - 1) = f(x, a * g)$). Hence a class of weak public parameters could exist (for example: $x_1 = a * (1)$, $x_2 = a * (256x^i)$ and p = 65521, the first polynomial gives away the upper half, the second polynomial the lower half of each coefficient of g). However, because x_1 and x_2 are choosen at random, it is very unlikely that such a weak key is being choosen. The next observation is, that a key with - for example 16000 bit - is mapped onto a public key with also 16000 bit. Therefore it can be assumed, that for every public key some equivalent private keys exist (for the example the other equivalent key would have been $g' = 38x^2 + 30x + 44$ - it maps onto the same public key and can be used normally for all operations). A bit probability math on one-to-one mappings, however, shows that there are in most cases at most two equivalent keys for a particular public key. Hence the lattice problem is not simplified enough. The last observation made in this paper about the public key is, that the private key is the optimal solution for the closest vector problem.

The private key can be computed by solving the closest vector problem using lattice reduction techniques on the following lattice:

$$\begin{pmatrix}
x_{1,n-1} & x_{1,n-2} & \cdots & x_{1,0} & x_{2,n-1} & \cdots & x_{2,0} & 0 \\
x_{1,n-2} & x_{1,n-3} & \cdots & x_{1,n-1} & x_{2,n-2} & \cdots & x_{2,n-1} & 0 \\
\vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\
x_{1,0} & x_{1,n-1} & \cdots & x_{1,1} & x_{2,0} & \cdots & x_{2,1} & 0 \\
-p & 0 & \cdots & 0 & 0 & \cdots & 0 & 0 \\
0 & -p & \cdots & 0 & 0 & \cdots & 0 & 0 \\
\vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\
0 & 0 & \cdots & -p & 0 & \cdots & 0 & 0 \\
0 & 0 & \cdots & 0 & -p & \cdots & 0 & 0 \\
\vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\
0 & 0 & \cdots & 0 & 0 & \cdots & -p & 0 \\
y_{1,n-1} * \frac{p}{q} & y_{1,n-2} * \frac{p}{q} & \cdots & y_{1,0} * \frac{p}{q} & y_{2,n-1} * \frac{p}{q} & \cdots & y_{2,0} * \frac{p}{q} & c
\end{pmatrix}$$

This lattice has a dimension of $3n + 1$ (e.g. for $n = 1021$ this would imply a dimension of 3064). No algorithm for computing the exact or a sufficient approximated solution in acceptable time for such big lattices is known today.

If an exact solution cannot be computed in practise, perhaps an approximated private key g' will suffice. Let's assume it would be possible to approximate g to a factor of 1.01 (i.e. $\|y_i'\|_1 \approx 1.01\|y_i\|_1$). This would imply a new single variable normal deviation of $\sigma \approx 2.4$ and a new signatures coefficient deviation of $\sigma_{Sig} \approx \sqrt{3t * 6}$. With such an big error signing messages would be impossible (because more than 70% of all error coefficients excede the treshold value). A similiar computation shows that decryption is infeasible because every coefficient will get corrupted by the introduced error.

As a result, it can be said, that with todays technology it is infeasible to approximate or compute the private key g in a way that the cryptosystem is compromised. It must be, however, noticed, that not a random lattice, but a cyclic modular lattice is used. It could be possible, that one day efficient techniques for cyclic lattices and modular lattices are found. As a matter of facts, this problem can be only approached by using appropriate keysizes (for modern computers and microcontrollers are keysizes of n = 1021 or n = 2039 neither a computational nor storage problem).

4.3 Attack against a single signature without knowledge of other signatures

An attacker can try to compute a single signature for a given document without knowledge of approximated private keys or other signatures. To do so, he has to find a signature s such that the signature verification algorithm is passed. That first of all means, that no absolute error coefficient is allowed to be greater than $3.5\sigma = 8$ for the suggested parameters.

Moreover, the signatures error polynomials are checked against the gaussian distribution. Using the recommded algorithm to accomplish this (see the appendix), even a small absolute extra error of on the average 0.1 per coefficient of the error polynomials Δ_i, causes signature rejection on 999 out of 1000 trials (for each trial you need a new polynomial s computed by lattice reduction techniques using a new r).

To crack a single signature, compute the signature algorithm up to step 3. (In step 4 s is computed). Now use the lattice given for breaking the private key and replace the last line (y_1 and y_2) bei ($v_i = e_1 * y_i + e_2 * r_i$) as computed during signature verification. Now a signature s is computed the approximates the v_i vector. Although that's a bit easier than cracking the private key g, it seems to be infeasible with todays lattice reduction techniques.

4.4 Attack against the private key with knowledge of other signatures

During signature verification the signatures error polynomial is checked. This error polynomial is the main difference between the new lattice based signature algorithm and a "normal" discrete logarithm zero knowledge proof. The basis signature algorithm doesn't leak any useful information to a possible attacker[12], the error polynomial, however, does. Let's have a closer look on the signatures verification algorithm:

$$\Delta_i = u_i - v_i = f(x_i, s) - e_1 * y_i - e_2 * r_i =$$

$$= Trunc(x_i * (e_1 * g + e_2 * r)) - e_1 * Trunc(x_i * g) - e_2 * Trunc(x_i * r)$$

$$\Delta_i = e_1 * Error(y_i, g) + e_2 * Error(r_i, r)$$

$Error(x, g)$ denotes here the reell polynomial given by:

$Error(x, g) = (Upsample(y_i) - x_i * g)/p * q$. Every coefficient of the polynomial of $Error(x, g)$ lies in the intervall $[-0.5; 0.5]$. The error is introduced by the no longer existing, already truncated part of the public key and the random values r_i. This error is mainly dominated by the random values (because the second zero knowledge proofs decision polynomial e_2 is weighted double). In each signature (when computed using the suggested parameters) only a maximum amount of 3 bits of information per error coefficient can be present. On average the are only 2 bits of information.

The error information can be used to deduce the upper parts of the truncated information of the public key [13] by solving the following closest vector problem:

$$\begin{pmatrix} e_{1,1}/q & e_{1,2}/q & \cdots & e_{1,k}/q & 0 \\ \Delta_{i,1} & \Delta_{i,2} & \cdots & \Delta_{i,k} & c \end{pmatrix}$$

[12] As has been proven a lot of times before

[13] the part, that has been truncated and that can be used to upsample a polynomial y_i and compute $g = y_i' * x^{-1}$

This notation means, that every row of that lattice contains actually all vectors of the cyclic lattice of $e_{1,j}$. The last line contains for every row a line of n coefficients of $\Delta_{i,j}$ and i is either 1 or 2. As the error polynomial of only one public key part is sufficient to recompute the public key, the other part mustn't be used. It must be noted, that this attack is (even for a large number of given signatures k) completely inefficient when using the suggested parameters. To see this (for the suggested parameter size), consider the effect: A normal error coefficient has a standard deviation $\sigma = 2.24$. Without the error introduced by the private key (by using the above lattice you try to subtract the correct error introduced by the private key), the standard deviation is reduced to $\sigma = 1.83$. Hence the coefficients are now smaller by an average value of 0.29. This small benefit isn't even sufficient to change most of the coefficients (because these are not reell numbers but integers). Moreover, the technique described above can be used only to compute approximately the two upper bits of the truncated information. Hence this method isn't useful to compute the private key.

4.5 Attack against a single encrypted message

To encrypt a message, two polynomials e_1 and e_2 are choosen and a lattice polynomial is computed. With these polynomials e_1 and e_2 the related message key can be computed. The most obvious way to crack a single encrypted message is to try to find these "decision" polynomials e_1 and e_2 or approximated version of these. Afterwards the message key can be computed and the message be decrypted. However, approximated polynomials e_1 and e_2 introduce larger errors and therefore are maybe not suitable to compute a message key. The approximated or exact polynomials can be computed by solving the closest vector problem using a similiar lattice as has been used to recover the secret key g. Using the suggested parameter sizes, it can be concluded, that for an approximation factor of 4 (i.e. $\|e_i'\|_1 \approx 4\|e_i\|_1$), it is infeasible to crack the messagekey, because the standard deviation of one coefficients error rises to $\sigma = 21$, hence approximately 80% of all message blocks (as defined in the section "Encryption algorithm") get corrupted.

4.6 Attack against the private key using a message response attack

During decryption an error is introduced by the one-way function. If this error is too big to be corrected, decryption fails. This fact can be used by an attacker. He chooses a ciphertext such that decryption only just works. This is done by increasing the error (which he doesn't know) by using succesively larger decision polynomials e_1 and e_2 and watching the response [14] of the legitimate recipient. He then gets information, what multiplier is needed to get an error. The author knows of no practical way to use this information because the error correcting

[14] Dependent on the scenario, this can be a timing attack - as error correction takes more time with more errors - or just a simple response like: "It worked", "It didn't work"

scheme can catch a lot of errors. Therefore you get only a lousy information about the introduced error.

5 Summary

In this paper a new cryptosystem usable for both public key encryption and digital signature generation was presented. Furthermore a first bit of cryptanalysis is given. The new "almost" linear homomorphic one-way function may emerge to a new cryptographic primitive function, that can be used for other applications like hash-functions, watermarking, biometric verification and hash-functions or more general: errorprone[15] security applications ... as well. The new algorithm, however, needs further analysis, to become a new trusted, secure alternative cryptographic algorithm.

Hence, every reader of this paper is encouraged, to send all his own ideas, reviews, suggestions, cryptoanalytical results, ... to the author in order to improve the cryptosystems. All reviews (positive or negative) sended to me, will be published on my website and are used to jmprove the new cryptosystem. If you are going to use this algorithm for any security critical process, wait at least one year (started with at publication date). If you just want to implement a working signature scheme for low security purposes (manipulation detection of virus scanners or serial number verification of software products), this algorithm is already secure enough to be used.

References

[1] Ming Kin Lai, Knapsack Cryptosystems: The Past and the Future, University of California, 2001

[2] Feng Bao, Yoshihide Igarashi, Break Finite Automata Public Key Cryptosystems, 1995

[3] J. Ding, J. E. Gower, D. S. Schmidt, Zhuang Zi: A new algorithm for solving multivariate polynomial equations over a finite field, University of Cincinnati

[4] Bruce Schneier, Applied Cryptography, Second Edition: Protocols, Algorithms, and Source Code in C, John Wiley and Sons, 1996

[5] M. Ajtai, C. Dwork, A public key cryptosystem with worst-case / average-case equivalence, 1996

[6] O. Goldreich, S. Goldwasser, S. Halevi, Public-Key Cryptosystems from Lattice Reduction Problems, 1997

[15]In a mathematical sense, i.e. error-prone inputs are provided

[7] O. Regev, Lattice-based Cryptography, Tel Aviv University

[8] www.ntru.com

[9] O. Goldreich, D. Micciancio, S. Safra, J.P. Seifert, Approximating shortest lattice vectors is not harder than approximating closest vectors

[10] O. Regev, Improved inapproximability of lattice and coding problems with preprocessing

[11] M. Alekhnovich, S. A. Khot, G. Kindler, N. K. Vishnoi, Hardness of Approximating the Closest Vector Problem with Pre-Processing

[12] R. C. Merkle, Public key distribution using approximately linear functions, Xerox Corp.

A Gaussian distribution test for the error polynomial Δ_i

During signature verification the error polynomial has to be checked against the expected gaussian distribution. Using the suggested parameter sets (t = 20, n = 1021), this can be done as follows:

```
 **Input:** An error polynomial $\Delta_i$ with coefficients $\delta_j$
**Output:** A boolean variable (true - accepted, false - rejected)
  For j=0 to n-1
    If $Abs(\delta_j) > 2$ then $c_1$++
    If $Abs(\delta_j) > 3$ then $c_2$++
    If $Abs(\delta_j) > 4$ then $c_3$++
    If $Abs(\delta_j) > 5$ then $c_4$++
  End For
   If $c_1 > 280$ then return false
   If $c_2 > 127$ then return false
   If $c_3 > 47$ then return false
   If $c_4 > 15$ then return false
   return true
```

B Encoding the message for encryption

Because the decryption process is error-prone, the message must be encoded using a suitable error correction scheme. Such a scheme is now presented:

1. Choose a random 128 bit key k.
2. Split the key into 7 13-digit base 3 numbers(e.g. 01220...)
3. Using a treshold sharing scheme, split each subkey, such that 3 out of 9 subkeys are sufficient to restore the original value
4. Multiply each subkey (now seen as polynomials Mod 3) with the CRC

polynomial given by 1002 ($= x^3 + 2$).
5. Attach all subkey-shares to form the message polynomial: the position determines the sharenumber.
6. Proceed with the encryption algorithm.

To decode this key, do the following:
1. By using the CRC polynomial, identify all corrupted polynomials.
2. Choose from the uncorrupted polynomials at random a set of shares, sufficient to recover all parts of the 128 bit key.
3. If not succesful, go to step 2 and choose again.