CFPKM : A Key Encapsulation Mechanism based on Solving System of non-linear multivariate Polynomials

20171129

Principal submitter

This submission is from the following team, listed in alphabetical order:

- O. Chakraborty, Sorbonne Uiversities/UPMC Univ Paris 06 and INRIA
- J.-C. Faugère, INRIA and Sorbonne Uiversities/UPMC Univ Paris 06
- L. Perret, Sorbonne Uiversities/UPMC Univ Paris 06 and INRIA

E-mail address (preferred): `ludovic.perret@lip6.fr`

Telephone (if absolutely necessary): +33-1-44-27-88-35

Postal address (if absolutely necessary):
Ludovic Perret,
Université Pierre et Marie Curie
LIP6- Équipe project INRIA/UPMC POLSYS
Boite courier 169
4 Place Jussieu
F-75252 Paris cedex 5, France

Auxiliary submitters: There are no auxiliary submitters. The principal submitter is the team listed above.

Inventors/developers: The inventors/developers of this submission are the same as the principal submitter. Relevant prior work is credited below where appropriate.

Owner: Same as submitter.

Signature: ×. See also printed version of "Statement by Each Submitter".

Document generated with the help of `pqskeleton` version 20170923.

Contents

1 Introduction 4

2 General algorithm specification (part of 2.B.1) 4

2.1 Parameter Space 4

2.2 Secret-key and Public-key 4

2.3 Functions in the KEM 5

2.3.1 KeyPair Generation 5

2.3.2 KeyEncapsulation 7

2.3.3 KeyDecapsulation 8

2.4 Correctness 9

2.5 Failure 14

3 List of parameter sets (part of 2.B.1) 14

3.1 Parameter set CFPKM128 14

3.2 Parameter set CFPKM182 15

4 Design rationale (part of 2.B.1) 15

5 Detailed performance analysis (2.B.2) 16

5.1 Description of platform 16

5.2 Time 16

5.3 Space 16

5.4 How parameters affect performance 16

6 Expected strength (2.B.4) in general 17

6.1 Security of the Key Encapsulation Mechanism 17

7 Expected strength (2.B.4) for each parameter set 18

7.1 Parameter set CFPKM128 18

7.2 Parameter set CFPKM182 18

8 Analysis of known attacks (2.B.5) 18

8.1 Usign Arora-Ge GB method 19

8.2 Exhuastive Search 20

8.2.1 Over the secret 20

8.2.2 Over the error 21

8.3 Hybrid attack 23

9 Advantages and limitations (2.B.6) 24

References 24

1 Introduction

The purpose of this document is to present CPFKM: a Key Encapsulation Mechanism Scheme. CPFKM is a based on the problem of solving a system of noisy non linear polynomials, also known as the PoSSo with Noise Problem. Applications based on PoSSo with Noise problem was first dealt by Faugere *et al.* in [2] introducing Symmetric Pollycracker Encryption schemes. Our scheme largly borrows its design rationale from key ecapsulation schemes based on the Learning With Errors(LWE) problem[11] and its derivatives. Some of these include the Peikert's Passivly secure KEM[10] and FRODO[6] which are based on the Ring-LWE problem[7]. The main motivation of buiding this scheme is to have a key exchange and encapsulation scheme based on the hardness of solving system of noisy polynomials. Not many attacks have been reported against PoSSo with noise problem. This is quite interesting as the PoSSo problem had been proposed quite a long time back and then very recently, [2] gave an encryption scheme.

2 General algorithm specification (part of 2.B.1)

2.1 Parameter Space

The scheme involves the following parameters :

- q, a large positive integer, which defines the finite field $\mathbb{F}_q$ for the Polynomial Ring $\mathbb{P}$, where we define our system of equations. It is taken of the form of 2^k for some $k \in \mathbb{Z}_+$,
- n, the number of variables which defines the Polynomial Ring $\mathbb{P}$,
- m, number of equations in the system of equations,
- s, is an integer which defines the range of values from where the secret and errors are chosen uniformly,
- B, is the number of most significant bits which are chosen to createa session key,

2.2 Secret-key and Public-key

The secret key is a concatenation of a random seed value and a secret vector $\mathbf{sa} \in < 0, s >^n$ chosen randomly from a uniform distribution $\mathcal{U}_s^n$. The seed helps to generate a system of polynomails $f'_1, \cdots, f'_n \in \mathbb{F}_q[x_1, \cdots, x_n]$ which is later used in the public key. It has the following structure,

$$SK = (seed||sa)$$

The public key in CPFKM is a concatenation of the same random seed value and vector $\mathbf{b_1} \in \mathbb{F}_q^m$. This vector $\mathbf{b_1}$ is result of solving the set of quadritic (may be higher degree also) polynomials with some added noise $f_1, \cdots, f_m \in \mathbb{F}_q[x_1, \cdots, x_n]$ over the chosen random secret value $\mathbf{sa}$. So each ith component of the vector is defined as follows

$$\mathbf{b_{1_i}} = f_i(\mathbf{sa})$$

and each f_i is a noisy polynomial of the structure

$$f_i(x_1, \cdots, x_n) = f_i'(x_1, \cdots, x_n) + e_i$$

where e_i is some noise chosen uniformly from the same range $< 0, s >$ as $\mathbf{sa}$. The public has the follwoing structure

$$PK = (seed||\mathbf{b1})$$

The Key Encapsulation is defined by three main algorithms namely KeyGen, Encapas and Decaps. The protocol has been has been summarized in Figure 1.

2.3 Functions in the KEM

2.3.1 KeyPair Generation

Function : crypto_kem_keypair(PK,SK)

The public key and the secret key are generated as a part of the KeypairGen function. The function generates a random value namely, *seed*. It takes use of another internal function called PolGen which, using the input of *seed*, generates a system of m multivariate quadritic polynomials $\{f_1(\mathbf{x}) \cdots f_m(\mathbf{x})\}$ in n variables where $\mathbf{x} = \{x_1, x_2, \cdots, x_n\}$. The coefficients of the polynomials are chosen from $[0, q^\alpha]$. The *seed*, which is an input to this function, is further an input to the random function which pseudo-randomly generates the coefficients.

This Polgen function has been summarized below

- For each of the m quadritic polynomial f_i create a structure of three vectors $QD \in \mathbb{Z}_{q^\alpha}^{n(n+1)/2)}, L \in \mathbb{Z}_{q^\alpha}^n, C \in \mathbb{Z}_{q^\alpha}$. The structure holds the coefficients of the polynomials such that QD holds the coefficients of the quadritic monomials, L holds the coefficients for the linear monomials and C holds the constant term,
- using a random function and the *seed*, populate these vectors
- return f_i

There after the KeyGen algorithm randomly generates a secret vector $\mathbf{sa}$ of dimension n from the uniform distribution $\mathcal{U}_s^n$. An error vector $e_1 \in \mathcal{U}_s^m$ is also generated. Each of the polynoamials f_i, from the earlier generated system of quadritic polynomials using Polgen,

Alice		**Bob**
KeyGen() :		
seed $\xleftarrow{\$} \{0,1\}^8$		
f ← PolGen(seed)		
$\mathbf{sa} \xleftarrow{\$} \mathcal{U}_s^n$		
$\mathbf{e_1} \xleftarrow{\$} \mathcal{U}_s^m$		
$\mathbf{b_1} \leftarrow f(\mathbf{sa}) + \mathbf{e_1}$		
PK ← seed$\|\|\mathbf{b_1}$, SK ← seed$\|\|\mathbf{sa}$		
	$\xrightarrow[\in \mathbb{Z}_q^{m\times 1} \times \{0,1\}^{\text{SEEDSIZE}}]{\text{PK}}$	
		Encaps():
		seed, $\mathbf{b1} \leftarrow unpack\_pk(PK)$
		f ← PolGen(seed)
		$\mathbf{sb} \xleftarrow{\$} \mathcal{U}_s^n$
		$\mathbf{e_2} \xleftarrow{\$} \mathcal{U}_s^m$
		$\mathbf{b_2} \leftarrow f(\mathbf{sb}) + \mathbf{e_2}$
		$\mathbf{e_3} \xleftarrow{\$} \mathcal{U}_s^m$
		$\mathbf{b_3} \leftarrow f(\mathbf{sb}) \odot \mathbf{b_1} + \mathbf{e_3}$
		$\mathbf{c} \leftarrow$ CrossRound$(\mathbf{b_3}, B)$
		Key$_B$ ←Rounding$(\mathbf{b_3}, B)$
		ct ← $\mathbf{c}\|\|\mathbf{b2}$
	$\xleftarrow[\in \mathbb{Z}_q^{m\times 1} \times \mathbb{Z}_2^{m\times 1}]{\text{ct}}$	
Decaps():		
$\mathbf{b_2}, \mathbf{c} \leftarrow unpack_ct(ct)$		
seed, $\mathbf{sa} \leftarrow unpack_sk(SK)$		
f ← PolGen(seed)		
Key$_A \leftarrow$ Red$(f(\mathbf{sa}) \odot \mathbf{b_2}, \mathbf{c}, B)$		

Figure 1: Our KEM Scheme based on PoSSo With noise

are evaluated over the secret vector sa and noise is added to them to generate another vector b_1 where

$$\mathbf{b_{1_i}} \leftarrow (f_i(\mathbf{sa}) + \mathbf{e_{1_i}})\ mod\ q$$

for ith component of the vector.

Finally the Public Key PK is made by concatenating the *seed* along with this vector **b1** using the *pack_pk* function. The Secret Key SK is formed by concatenating *seed* and the secret vector **sa** using the *pack_sk* funtion. The function then outputs the PK and SK.

2.3.2 KeyEncapsulation

Function : crypto_kem_enc(ct,SS,PK)

The encapsulation process encodes the shared secret using the public key of Alice. It takes use of some extra functions which have been defined below.

1. **CrossRound(w, B)**:This function takes in an integer $w \in [0, q)$ and given B, outputs the $(B+1)$'th most significant bit of $\log q$-bit binary representation of w, which has been referred to as the CrossRound bit.

   $$CrossRound(w, B) = \lfloor w \cdot 2^{-\bar{B}+1} \rfloor \ mod \ 2$$

   This function can be extended to a vector of integers $\mathbf{w} \in [0, q)^m$. Thus on an input of a vector, CrossRound works independetly on each component of vector and outputs another vector carrying the Crossround bit.

2. **Rounding(w, B)**: This function takes in an integer $w \in \mathbb{Z}_+$ and given B, outputs the B most significant bits of $\log q$-bit binary representation of $(w + 2^{\bar{B}-1})$ mod q , where $\bar{B} = Ceiling(\log q) - B$.

   $$Rounding(w, B) = \lfloor ((w + 2^{\bar{B}-1}) \ mod \ q) \cdot 2^{-\bar{B}} \rfloor$$

   where $\lfloor \cdot \rfloor$ is the Floor function. This function can be extended to a vector of integers $\mathbf{w} \in \mathbb{Z}_+^m$. Thus on an input of a vector, Rounding works independetly on each component of vector and outputs another vector carrying the Rounding value of each component.

3. $\odot$: This function takes in two vectors $\mathbf{a}$ and $\mathbf{b}$ returns a vector $\mathbf{y}$ which is a component wise scalar product of $\mathbf{a}$ and $\mathbf{b}$. I.e , $\mathbf{y}_i = \mathbf{a}_i \cdot \mathbf{b}_i$ for each i^{th} component of the vectors.

Thus the function for Key encapsulation follows the follwing procedure for creation and encapsulation of the shared key.

1. Encaps takes in the PK and then uses the *unpack_pk*[1] process to get $\mathbf{b_1}$ and the *seed*.
2. Uses the *seed* and the PolGen function to generate the same system of quadritic polynomials $\{f_1(\mathbf{x}) \cdots f_m(\mathbf{x})\}$ in n variables where $\mathbf{x} = \{x_1, x_2, \cdots, x_n\}$
3. Randomly sample vectors $\mathbf{sb} \leftarrow \mathcal{U}_s^n$, $e_2 \leftarrow \mathcal{U}_s^m$ and $e_3 \leftarrow \mathcal{U}_s^m$.
4. Computes $\mathbf{b_{2_i}} = (f_i(sb) + \mathbf{e_{2_i}}) \ mod \ q$ for each ith component of the vector.
5. Compute $\mathbf{b_{3_i}} = f_i(sb) \odot \mathbf{b_{1_i}} + \mathbf{e_{3_i}}$ for each of the i component.

---

[1] The details of the functions *pack_sk*, *unpack_sk*, *pack_pk*, *unpack_pk*, *pack_ct*, *unpack_ct* can be found in the repository

6. Uses the $\mathbf{CrossRound(b_3}, B\mathbf{)}$ function over the vector $\mathbf{b_3}$ to output a hint vector $\mathbf{c}$.
7. The key for Bob, $\mathbf{Key_{Bob}}$ is derived using the $\mathbf{Rounding(b_3}, B\mathbf{)}$ function thus giving the B most significant bits of each of the i component of $\mathbf{b_3}$.
8. Returns $ct = pack\_ct(\mathbf{b_2}, \mathbf{c})$ and SS $= \mathbf{Key_{Bob}}$.

2.3.3 KeyDecapsulation

Function : crypto_kem_dec(SS,ct,SK)

Alice does the decapsulation process which uses the ciphertext ct from Bob and uses Alice's secret key SK, to derive the shared secret key SS. The kem_dec function calls another function called Red. The function is described below.

Red(w, c, B): On input of vectors $\mathbf{w} \in \mathbb{Z}_+^m$ and $\mathbf{c} \in \mathbb{Z}_2^m$, Red$(\mathbf{w}, c, B)$ outputs Rounding(v, B), where $\mathbf{v}$ is the closest element to $\mathbf{w}$ such that CrossRound$(\mathbf{v}, B) =$ c. This function takes in $\mathbf{w} = f(\mathbf{sa}) \odot \mathbf{b_2}$, and follows the procedure below for each ith component of the vector independently,

- checks CrossRound$(\mathbf{w_i} \; mod \; q, B)$ if its equal to $\mathbf{c_i}$ or not. If its true, then it returns Rounding$(\mathbf{w_i}, B)$.
- If the value is false, then it adds $2^{\bar{B}-2} - 1$ to $\mathbf{w_i}$ and then checks if CrossRound$(\mathbf{w_i} + 2^{\bar{B}-2} - 1 \; mod \; q, B)$ is equal to $\mathbf{c_i}$ or not. If true then returns Rounding$(\mathbf{w_i} + 2^{\bar{B}-2} - 1, B)$
- If still false, then substracts $2^{\bar{B}-2} - 1$ from $\mathbf{w_i}$ and checks if CrossRound$(\mathbf{w_i} - 2^{\bar{B}-2} + 1 \; mod \; q, B)$ is equal to $\mathbf{c_i}$ or not. If true then returns Rounding$(\mathbf{w_i} - 2^{\bar{B}-2} + 1, B)$.
- If still false, then it returns 0 .

So the decapsulation function follows the steps below

1. Uses $unpack\_sk$(SK) to get the secret vector sa and $seed$ used by Alice earlier to generate her public key PK.
2. Unpacks the ciphertext ct using $unpack\_ct$, to get $\mathbf{b_2}$ and the hint vector $\mathbf{c}$
3. Use the $seed$ to generate the same system of polynomials f_i.
4. Computes $f_i(sa) \odot \mathbf{b_{2_i}}$ for each ith component.
5. Calls the Red function on the input of $f(\mathbf{sa}) \odot \mathbf{b_2}$, the hint vector $\mathbf{c}$ and B, the number of bits over which the key reconciliation has been agreed upon. The Red function outputs SS $= \mathbf{Key_{Alice}}$.
6. Returns $\mathbf{Key_{Alice}}$.

2.4 Correctness

We consider a large modulus q as a power of 2. For a choice of $1 \leq B < \log q - 1$ let, $\bar{B} = \log q - B$. We also consider that We define the following terms just for purpose of our proof.

Definition 1 (Interval)**.** *A set of* $2^{\bar{B}}$ *consecutive positive integers which is represented as* $< i \cdot 2^{\bar{B}}, (i+1) \cdot 2^{\bar{B}} - 1 >$ $\ for\ i = 0\ to\ \infty$*.*

Definition 2 (Subinterval)**.** *A set of* $2^{\bar{B}-1}$ *consecutive positive integers which is represented as* $< i \cdot 2^{\bar{B}-1}, (i+1) \cdot 2^{\bar{B}-1} - 1 >$ *for* $i = 0$ *to* ∞*.*

Thus an interval has positive integers with the same exact most significant bits except the $\bar{B}$ least significant bits in their binary representation, whereas a subinterval has positive integers with the same most significant bits except the $\bar{B} - 1$ least significant bits. Thus, a subinterval splits up an interval equally according to their $\bar{B}$th least significant bit.

Let us denote a simple modulus map $h : \mathbb{Z}_+ \rightarrow < 0, q-1 >$ as

$$h(v) = v\ mod\ q$$

Lemma 1. *Suppose we have a large modulus* q *being a power of* 2*. With the definition of the modulus map as above, a subinterval* $I \in \mathbb{Z}_+$*, maps to a subinterval in* $< 0, q-1 >$*, i.e* $h(I) \in < 0, q-1 >$ *is another subinterval.*

Proof. Now we have $q = 2^k$, such that k is large enough, then $q\ mod\ 2^{\bar{B}-1} = 0$. Thus it is a starting value/point of some subinterval. So for any subinterval $I =< i \cdot 2^{\bar{B}-1}, (i+1) \cdot 2^{\bar{B}-1} - 1 >$,

$$\begin{aligned}
h(I) &=< i \cdot 2^{\bar{B}-1}\ mod\ q, ((i+1) \cdot 2^{\bar{B}-1} - 1)\ mod\ q > \\
&=< i \cdot 2^{\bar{B}-1}\ mod\ 2^k, ((i+1) \cdot 2^{\bar{B}-1} - 1)\ mod\ 2^k > \\
&=< i \cdot 2^{\bar{B}-1}\ mod\ (2^{k-\bar{B}+1} \cdot 2^{\bar{B}-1}), ((i+1) \cdot 2^{\bar{B}-1} - 1)\ mod\ (2^{k-\bar{B}+1} \cdot 2^{\bar{B}-1}) > \\
&=< (i\ mod\ 2^{k-\bar{B}+1}) \cdot 2^{\bar{B}-1}, (((i+1)\ mod\ 2^{k-\bar{B}+1}) \cdot 2^{\bar{B}-1} - 1)\ mod\ 2^k > \\
&=< (i\ mod\ 2^{k-\bar{B}+1}) \cdot 2^{\bar{B}-1}, ((i\ mod\ 2^{k-\bar{B}+1} + 1) \cdot 2^{\bar{B}-1} - 1) \mod 2^k > \\
&=< j \cdot 2^{\bar{B}-1}, ((j+1) \cdot 2^{\bar{B}-1} - 1)\ mod\ 2^k >
\end{aligned}$$

Now $j < 2^{k-\bar{B}+1}$, this implies that $(j+1) \cdot 2^{\bar{B}-1} \leq 2^k$, which means that $(j+1) \cdot 2^{\bar{B}-1} - 1 < 2^k$.Hence we can write

$$h(I) =< j \cdot 2^{\bar{B}-1}, (j+1) \cdot 2^{\bar{B}-1} - 1 >$$

where $j = (i\ mod\ 2^{k-\bar{B}+1})$ is an integer. We see that $h(I)$ is also has a form of an subinterval. Thus any subinterval $\in \mathbb{Z}_+$ is mapped to some subinterval $I' = h(I) \subset < 0, q-1 >$. ☐

We assume that any integer $\in < 0, q-1 >$ **has a binary representation in** $\log q$ **bits.**

Lemma 2. *For a large modulus $q = 2^k$, when two positive integers v and w lie in the same subinterval, their CrossRound bits are same and when in adjacent intervals, then their CrossRound bits are different.*

Proof. Let's assume they lie in the same subinterval I, then

$$v, w \in I =< i \cdot 2^{\bar{B}-1}, (i+1) \cdot 2^{\bar{B}-1} - 1 >$$

for some particular value of i. From the definition of the mapping h, we see that $v \ mod \ q \in h(I)$ and $w \ mod \ q \in h(I)$, i.e they are in the same subinterval $h(I)$. This implies that the $\bar{B}$th least significant bit of $v \ mod \ q$ and $w \ mod \ q$ are the same, since in an subinterval all the bits except the $\bar{B} - 1$ least significant bits are same for all intergers in the subinterval (Definition 2). And from the definition of the CrossRound function (Section **??**), this $\bar{B}$th least significant bit is the CrossRound bit. Hence when both v and w are in the same subinterval, their CrossRound bits are equal.

Now let v and w lie in two adjacent subintervals. We denote the two subintervals as $v \in I_1 =< i \cdot 2^{\bar{B}-1}, (i+1) \cdot 2^{\bar{B}-1} - 1 >$ and $w \in I_2 =< (i+1) \cdot 2^{\bar{B}-1}, (i+2) \cdot 2^{\bar{B}-1} - 1 >$ for some i. Then I_1 maps onto some subinterval $I'_1 = h(I_1) \subset< 0, q-1 >$ and I_2 onto $I'_2 = h(I_2) \subset< 0, q-1 >$. Here

$$I'_1 = h(I_1) =< j \cdot 2^{\bar{B}-1}, (j+1) \cdot 2^{\bar{B}-1} - 1 >$$

$$I'_2 = h(I_2) =< ((j+1) \ mod \ 2^{k-\bar{B}+1}) \cdot 2^{\bar{B}-1}, ((j+2) \ mod \ 2^{k-\bar{B}+1}) \cdot 2^{\bar{B}-1} - 1 >$$

where $j = (i \ mod \ 2^{k-\bar{B}+1})$. As $j \neq ((j+1) \ mod \ 2^{k-\bar{B}+1})$, hence $I'_1 \neq I'_2$, i.e they dont map on to the same subinterval in $< 0, q-1 >$.
It is also important to note that as h is just a simple modulus map and the modulus q is a power of 2, so for any $v \in \mathbb{Z}_+$, $h(v)$ is the $\log q$ least significant bits of binary representation of v. Hence there is no change in the $\bar{B}$th least significant bit after the modulo operation.

As I_1 and I_2 are two adjacent subintervals, they differ by their $\bar{B}$th least significant bit. This implies that $h(I_1)$ and $h(I_2)$ have different $\bar{B}$th least significant bit, as the mapping doesnt change the $\log q$ least significant bits of any integer in I_1 or I_2.
Now $v \ mod \ q \in h(I_1)$ and $w \ mod \ q \in h(I_2)$. So their CrossRound bits, which is the $\bar{B}$th least significant bit are different. ☐

Thus now to prove the correctness, we propose the following Theorem.

Theorem 1. *Let the choice of a large modulus q which is a power of 2. Let us represent for any ith component of the vectors $\mathbf{b3}$ and $f(\mathbf{sa}) \cdot \mathbf{b_2}$ as $v = \mathbf{b_{3_i}} \in \mathbb{Z}_+$ and $w = f_i(\mathbf{sa}) \cdot \mathbf{b_{2_i}} \in \mathbb{Z}_+$. The above Key exchange protocol is correct with a high probailty i.e $\mathbf{K_{Alice}} = \mathbf{K_{Bob}}$ when for all i components of the vectors of dimension m,*

$$w \in (v' - 2^{\bar{B}-2} - 1, v] \cup [v, v'' + 2^{\bar{B}-2} + 1)$$

where $v' = \lfloor v \cdot 2^{-\bar{B}+1} \rfloor \cdot 2^{\bar{B}-1}$*,* $v'' = v' + 2^{\bar{B}-1}$ *and* $\bar{B} = Ceiling(\log q) - B$*.* B *is the number of most significant bits chosen for key agreement such that* $1 \leq B < Ceiling(\log q) - 1$ *and* $|\cdot|$ *gives the absolute value component wise.*

Proof. Let us consider the integers, $\mathbf{b_{3_1}} = v \in \mathbb{Z}_+$ and $f_1(\mathbf{sa}) \cdot \mathbf{b_{2_1}} = w \in \mathbb{Z}_+$. Consider the subintervals I_1 and I_2 such that $v \in I_1$ and $w \in I_2$.

1. **Case 1** : When v and w lie in the same subinterval,i.e $I_1 = I_2$. So from Lemma 2, we conclude that CrossRound(v mod q,B) = CrossRound(w mod q,B).
   Thus Alice performs Rounding(w, B) as a part of the Red function to get $\lfloor((w + 2^{\bar{B}-1})\ mod\ q) \cdot 2^{-\bar{B}}\rfloor$ and Bob does Rounding(v, B)= $\lfloor((v + 2^{\bar{B}-1})\ mod\ q) \cdot 2^{-\bar{B}}\rfloor$.
   Since both v and w are in same subinterval, this implies that $v+2^{\bar{B}-1}$ and $w+2^{\bar{B}-1}$ are also in same subinterval. Now from Lemma 1,we can infer that both $(v + 2^{\bar{B}-1}\ mod\ q)$ and $(w + 2^{\bar{B}-1}\ mod\ q)$ lie again in the same subinterval in $< 0, q-1 >$. We assumed that any interger in $< 0, q-1 >$ has a $\log q$-bit binary representation, Hence the B most significant bits for both are also equal, as in a subinterval for any two integers all the bits excpet the $\bar{B}-1$ least significant bits are equal. Hence we conclude that
   $$\lfloor((w + 2^{\bar{B}-1})\ mod\ q) \cdot 2^{-\bar{B}}\rfloor = \lfloor((v + 2^{\bar{B}-1})\ mod\ q) \cdot 2^{-\bar{B}}\rfloor$$
   $$\implies \mathbf{K_{Alice}} = \mathbf{K_{Bob}}$$

2. **Case 2**: When v and w lie in adjacent intervals. From Lemma 2, CrossRound(v mod q,B) $\neq$ CrossRound(w mod q,B).
   So, according to the Red function, first we add $2^{\bar{B}-2} - 1$ to w to get $w' = w + 2^{\bar{B}-2} - 1$. Now, as per Lemma 3, only two subcases are possible, either $w' \in I_1$ or $w' \in I_2$.

   If $w' \in I_1$, then this implies that by Lemma 2, CrossRound(v mod q,B) = CrossRound(w' mod q,B). This is another instance of Case 1. Therefore Alice performs Rounding(w', B) inside the Red function and Bob computes Rounding(v, B). As per Case 1 we conclude, $\mathbf{K_{Alice}} = \mathbf{K_{Bob}}$.

   Now consider the other subcase, i.e when $w' \in I_2$. By Lemma 2 it means that CrossRound(v mod q,B) $\neq$ CrossRound(w' mod q,B). Hence following steps of the Red function, we substract, $2^{\bar{B}-2} - 1$ from w to get $w'' = w - 2^{\bar{B}-2} + 1$. Now again two subcases are possible (Lemma 3), $w'' \in I_1$ or $w'' \in I_2$.

   If $w'' \in I_1$, then we have CrossRound(v mod q,B) = CrossRound(w'' mod q,B) by Lemma 2. We thus find another instance of Case 1. Hence, Alice does Rounding(w'', B) and Bob does Rounding(v, B), which gives us $\mathbf{K_{Alice}} = \mathbf{K_{Bob}}$.

   Now let us look at the remaining case of $w'' \in I_2$. At this stage of Red function, we already have that $w' \in I_2$ and $w \in I_2$ while $v \in I_1$ and $I_1 \neq I_2$. $w' \in I_2$ and $w'' \in I_2$ imples that w lies in middle of the subinterval I_2. This means that
   $$w \notin (v' - 2^{\bar{B}-2} - 1, v' - 1] \cup I_1 \cup [v'' + 1, v'' + 2^{\bar{B}-2} + 1)$$

But this is in contradiction to our initial assumption.

3. **Case 3**: When v and w lie in two different subintervals separated by atleast one interval. This implies that $|w - v'| > 2^{\bar{B}-1}$ or $|w - v''| > 2^{\bar{B}-1}$. From our assumption, we have that if $w \notin I_1$, then either $|w - v'| < 2^{\bar{B}-2} - 1$ or $|w - v''| < 2^{\bar{B}-2} - 1$. We find a clear contradiction to our assumption.

So, if the assumption of our theorem holds, after the Rounding operation, the protocol produces the same key for Alice and Bob. ☐

Lemma 3. *Suppose we have two subintervals I_1 amd I_2 such that for two positive intergers, $v \in I_1$ and $w \in I_2$. We also have*

$$w \in (v' - 2^{\bar{B}-2} - 1, v' - 1] \cup I_1 \cup [v'' + 1, v'' + 2^{\bar{B}-2} + 1)$$

where $v' = \lfloor v \cdot 2^{-\bar{B}+1} \rfloor \cdot 2^{\bar{B}-1}$ and $v'' = v' + 2^{\bar{B}-1}$. For $w' = w + 2^{\bar{B}-2} - 1$ and $w'' = w - 2^{\bar{B}-2} + 1$, only two cases are possible, w' and w'' are either in I_1 or I_2.

Proof. To see that this is true, suppose $w' \notin I_1$ and $w' \notin I_2$. This means that $|w' - v''| > 2^{\bar{B}-1}$. Which further implies that $|w - v''| > 2^{\bar{B}-2} - 1$, which is in clear contradiction to our initial assumption about w being in the range given. So w' must lie in either I_1 or I_2. The proof is similar for w''. ☐

Now the assumption in the Theorem 1 is dependent on the range s. We would like to determine the range by fixing the rest of the parameters. Let the choice of q be 2^k. The choice of the range from which the error and secret is chosen, let's suppose be $s = Round(2^\beta)$. From Theorem 1, we get

$$|\mathbf{b_3}_i - f_i(\mathbf{sa}) \cdot \mathbf{b_2}_i| < 2^{(\log q - B - 2)}$$

$$\implies |\mathbf{e_3}_i + \mathbf{e_1}_i \cdot \mathbf{b_2}_i - \mathbf{e_2}_i \cdot \mathbf{b_1}_i| < 2^{(\log q - B - 2)}$$

$$\implies s + s \cdot |\mathbf{b_1}|_{max} + s \cdot |\mathbf{b_2}|_{max} < 2^{(\log q - B - 2)}$$

If we replace $|\mathbf{b_1}|_{max}$ and $|\mathbf{b_2}|_{max}$ by $|b| = max(|\mathbf{b_1}|_{max}, |\mathbf{b_2}|_{max})$ we get

$$s + 2 \cdot s \cdot |b| < 2^{(\log q - B - 2)} \tag{1}$$

The choice of the coefficients is from $[0, q^\alpha]$, hence the maximum possible value of a coeffcient is $q^\alpha \approx 2^{\alpha k}$. First we need to determine the maximum possible value for $|b|$.
Now,

$$b = f(s) + e$$

$$b = (\sum_{i,j}^{n} a_{ij} x_i x_j + \sum_{i}^{n} b_i x_i + c) + e$$

$$= \mathcal{O}(n^2) \cdot 2^{\alpha k} \cdot 2^{2\beta} + n \cdot 2^{\alpha k} \cdot 2^{\beta} + 2^{\alpha k} + 2^{\beta}$$

Now that we have a maximum value of $|b|$ in terms of n, we can now look at LHS of Eqn 1,

$$s + 2 \cdot s \cdot |b|$$

Replacing the corresponding values we obtain

$$LHS = 2^{3\beta}2^{2\log n+\alpha k+1} + 2^{2\beta}(2^{\log n\alpha k+1}+1) + 2^{\beta}(2^{\alpha k+1}+1) \tag{2}$$

RHS of Eqn 1 is $2^{\log q-B-2}$. Replacing we get

$$RHS = \frac{2^k}{2^{B+2}}$$

So putting together the LHS and RHS of Eq 1, we have

$$2^{3\beta}2^{2\log n+\alpha k+1} + 2^{2\beta}(2^{\log n\alpha k+1}+1) + 2^{\beta}(2^{\alpha k+1}+1) < \frac{2^k}{2^{B+2}} \tag{3}$$

Corollary 1 (Asymptotic Result)**.** *With the choice of large modulus $q = 2^k$, the above proposed protocol succeeds with high probabilty , if following holds*

$$\beta < \left(\frac{k(1-\alpha) - B - 2 - 2\log n}{3}\right)$$

Corollary 1 gives an approximate upper bound for the range $s = 2^{\beta}$. Thus choosing s the range accordingly (Ref. figure 2) for sampling our error and secret, our key exchange and agreement works correctly. Although for accurate working of the protocol with probabilty 1, we need a much stricter bound which can be obtained by solving the Eq 3 for the variable β, in terms of the other variables α, B and k. A good choice of the other parameters is $k > 7\log n$, $B = 25$ and $\alpha = 0.3$

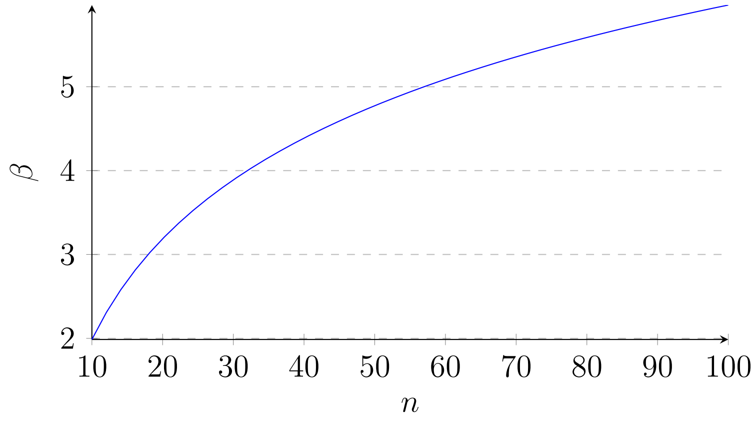

Figure 2: Plot showing the relationship of β, the factor influencing the range s, with n for a choice of $k = 8\log n$, $B = 4$ and $\alpha = 0.3$

The choice for B, the number of most significant bits chosen per sample has been set to 4(for our recomended parameters), as compared to the reconciliation mechanism by Peikert [10] which extracts a single bit per sample. This choice is backed by the fact that an exhaustive search for directly finding the shared secret will take at least 2^{4m} operations and our goal is that $4m$ should be larger than atleast our initial target of 128 bits of security.

2.5 Failure

The assumption on which our key agreement works is that $f_i(\mathbf{sa}) \cdot \mathbf{b_2}_i$ lies in a range of integers $(v' - 2^{\bar{B}-2} - 1, v] \cup [v, v'' + 2^{\bar{B}-2} + 1)$, where $v = \mathbf{b_{3_i}}$, $v' = \lfloor v \cdot 2^{-\bar{B}+1} \rfloor \cdot 2^{\bar{B}-1}$ and $v'' = v' + 2^{\bar{B}-1}$. When the difference $|\mathbf{b_3}_i - f_i(\mathbf{sa}) \cdot \mathbf{b_2}_i|$ is less than $< q/2^{B+2}$, then it always works with probabilty 1. When the difference is $\geq 3q/2^{B+2}$ then the probabilty is 0. In between the two extremes of $2^{\bar{B}-2}$ and $3 \cdot 2^{\bar{B}-2}$, the probability of success decreases linearly. Let us denote the probabilty of success for a component of the m dimensional system as p_s. So

$$p_s = \begin{cases} 1, & \text{if } |v-w| < 2^{\bar{B}-2} \\ 0, & \text{if } |v-w| \geq 3 \cdot 2^{\bar{B}-2} \\ \frac{3 \cdot 2^{\bar{B}-2} - |v-w|}{2^{\bar{B}-1}} & \text{if } 2^{\bar{B}-2} \leq |v-w| < 3 \cdot 2^{\bar{B}-2} \end{cases} \tag{4}$$

Now the probabilty distribution of the distance $|v - w|$ is uniform , since all the parameters effecting the distance, namely, the error terms e_1, e_2, e_3 and coefficients of f_i and the secrets sa and sb all follow an uniform distribution. So, from equation 2, we can see that the error is bounded by

$$\text{Maxerr}(\beta) = 2^{3\beta} 2^{2\log n + \alpha k + 1} + 2^{2\beta}(2^{\log n\alpha k + 1} + 1) + 2^{\beta}(2^{\alpha k + 1} + 1)$$

So, the total probabilty of success of the scheme is given by,

$$P_s(\beta) = \begin{cases} \left(\sum_0^a \frac{1}{\text{Maxerr}(\beta)} \right)^m = 1, & \text{if Maxerr}(\beta) < 2^{\bar{B}-2} \\ \left(\sum_0^a \frac{1}{\text{Maxerr}(\beta)} + \sum_{i=a+1}^b \frac{3 \cdot 2^{\bar{B}-2} - i}{2^{\bar{B}-1} \cdot \text{Maxerr}(\beta)} \right)^m, & \text{otherwise.} \end{cases} \tag{5}$$

where $a = min(2^{\bar{B}-2} - 1, \text{Maxerr}(\beta))$, $b = min(3 \cdot 2^{\bar{B}-2} - 1, \text{Maxerr}(\beta))$ and $m = n + 1$ is the number of samples that we use for our system/scheme. Thus the probabilty of failure of the scheme is given by $P_f(\beta) = 1 - P_s(\beta)$.

3 List of parameter sets (part of 2.B.1)

Following the analysis of Section 8, we propose below a set of 2 parameters for 128 and 182 bits of classical security.

3.1 Parameter set CFPKM128

We choose $\log q = 50$, $n = 80$, $m = 81$, $B = 6$, $SEEDSIZE = 48$ bytes, $COFSIZE = 4096$ and $RANGE = 7$. This gives a public key of size 696 bytes, secret key of 128 bytes and a shared secret of size 81 bytes.

3.2 Parameter set CFPKM182

We choose $\log q = 55$, $n = 115$, $m = 116$, $B = 6$, $SEEDSIZE = 67$ bytes, $COFSIZE = 16384$ and $RANGE = 6$. This gives a public key of size 928 bytes, a secret key of 182 bytes and a shared secret of 116 bytes.

4 Design rationale (part of 2.B.1)

In the past few years, multivariate schemes[12, 5] have been gathering a lot of attension in the crypto world. New public key encryptions and signatures based on multivariate cryptography have been proposed. But Key encapsulation Mechanisms and Key exchange/agreement schemes based on multivariate cryptography has not been touched upon very much. The main rationale of creating a Key Encapsulation mechanism based on multivariate non linear polynomials is the idea of a new problem based on solving a system of non-linear noisy equations , also known as the PoSSo with noise problem.

Definition 3. *Let $\mathbb{K}$ be a finite field and $\mathbb{P} = \{f_1, \cdots, f_m\} \subset \mathbb{K}[x_1, \cdots, x_n]$ be a system of polynomials. The problem of polynomial system with noise (PoSSo with Noise) is finding a solution $(x_1, \cdots, x_n) \in \mathbb{F}^n$ such that for all $f_i \in \mathbb{K}$, we have $f_i(x_1, \cdots, x_n) + e_i = 0$, where e_i is error chosen from a certain distribution.*

. This problem has similar bearings to the Learning with Error problems, with a distinct difference of this being non-linear.That is why the PoSSo with noise problem can be possibly stated in both its search as well as decision versions.

Search PoSSo With noise. The problem of finding a solution to a system of noisy non-linear equations.

Decision PoSSo with noise. The problem of deciding whether a solution is an actual solution to the system of noisy non linear equations.

This above definition of hardness of Decision PoSSo With noise problem is the main rationale behind developing this scheme. In order to break our protocol, the main motivation is to find the secret sa or sb used by either of Alice or Bob. We have a system of equations

$$b_{1_1} = f_1(\mathbf{x}) + e_{1_1}$$

$$b_{1_2} = f_2(\mathbf{x}) + e_{1_2}$$

$$\vdots$$

$$b_{1_m} = f_m(\mathbf{x}) + e_{1_m}$$

This is a system of non-linear multivariate polynomials, whose solution is $\mathbf{x} = sa$. The process of finding this solution is the exact hard problem of Search *PoSSo with Noise*, while if an adversary is able to find a candidate solution, the problem of deciding whether it is an actual solution or not is the problem of Decision PoSSo With noise. The Posso with Noise problem has been proved to be NP-hard [2]

5 Detailed performance analysis (2.B.2)

5.1 Description of platform

Computer	OS	Architecture	Processor	Frequency	RAM	Version of gcc
Laptop	Linux Mint 18.1	x86_64	i7-6600U	2.60 GHz	31.3 GiB	gcc 5.4.0

Table 1: Materials

5.2 Time

The following measurements are for the KEM. For the measures, it runs a number of tests such that the global used time is greater than 10 seconds and the global tim eis divided by the number of tests. For our scheme with CFPKM128 the key generation, takes 72 ms. The encapsulation scheme takes on a average about 108 ms(over a run of 30 tests) The decapsulation of the shared secret key takes about 143 ms.

5.3 Space

Sizes are straightforwardly calculated from parameters (and confirmed in various experiments). From the structure of public key we see that it involves the seed and the public vector **b1**. The total size of public key for CFPKM128 turns out to be 696 bytes. The secret key is 128 bytes for the parameters of CFPKM128. The ciphertexts are 729 bytes long, whereas the shared secret is of 81 bytes.

5.4 How parameters affect performance

The Key Encapsulation is mainly effected by the number of equations m in our system, the range s and also by the number of most significant bits B that we use in our scheme. From Section 2.5, we see that the schemes failure probabilty is a function of the range s. his probabilty is over the m equations used in the scheme. So the efficiency of the scheme is dependent on both s and m. Also from Section 8, the fastest attack, exhaustive search over the secret, tells us that the security of the scheme is a factor of both s and m.

6 Expected strength (2.B.4) in general

The security of our key encapsulation mechanism can be reduced down to the PoSSo with noise problem.

6.1 Security of the Key Encapsulation Mechanism

Theorem 2. *Given two samples, our KEM is IND-CPA secure, assuming the hardness of Posso with noise. In other words, for any adversary $\mathcal{A}$, there exists an adversary $\mathcal{B}$ such that*

$$Adv_{KEM}^{CPA} \leq 2 \cdot Adv_{n,m}^{PoSSoWN}(\mathcal{B})$$

Proof. It proceeds by the sequence of games shown in Figure 3. Let $\mathcal{E}_i$ be the event that the adversary guesses the bit b^* in Game i.

Game 0. Let $\mathcal{A}$ be an adversary that is executed in the IND-CPA attack game. Let us call it as Game 0. Now

$$\mathrm{Adv}_{\mathrm{KEM}}^{\mathrm{CPA}} = |Pr[\mathcal{E}_0] - 1/2| \tag{6}$$

Game 1 In this game, the Public key is chosen uniformly randomly. It is possible to verify that there exists an adversary $\mathcal{B}$ with the same running time as that of $\mathcal{A}$ such that

$$|Pr[\mathcal{E}_0] - Pr[\mathcal{E}_1]| \leq \mathrm{Adv}_{\mathrm{n,m}}^{PoSSoWN}(\mathcal{B}) \tag{7}$$

In other words, Game 0 and Game 1 are computationally indisguinshable from each other under the Decision PoSSo with Noise hardness assumption.

Game 2. in this game, the values of ct and K in place of being calculated through the Encapsulation process, are replaced by uniform random values. In Game 2, the values ct and K are chosen from uniform disributionm thus the adversary has no information about the bit b^* and therefore

$$Pr[\mathcal{E}_2] = 1/2 \tag{8}$$

Again there exists an adversary $\mathcal{B}$ with the same running time as that of $\mathcal{A}$ such that

$$|Pr[\mathcal{E}_1] - Pr[\mathcal{E}_2]| \leq \mathrm{Adv}_{\mathrm{n,m}}^{PoSSoWN}(\mathcal{B}) \tag{9}$$

Thus the Game 1 and Game 2 are again computationallty indisnguishable. Thus we can conclude from the sequence [6-9] of equations we get the required result.

□

```
Game 0:
1.$f \leftarrow^{\$} GF(q^{\alpha})[x_1, \cdots x_n]$
2. PK,SK $\leftarrow$KeyGen$_1(f)$

3.ct,K $\leftarrow$ Encaps$_1$(PK)

4. K'$\leftarrow^{\$} \mathcal{U}^{256}(0,1)$
5. $b^* \leftarrow^{\$} \{0,1\}$
6. if $b^* = 0$
Return(f,PK,ct,K)
7. else
Return(f,PK,ct,K')
```

```
Game 1:
1.$f \leftarrow^{\$} GF(q^{\alpha})[x_1, \cdots x_n]$
2. SK $\leftarrow \mathcal{U}$
3. PK $\leftarrow^{\$} \mathcal{U}$
4. ct,K $\leftarrow$ Encaps$_1$(PK)

5. K'$\leftarrow^{\$} \mathcal{U}^{256}(0,1)$
6. $b^* \leftarrow^{\$} \{0,1\}$
7. if $b^* = 0$
Return(f,PK,ct,K)
8. else
Return(f,PK,ct,K')
```

```
Game 2:
1.$f \leftarrow^{\$} GF(q^{\alpha})[x_1, \cdots x_n]$
2. SK $\leftarrow \mathcal{U}$
3. PK $\leftarrow^{\$} \mathcal{U}$
4. ct$\leftarrow^{\$} \mathcal{U}$
5. K$\leftarrow^{\$} \mathcal{U}$
6. K'$\leftarrow^{\$} \mathcal{U}^{256}(0,1)$
7. $b^* \leftarrow^{\$} \{0,1\}$
8. if $b^* = 0$
Return(f,PK,ct,K)
9. else
Return(f,PK,ct,K')
```

Figure 3: Sequence of games for the proof

7 Expected strength (2.B.4) for each parameter set

7.1 Parameter set CFPKM128

The expected security level of this implemetation is 128 bits. From Section8, we find that the exhasutive search attack over the secret is the most efficiet of all the attacks discussed. With the set parameter, we find it takes 2^{129} number of operations, which is more than the level of security we are looking for.

7.2 Parameter set CFPKM182

The expected security level of this is 182 bits.

8 Analysis of known attacks (2.B.5)

In this part we provide the summary of the main attacks against CFPKM. In Section 8.1 we consider the Arora-Ge method of solving a system of noisy equations by removing the error and then using Grobner Basis techniques. In Section 8.2, we consider the possiblilty of a exhaustive search over the secret. We also consider the exhaustive search over the secret and then using Grobner Basis techniques to solve the resultant system of equations with Grobner Basis.

In order to break our protocol, the main motivation is to find the secret sa or sb used by either of Alice or Bob. We have a system of equations

$$b_{1_1} = f_1(\mathbf{x}) + e_{1_1}$$

$$b_{1_2} = f_2(\mathbf{x}) + e_{1_2}$$

$$\vdots$$

$$b_{1_m} = f_m(\mathbf{x}) + e_{1_m}$$

This is a system of non-linear multivariate polynomials, whose solution is $\mathbf{x} = sa$. The process of finding this solution is the exact hard problem of *PoSSo with Noise*. In the following sections, we discuss the possible methods of solving this problem and thus provide the hardness results, which relates to our problem.

Any solution that is provided by the solving using following methods, what is the guarantee that it is the solution. Or what property tells us that our system has uniqueness of solution. This narrows down to the effety

8.1 Usign Arora-Ge GB method

For solving our system of equations using Gröbner Basis solving techniques, we need the polynomials in the form which has no errors. This approach has been used in [1, 3]. In our case, we have $\mathbf{b1} = f(\mathbf{sa}) + \mathbf{e1}$ and $\mathbf{b2} = f(\mathbf{sb}) + \mathbf{e2}$. Here $f(x)$ is a quadritic polynomail. The errors are chosen from a discrete uniform distribution over a range $[0, s]$.

So suppose we represent our error as η, then $\eta = b - f(\mathbf{x})$. Here $\mathbf{x}$ is vector of variables indexed as $x_1, x_2, ...x_n$. We have m number of these equations. Let us represent

$$P(\eta) = \eta \prod_{j=1}^{s} (\eta - j)$$

Choosing the system of equations represented $\mathbf{b1} = f(\mathbf{sa}) + \mathbf{e1}$. So replacing $\eta = \mathbf{b1_i} - f_i(\mathbf{x})$, we get a system of equations represented as

$$P_i(\mathbf{b1_i} - f_i(\mathbf{x})) = (\mathbf{b1_i} - f_i(\mathbf{x})) \prod_{j=1}^{s} (\mathbf{b1_i} - f_i(\mathbf{x}) - j)$$

Hence we a system of m polynomials in n variables of degree $d = 2s + 2$, keeping in account of the degree of each f_i being 2 (i.e quadritic).
Now this polynomial $P_i = 0$ when $\mathbf{x} = \mathbf{sa}$. Also the secret is chosen uniformly from the range $[0, s)$. In addition to our system of polynomials, we have another set of n equations of the form

$$x_1(x_1 - 1) \cdots (x_1 - s) = 0$$

$$x_2(x_2-1)\cdots(x_2-s)=0$$
$$\vdots$$
$$x_n(x_n-1)\cdots(x_n-s)=0$$

So if we are able to find a Gröbner Basis of this system of equations along with system $P_i(\mathbf{x})=0$, then we will be able to recover **sa**.

Jean Charles Faugere's F5 algorithm computes the Gröbner Basis of a system of polynomials defined in a Polynomial Ring. The complexity of F5 algorithm [4] over a system of m' polynomials(forming a semi-regular sequence) in $\mathbb{Z}_q$ is given by

$$\mathcal{O}\Big(m'D_{reg}\binom{n+D_{reg}}{D_{reg}}^{\omega}\Big), as\ D_{reg}\rightarrow\infty$$

where $2\leq\omega<3$ is a linear algebra constant and D_{reg} is a degree of regularity of $<P_1,P_2,...P_{m'}>$.

Assumption: The system of equations $P_i(\mathbf{x})=0$, can be considered to be semi-regular sequence, since coefficients of $f_i(\mathbf{x})$ are (pseudo)random by choice.

So its Hilbert polynomail[8] is given by

$$H(z)=\frac{(1-z^{s+1})^n(1-z^d)^{m'}}{(1-z)^n} \tag{10}$$

where $d=2s+2$ and m' is the number of available equations and n is the number of variables. The Degree of regularity D_{reg} is given by the index of the first non-positive coefficient in the expansion of the Hilbert polynomial.

The number of samples that is assumed the attacker can have and running the F5 is $m'=n(1+\frac{1}{\log n})$.

8.2 Exhuastive Search

8.2.1 Over the secret

It is also possible that the attacker can do an exhasutive search over the possible values of the secret sa or sb. The secret is chosen from $[0,s]^n$. Since sa is a vector with dimension n, hence following the worst-case scenario, the attacker has to compute s^n possible solutions. So the number of operations is s^n. Since $s\approx n^{\beta}$, hence the number of operations in exhaustive search turns out to be $n^{\beta n}$.

n	D_{reg}	Arora-Ge-GB($\omega := 2.35$)	Arora-Ge-GB($\omega := 2$)
10	11	51	44
15	14	70	60
20	17	88	77
25	19	104	90
30	22	122	106
35	25	141	121
40	42	195	169
45	46	217	186
50	50	238	204
55	55	262	224
60	59	283	243
65	63	304	261
70	67	325	279
75	71	346	297
80	75	367	315
85	79	389	333

Table 2: Parameters with $s \approx n^{0.25}$

Note 1. *If we increase the value of our range s, then the time for exhaustive search increases. But this in turn also increases the degree of regularity and hence also the time complexity of the Arora-Ge GB attack.*

8.2.2 Over the error

With SODA approach

Another possible way of an attack is to do an exhaustive search over the possible values of the error, and then problem reduces to just solving a system of m non-linear equations in n variables. Using the SODA paper[9] allows us to solve the system of equations faster than brute force. So once the brute force (time complexity being s^m)over the error values has been solved. We have the set of equations with known errors , hence we now have a system of equations

$$\begin{gathered} f_1(\mathbf{x}) - \mathbf{b1}_1 - \mathbf{e}_1 = 0 \\ f_2(\mathbf{x}) - \mathbf{b1}_2 - \mathbf{e}_2 = 0 \\ \vdots \\ f_m(\mathbf{x}) - \mathbf{b1}_m - \mathbf{e}_m = 0 \end{gathered}$$

n	s	AG-GB	Ex-Sec	Ex-Err-SODA	Ex-Err-GB(D$_{reg}$)	HYB
30	2	106	30	74	94 (9)	191
35	2	121	35	86	107 (10)	224
40	3	168	63	120	157 (14)	251
45	3	186	71	134	174 (15)	276
50	3	204	79	149	190 (16)	307
55	3	224	87	164	206 (17)	338
60	3	243	95	178	222 (18)	364
65	3	261	103	193	238 (19)	394
70	3	279	111	208	258 (21)	420
75	3	297	119	223	274 (22)	451
80	3	315	**129**	237	290 (23)	481
85	3	333	135	252	306 (24)	507
90	3	351	142	267	322 (25)	538
95	3	370	151	281	338(26)	568
100	3	389	158	296	354(27)	594
105	3	404	166	311	374(29)	624
110	3	425	174	325	390(30)	649
115	3	442	**182**	340	406(31)	680
120	3	460	190	355	422(32)	711

Table 3: Comparing time complexity with $s \approx n^{0.25}$, all time complexity values in $\log_2$. The column AG-GB represents the Arora-Ge style Gröbner Basis attack, EX-Sec represents Exhaustive search over the Secret sa, EX-Err represents the Exhaustive search over the errrs and then using SODA and Gróbner Basis algorithms and finally HYB represents the hybrid approach from Section 8.3

Now [9] states the time complexity is

$$\mathcal{O}\left(q^n \cdot \left(\frac{\log q}{ek}\right)^{-n}\right)$$

for finding the satisfiablity of a system of equations where the solutions are in $\mathbb{F}_q$. So in total the time complexity of doing a exhaustive search of the error and then the SODA algo is therefore

$$s^m \cdot \mathcal{O}\left(s^n \cdot \left(\frac{\log s}{ek}\right)^{-n}\right)$$

With Gröbner Basis approach

Yet another alternative approach would be to find the Gröbner Basis of this system of equations. Finding Gröbner basis through Jean Charles Faugere's F5 [4], takes

$$\mathcal{O}\left(m \cdot D_{reg}\left(\binom{n+D_{reg}}{D_{reg}}^{\omega}\right)\right)$$

So the total complexity including the exhaustive search over the errors is

$$s^m \cdot \left(m \cdot D_{reg}\left(\binom{n+D_{reg}}{D_{reg}}^{\omega}\right)\right)$$

where D_{reg} is the degree of regularity over the system of m equations of degree $d = 2s + 2$ and n equations of degree $s + 1$ in n variables (Refer to Eq 10 for how to determine the D_{reg} in Section 8.1).

In Table 3 , we observe that Exhaustive search over the secret or the error is a faster and much efficient attack on our scheme. This is because our search space for both secret as well as the error is much smaller than the finite field related to the polynomial ring in which the polynomials have been defined.

8.3 Hybrid attack

Another possible way is to use an hybrid approach.In the previous section, we do a Grober basis attack on the whole system. With the system of equations that we have, suppose out of them only certain equations are non-noisy. So if we select only such non-noisy equations and then solve the Grobner basis just over these non noisy equations gives us a solution to the system of equations.

Suppose we are given a system of m equations in n variables. Now the errors can be achieved by doing an exhaustive search. So now assuming that the error distribution is uniform over the range $[0, s]$, we can say that $t = m/(s + 1)$ number of equations may be the exact solutions. That t equations are such that

$$f_1 = \cdots = f_t = 0$$

So now the Grobner Basis attack on this sub-system takes

$$\left(tD_{reg}\binom{n+D_{reg}}{D_{reg}}^{\omega}\right)$$

number of operations. So, the total complexity of this attack turns out to be

$$\binom{m}{t} \cdot \left(tD_{reg}\binom{n+D_{reg}}{D_{reg}}^{\omega}\right)$$

Though comparing the performance of this attack with exhaustive search, we see that for $n = 20$, we are gettting a security of 130 bits. But on the other hand, with $n = 30$ and the secret vector being chose from a small range s, the exhaustive search performs much better. It is also important to mention that in case of this attack, the minimum number of equations m will not suffice with $n + 1$. It has to be much larger than this, big enough such that $t > n$ for the Grobner basis attak to work. Hence we just report the performace of this attack in table 3.

9 Advantages and limitations (2.B.6)

CFPKM, is dependent on small secret and errors, which is one limitation of the proposed scheme. It has been left a future work for futher improving the performance of the scheme. But on the other hand, CFPKM has a lot of advantages.

The key encapsulation mechanism has been built in such a way that, it uses input from both the users to get a shared key, rather than the trivial way a Key encapsulation works. So this KEM can easily be modified to a key-exchange and agreement protocol as well. One of the major advantages that CFPKM has, is the cheap communication costs and key sizes. In comparision to similar KEM's based on Learning with Errors, this protocol is able to achieve similar levels of security with much lower values of comparable parameters. This is because of the use of a relatively newer hard problem of PoSSo with Noise. Solving a system of equations with noise is NP-Hard in itself.

References

[1] M. Albrecht, C. Cid, J.-C. Faugere, R. Fitzpatrick, and L. Perret. Algebraic algorithms for lwe problems. 2014.

[2] M. R. Albrecht, J.-C. Faugère, P. Farshim, G. Herold, and L. Perret. Polly cracker, revisited. Cryptology ePrint Archive, Report 2011/289, 2011. https://eprint.iacr.org/2011/289.

[3] S. Arora and R. Ge. New algorithms for learning in presence of errors. *Automata, languages and programming*, pages 403–415, 2011.

[4] M. Bardet, J.-C. Faugere, and B. Salvy. On the complexity of gröbner basis computation of semi-regular overdetermined algebraic equations. In *Proceedings of the International Conference on Polynomial System Solving*, pages 71–74, 2004.

[5] O. Billet and J. Ding. Overview of cryptanalysis techniques in multivariate public key cryptography. *Gröbner bases, coding and cryptography*, pages 263–283, 2009.

[6] J. Bos, C. Costello, L. Ducas, I. Mironov, M. Naehrig, V. Nikolaenko, A. Raghunathan, and D. Stebila. Frodo: Take off the ring! practical, quantum-secure key exchange from lwe. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1006–1018. ACM, 2016.

[7] Z. Brakerski and V. Vaikuntanathan. Fully homomorphic encryption from ring-lwe and security for key dependent messages. In *Annual cryptology conference*, pages 505–524. Springer, 2011.

[8] D. Cox, J. Little, and D. O'shea. *Ideals, varieties, and algorithms*, volume 3. Springer, 1992.

[9] D. Lokshtanov, R. Paturi, S. Tamaki, R. Williams, and H. Yu. Beating brute force for systems of polynomial equations over finite fields. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2190–2202. SIAM, 2017.

[10] C. Peikert. Lattice cryptography for the internet. In *International Workshop on Post-Quantum Cryptography*, pages 197–219. Springer, 2014.

[11] O. Regev. On lattices, learning with errors, random linear codes, and cryptography. *Journal of the ACM (JACM)*, 56(6):34, 2009.

[12] C. Wolf. Multivariate quadratic polynomials in public key cryptography. *IACR Cryptology ePrint Archive*, 2005:393, 2005.