EXCALIBUR KEY-GENERATION PROTOCOLS FOR DAG HIERARCHIC DECRIPTION

GERALDINE MONSALVE SANTANDER

Thesis submitted to the Office of Research and Graduate Studies in partial fulfillment of the requirements for the degree of Master of Science in Engineering

Advisor:
JUAN REUTTER DE LA MAZA

Santiago de Chile, Enero 2019

© 2019, GERALDINE MONSALVE SANTANDER
EXCALIBUR KEY-GENERATION
PROTOCOLS FOR DAG HIERARCHIC
DECRIPTION

GERALDINE MONSALVE SANTANDER

Members of the Committee:
JUAN REUTTER DE LA MAZA
DOMAGOJ VRGOC
FERNANDO KRELL
HÉCTOR JORQUERA

Thesis submitted to the Office of Research and Graduate Studies
in partial fulfillment of the requirements for the degree of
Master of Science in Engineering

Santiago de Chile, Enero 2019

© 2019, GERALDINE MONSALVE SANTANDER
Gratefully to my mother
ACKNOWLEDGEMENTS

I want to start by thanking my parents for their care, love and dedication. Specially to Julia, for spending so many years of her life in our growth. I also want to thank Anggie, for being my first teacher.

I would like to thank Juan Reutter for his willingness and understanding. He gave me the freedom to explore a new area for both of us and also allowed me to work in other valuable activities. I also want to thank Francisco Vial for bringing Cryptography to Departamento de Ciencia de la Computación and for always being willing to help.

I want to thank Héctor Jorquera, Domagoj Vrož and Fernando Krell for accepting to be part of the committee. Specially Fernando, for his dedication in my tesis.

Finally, I would also like to thank the DCC, specially the members of the ACM Student Chapter for making my time at university happy and Miguel for his company and love.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACKNOWLEDGEMENTS</td>
<td>iv</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>vii</td>
</tr>
<tr>
<td>LIST OF TABLES</td>
<td>viii</td>
</tr>
<tr>
<td>ABSTRACT</td>
<td>ix</td>
</tr>
<tr>
<td>RESUMEN</td>
<td>x</td>
</tr>
<tr>
<td>1. INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>1.1. Our contributions</td>
<td>2</td>
</tr>
<tr>
<td>1.2. Overview of the article</td>
<td>3</td>
</tr>
<tr>
<td>2. PRELIMINARIES</td>
<td>4</td>
</tr>
<tr>
<td>2.1. Indistinguishability</td>
<td>4</td>
</tr>
<tr>
<td>2.2. Quotient ring $R_q$</td>
<td>5</td>
</tr>
<tr>
<td>2.3. Invertible bounded Gaussian distributions in the quotient ring</td>
<td>6</td>
</tr>
<tr>
<td>2.4. FHE-NTRU encryption and the multikey property</td>
<td>6</td>
</tr>
<tr>
<td>3. SECURITY DEFINITIONS</td>
<td>8</td>
</tr>
<tr>
<td>3.1. The model of computation</td>
<td>8</td>
</tr>
<tr>
<td>3.1.1. Interactive Turing Machines</td>
<td>9</td>
</tr>
<tr>
<td>3.2. Simulation-based MPC security against semi-honest adversaries</td>
<td>10</td>
</tr>
<tr>
<td>4. HARDNESS ASSUMPTIONS</td>
<td>13</td>
</tr>
<tr>
<td>4.1. Decisional Small Polynomial Ratio Assumption</td>
<td>13</td>
</tr>
<tr>
<td>4.2. Small factorizations in $R_q$</td>
<td>14</td>
</tr>
<tr>
<td>5. MPC KEY GENERATION PROTOCOLS</td>
<td>15</td>
</tr>
<tr>
<td>5.1. Oblivious Transfer protocol</td>
<td>15</td>
</tr>
<tr>
<td>5.2. A Scalar Product protocol</td>
<td>15</td>
</tr>
</tbody>
</table>
5.3. Secure MPC protocols for multiplication in $R_q$ .......................... 17
5.4. Excalibur key generation protocols .......................................... 18

6. SECURITY ANALYSIS ..................................................................... 25
6.1. Extracting keys after the protocol .............................................. 25
6.2. Extracting secrets during the protocols ....................................... 27
   6.2.1. Proof of proposition 6.2 ..................................................... 31
   6.2.2. Proof of proposition 6.3 ..................................................... 36
   6.2.3. Proof of proposition 6.4 ..................................................... 37
   6.2.4. Proof of proposition 6.5 ..................................................... 39

7. PARAMETERS AND COMPLEXITY .............................................. 41
7.1. Algorithmic Complexity ......................................................... 42

8. CONCLUSIONS ........................................................................... 44
8.1. Future work ............................................................................ 44

REFERENCES .................................................................................. 46
LIST OF FIGURES

5.1 Exc\textsubscript{pk}: $P_1$ learn $r'_1$ ................................. 21
5.2 Exc\textsubscript{pk}: $P_i$ learn $u_i$ ................................. 21
5.3 Exc\textsubscript{pk}: Each $P_i$ learn $w_i$ ................................. 22
5.4 Exc\textsubscript{pk}: Each $P_i$ learn $z_i$ ................................. 22
5.5 Exc\textsubscript{sk}: Each $P_i$ learns the product of $\beta_i$'s plus some noise ................................. 23
5.6 Exc\textsubscript{sk}: $P_1$ learns his secret key ................................. 24
LIST OF TABLES
ABSTRACT

Public-key cryptography applications often require structuring decryption rights according to some hierarchy. This is typically addressed with re-encryption procedures or relying on trusted parties, in order to avoid secret-key transfers and leakages. In a novel approach, Goubin and Vial-Prado take advantage of the Multikey FHE-NTRU encryption scheme to propose, in 2016, *Excalibur* procedures that define decryption rights at key-generation time, and preventing leakage of all secrets involved, even by the powerful-key holder. Their algorithms work for two parties, allowing to form chains of users with inherited decryption rights. In this thesis we provide new protocols for generating *Excalibur* keys under any DAG-like hierarchy, that extend Goubin and Vial-Prado’s previous work, and present formal proofs of security against semi-honest adversaries. Our protocols are compatible with the homomorphic properties of FHE-NTRU, and the base case of our security proofs may be regarded as a more formal, simulation-based proof of said work.

**Keywords:** Secure Multiparty Computation, FHE-NTRU scheme.
RESUMEN

Aplicaciones criptográficas de llave pública usualmente requieren estructurar privilegios de desencriptación acorde a alguna jerarquía. Para evitar filtraciones y transferencias de las llaves secretas, usualmente esto se resuelve mediante procesos de re-recriptación o recurriendo a entidades confiables. En un enfoque inicial Goubin y Vial-Prado aprovechan el esquema de encriptación multillave FHE-NTRU para proponer los protocolos Excalibur. Estos protocolos definen privilegios de desencriptación al momento de la creación de llaves, evitando filtraciones de cada uno de los secretos involucrados, incluso por parte de quien posee la llave poderosa, i.e. los privilegios de desencriptación. Los algoritmos se definen para escenarios con dos participantes, y se extienden para cadenas de participantes con privilegios de desencriptación heredables. En esta tesis proponemos nuevos protocolos para la generación de llaves Excalibur, en un escenario de jerarquía DAG, extendiendo así el trabajo previo de Goubin y Vial-Prado. Además, se presentan demostraciones formales de seguridad en presencia de adversarios semi-honestos, donde el caso base de nuestras demostraciones de seguridad pueden considerarse como una prueba basada en simulación más formal del trabajo anteriormente mencionado. Finalmente, nuestros protocolos son compatibles con las propiedades homomórficas del esquema FHE-NTRU.

Palabras Claves: Computación de múltiples partes segura, esquema FHE-NTRU.
1. INTRODUCTION

In some public-key cryptography applications, parties own decryption rights over ciphertexts according to some hierarchic structure. For example, Bob is an employee of Alice and not be available for some months. Bob configured the forwarding of all his mails to Alice for this time, and then Alice need to decrypt Bob’s messages. If Bob simply transfers his secret key to Alice, she may leak or sell Bob’s secret, causing a lot more damage than leaking Bob’s plaintexts only. Overcoming this, proxy re-encryption and hierarchical identity-based encryption schemes rely on trusted parties to generate master secret keys or involve public re-encryption procedures.

Using a novel approach, authors in (Goubin & Vial Prado, 2016) proposed two-party computation protocols that securely perform a key generation procedure of the celebrated NTRU-based Multikey-FHE (López-Alt, Tromer, & Vaikuntanathan, 2012). The result is a key pair \((sk_A, pk_A)\) for Alice such that \(sk_A\) can decrypt all of Bob’s ciphertexts, and no information about Bob’s secret key can be deduced from this key pair, the execution of the protocol or any public values. Moreover, Alice’s and Bob’s secrets are tied together, thus effectively avoiding leakage by Alice (assuming she is not willing to reveal her own secret key \(sk_A\)). The newly-created Excalibur key pair behaves as a regular key of the system, even allowing multikey homomorphic operations when using sufficiently large parameters such as the ones suggested in (López-Alt et al., 2012). In addition, these keys can be used as inputs to generate more powerful Excalibur keys, allowing decryption inheritance for a bounded chain of users. In addition, this whole procedure can be regarded as an automatic \(N\)-hop proxy-reencryption scheme, addressing a re-encryption paradigm in fully homomorphic encryption scenarios, as pointed out in (Goubin & Vial Prado, 2016).

In this tesis we extend these key-generation protocols and provide multiparty computation protocols in the advised cyclotomic polynomial ring of (Stehlé & Steinfeld, 2011) that can securely generate FHE-NTRU keys for some types of DAG-like hierarchies, in such a way that the key of a particular node \(n\) in this hierarchy can decrypt all messages
encrypted for nodes below it, while not having access to secrets (other than own private keys). In order to do this, we address the case where Bob above is replaced by a set of participants.

As typical Multiparty Computation applications, our protocols require the composition of several routines. In all generality, this poses additional security restrictions, as a composition of two secure protocols in a stand-alone setting is not necessarily secure. This has been the subject of extensive research, and we highlight the Universal Composability (UC) framework proposed by Canetti (Canetti, 2001) in which any two UC-secure protocols may be arbitrarily composed ensuring the inheritance of security.

In our protocols only non-concurrent composition is performed and security is proven with simulation-based proofs, but only in the stand-alone setting. This means that when our protocol runs in isolation (with all necessaries subroutines) it is secure, but if runs alongside other protocols it could not be. Prove that our protocols are secure for UC framework remains as future work.

1.1. Our contributions

This work provides new protocols that extend those from (Goubin & Vial Prado, 2016), addressing the case where $k > 2$ parties jointly generate an NTRU-FHE key pair with additional decryption rights, and preventing leakage of secret keys from the receiving party. To this end, we propose secure multi-party computation protocols in cyclotomic polynomial rings between parties $P_1, \ldots, P_k$ that generate a key-pair for party $P_1$ with decryption rights over all parties involved, and such that no information about other secret-keys can be leaked from the execution of the protocol, inputs and outputs, and public values, even if some parties collude.
We provide security analysis of these protocols in the semi-honest but colluded setting, where parties follow the protocol and sample from the correct distributions but may cooperate with each other to deduce secrets. The base case of our security analysis may be regarded as a more formal, simulation-based proof of the protocols in (Goubin & Vial Prado, 2016). Achieving security in malicious adversarial settings, where parties may deviate or sabotage the protocol, is a challenging problem for \( k > 2 \) parties, which we leave for future work.

In order to generate an Excalibur key pair that inherits decryption of 3 other keys with 128 bits of security in mind, parties using our protocols need to perform around \( 2^{24} \) 1-out-of-2 Oblivious Transfer protocols and \( 2^{24} \) multiplications in \( \mathbb{Z}_q[x]/(x^n + 1) \) with secure NTRU parameters, this can be performed in range of minutes on a regular laptop, as per our simulations and (Asharov, Lindell, Schneider, & Zohner, 2013). We are confident that there is much to optimize in these procedures, opening another interesting angle for future work.

1.2. Overview of the article

We begin in section 2 by revisiting necessary concepts to construct our protocols, including the quotient ring \( R_q \) and FHE-NTRU scheme. In sections 3 and 4, we define the notions of security we want to achieve, and state the corresponding underlying assumptions. Our protocols are presented in 5, with proofs and security analysis described in 6.
2. PRELIMINARIES

We denote tuples of elements with bold letters. The indicator function or caracteristic function $X_B : A \mapsto \{0, 1\}$ indicates membership of an element $x \in A$ in a subset $B \subseteq A$. This function outputs 1 if the preimage is in $B$ and 0 otherwise:

$$X_B(x) = \begin{cases} 1 & \text{if } x \in A \\ 0 & \text{if } x \notin A \end{cases}$$

2.1. Indistinguishability

For a probability distribution $X$ with sample space $S$, $e \leftarrow X$ denote that $e$ is sampled from $S$ with the distribution $X$, and $e \leftarrow S$ denote that $e$ is sampled from $S$ according to the uniform distribution.

A distribution ensemble $X = \{X(\lambda, a)\}_{\lambda \in \mathbb{N}, a \in D}$ is an infinite sequence of probability distributions, where a distribution $X(\lambda, a)$ is associated with each values of $\lambda \in \mathbb{N}$ and $a \in D$ for some domain $D$. The distribution ensembles considered in this work are outputs of computations (protocols or simulators algorithms), where $a$ corresponds to various types of inputs and $\lambda$ is taken to be the security parameter. As usual, the complexity of our protocols is measured in terms of the security parameter.

Let $\mu$ a negligible function. Two distribution ensembles $X$ and $Y$ are statistically indistinguishables, writed as $X \approx^s Y$, if for all sufficiently large $\lambda$ and all $a$ we have that

$$\frac{1}{2} \sum_b | \Pr[X(\lambda, a) = b] - \Pr[Y(\lambda, a) = b] | < \mu(\lambda)$$

Two distribution ensembles $X$ and $Y$ are computational indistinguishables, writed as $X \approx^c Y$, if for every algorithm $D$ that is probabilistic polynomial-time in it first input, for all sufficiently large $\lambda$ and all $a$ and all auxiliary information $w \in \{0, 1\}^*$ we have

$$| \Pr[D(\lambda, a, w, x) = 1] - \Pr[D(\lambda, a, w, y) = 1] | < \mu(\lambda)$$
where $x \leftarrow X(k, a)$ and $y \leftarrow Y(k, a)$.

### 2.2. Quotient ring $R_q$

Let $q$ be a large prime. For an integer $x \in \mathbb{Z}$, $[x \mod q]$ represents the modular reduction of $x$ into the set $\{-\lfloor q/2 \rfloor, \ldots, \lfloor q/2 \rfloor\}$, which we denote by $\mathbb{Z}_q$.

For a polynomial $p(x) = a_0 + a_1 x + \cdots + a_{n-1} x^{n-1} \in R_q$, let $\mathbf{p} = (a_0, \ldots, a_{n-1}) \in \mathbb{Z}_q^n$ be the coefficient vector of $p$. We denote by $||\mathbf{p}||_\infty$ the infinity norm of $p$, defined as $||\mathbf{p}||_\infty = \max_k |a_k|$.

A $n$-th root of unity is a complex number $\zeta$ such that $\zeta^n = 1$. There are exactly $n$ different $n$-th roots of unity: $e^{2\pi i/n} = \cos \left(\frac{2k\pi}{n}\right) + i \sin \left(\frac{2k\pi}{n}\right)$, for $0 \leq k < n$ and $n$ a positive integer. A primitive $n$-th root of unity is an $n$-th root of unity $\zeta$ whose order is $n$.

The $n$-th cyclotomic polynomial $\Phi_n$ is defined by $\Phi_n(x) = \prod_{\zeta \in P(n)} (x - \zeta)$, where $P(n)$ denotes the set of all primitive $n$-th roots of unity. The $2^k$-th cyclotomic polynomial $\Phi_{2^k}$ is equal to $x^{2^{k-1}} + 1$, for $k$ an integer.

For $n$ a power of two, let $R_q \overset{\text{def}}{=} \mathbb{Z}_q[x] / \Phi_{2^n}(x)$ be the cyclotomic ring of polynomials modulo $\Phi(2^n) = x^{2^{n-1}} + 1$ and coefficients in $\mathbb{Z}_q$. Note that the ring $R_q$ is not a unique factorization domain as $x^n + 1$ is generally not irreducible in $\mathbb{Z}_q$. 

5
2.3. Invertible bounded Gaussian distributions in the quotient ring

In the following definition, let $\Gamma_r$ be the Gaussian distribution on $\mathbb{Z}_q^n$ centered about 0 and standard-deviation $r$, for a real number $r > 0$.

**Definition 2.1** (Bounded Discrete Gaussian distribution over $R_q$). For a real number $B$ such that $0 < B \ll q$, let $\mathcal{G}_B$ be the $B$-bounded discrete Gaussian distribution over $R_q$, that is, the distribution that samples polynomials from $R_q$ as follows:

(i) Sample vector $x \leftarrow \Gamma_B$, and restart if $||x||_\infty > B$.

(ii) Output the polynomial $p \in R_q$ whose coefficients vector is $\lfloor x \rfloor$.

Let $\mathcal{G}_B^\times$ be the distribution that samples from $\mathcal{G}_B$ until the output is invertible.

2.4. FHE-NTRU encryption and the multikey property

The public key cryptosystem NTRU is the fastest known lattice-based encryption scheme. In it, units of $R_q$ with small coefficients are used as secret keys. In fact, for $n$ a power of 2 and $q = 1 \mod 2n$, the $2n$-th cyclotomic polynomial $X^n + 1$ splits into $n$ linear factors in $\mathbb{Z}_q$ ensuring a large key-space, as in the advised modifications by Stehlé and Steinfeld (Stehlé & Steinfeld, 2011).

The Multikey FHE scheme presented in (López-Alt et al., 2012) uses the modified version of NTRU (N$^h$-truncated) encryption scheme, which we present here for the sake of completeness.

**Parameters:** Let $n$ be a power of 2, $q$ be a large prime such that $q = 1 \mod 2n$ and $0 < B \ll q$. Recall that $R_q = \mathbb{Z}_q[x]/(x^n + 1)$.

**Key Generation:** Sample a polynomial $f \leftarrow \mathcal{G}_B$ and set $\text{sk} := 2f + 1$, repeat if $\text{sk}$ is not invertible. Sample $g \leftarrow \mathcal{G}_B^\times$ and define $\text{pk} := 2g \cdot \text{sk}^{-1}$. Output the pair ($\text{pk}, \text{sk}$).

**Encryption:** Given a message $m \in \{0, 1\}$ and public-key $\text{pk}$, sample $s, e \leftarrow \mathcal{G}_B$, and output $c = m + 2e + s \cdot \text{pk} \mod q$. 

6
**Decryption:** Given a ciphertext $c$ and secret-key $sk$, output $m = c \cdot sk \mod 2$.

The linearity of the decryption equation allowed authors of (López-Alt et al., 2012) to construct the first Multikey FHE scheme, where the result of homomorphic operations involving ciphertexts related to different entities can be jointly decrypted by these parties. This is showed in lemma 2.1.

As noted by (Goubin & Vial Prado, 2016) and showed in corollary 2.1, this linearity also implies that a secret key with small extra multiplicative factors (such as other secret keys) is able to correctly decrypt.

**Lemma 2.1.** (Goubin & Vial Prado, 2016) Let $(pk, sk) \leftarrow \text{Keygen}(1^k)$, $m \in \{0, 1\}$ and let $c \leftarrow \text{Enc}(pk, m)$. Let $\theta \in R_q$ be $M$-bounded polynomial satisfying $\theta \mod 2 = 1$. If $M < (\frac{1}{q})(\frac{q}{n^K})$ then

$$\text{Dec}(sk, c) = \text{Dec}(\theta \cdot sk, c) = m.$$  

**Corollary 2.1.** (The multikey property) Let $(pk_1, sk_1)$ and $(pk_2, sk_2)$ be valid pairs of keys, $m_1, m_2 \in \{0, 1\}$ and let $c_1 \leftarrow \text{Enc}(pk_1, m_1)$, $c_2 \leftarrow \text{Enc}(pk_2, m_2)$. Let $\tilde{sk} = sk_1 \cdot sk_2 \in R_q$. Then

$$\text{Dec}(\tilde{sk}, c_1) = m_1, \text{Dec}(\tilde{sk}, c_2) = m_2$$

provided that $K$ is small enough.
3. SECURITY DEFINITIONS

A cryptographic protocol includes a set of parties that need to compute jointly an algorithm, guaranteeing some secure properties in the face of an adversary. A simple example is the communication between parties, where an external adversary has access to the communication network. For example, if the adversary only has access to see the messages, the secrecy is the property that we wish to guarantee. But if the adversarial behaviour is active, namely can modify messages in the communication network, we also need to guarantee the integrity of messages.

These protocols involve different and complex types of trust relationships and different types of security properties. A first distinction is that adversarial behaviour can be semi-honest or malicious. Semi-honest adversaries follow the protocol specification exactly, but they may cooperate with each other to deduce information. In contrast, malicious adversaries may arbitrarily deviate from the protocol specification.

Another distinction is between static and adaptive adversarial behaviour. An static adversary controls a fixed set of corrupted parties, while an adaptive adversary chooses which parties to corrupt during the computation, based on the information obtained so far.

In this chapter, we present the model of computation and definitions of security used in this work. We only consider static semi-honest adversaries, leaving malicious and adaptive case for future work. Additionally, we assume honest majority, i.e. the case where the set of honest parties is bigger than the set of corrupted parties.

3.1. The model of computation

With the goal of formalizing the notion of security in cryptographic protocols, we present the model to represent and analyze them.
3.1.1. Interactive Turing Machines

An Interactive Turing Machine (ITM) is a extension of the standard deterministic multi-tape Turing Machine, that allows communication between pairs of machines. In (Goldreich et al., 2005) authors give a first definition of ITM in the context of interactive proof systems, in (Canetti, 2001) defined a formalization of ITM accommodated to multyparty computation protocols.

**Definition 3.1.** An Interactive Turing Machine (ITM) $M$ is deterministic multi-tape Turing Machine. The tapes in the machine are:

- A read-only identity tape.
- A read-only input tape.
- A read-only random tape.
- A read-and-write work tape.
- A pair of communication tapes: a read-only tape representing the information coming from other machines within the same protocol instance; and a write-only tape holding the outgoing messages together with the addressing information for delivery the messages.
- A read-and-write one-bit activation tape.

An Interactive Turing Machine represent the static object: the idea of algorithm, while an Interactive Turing Machine Instance (ITI) represent the run-time object. A cryptographic protocol is just defined as a single ITM and if the protocol specify different algorithms for different parties, then the ITM must be to describe this algorithms.

Our model consider a set of ITIs that can communicate with each other, they can write on the communication tape of other ITI. The execution of this system of ITIs is just a sequence of activations of the ITIs in the system. In each activation, the only activated ITI follow its own transition function based on its current state and contents of tapes until it enters in a special idle state. At the end of each activation, the ITI whose tape was written
is the next ITI that will be activated. There is a special and predeterminated ITI which is the first to activate in the execution, and this execution ends when the initial ITI halts.

3.2. Simulation-based MPC security against semi-honest adversaries

The idea behind the notion of security is to guarantee that performing a cryptographic protocol is just as good as executing an idealized computational process where security is assured. According to this, following previous attempts of formalization, e.g., (Lindell, 2017) and (Canetti, 2000), our notion of security is based on the idea of simulating functionalities. This refers to compare what happens in the real world versus what happens in and ideal world.

Let $P = \{P_1, \ldots, P_k\}$ a fixed set of parties and $\lambda$ the security parameter. A functionality is a $(k + 1)$-ary function

$$f : \mathbb{N} \times (\{0, 1\}^*)^k \rightarrow (\{0, 1\}^*)^k.$$ 

We usually write $f = (f_1, \ldots, f_k)$, where each $f_i$ is a random $k + 1$-ary function that outputs a string, i.e., $f_i : \mathbb{N} \times (\{0, 1\}^*)^k \rightarrow \{0, 1\}^*$. We only are interesting in functionalities that are computable in polynomial time with respect to the security parameter $\lambda$, this means there is a deterministic turing machine with input $x$ can output $f(\lambda, x)$ in polynomial time with respect the security parameter $\lambda$.

A protocol $\pi$ compute a functionality $f$ if with input $x_i$ for each $P_i$, the protocol gives to $P_i$ the output $f_i(\lambda, x_1, \ldots, x_k)$. As usual, the idea is to show that a protocol computing a functionality $f$ is secure if all possible information that can be computed by a collusion of some parties can be simulated by means of the combined input and output of these parties when executing the protocol.

Let $f$ be a functionality, and $\pi$ a protocol for computing $f$. The view of the $i$-th party when executing $\pi$ on input $x = \{x_1, \ldots, x_k\}$ and security parameter $\lambda$, denoted as
view_\pi^i(\lambda, x) \), is a tuple

\((x_i, r_i, m_i^1, \ldots, m_i^j)\),

where \(r_i^j\) is the content of the internal random tape of the \(i\)-th party, and \(m_i^1, \ldots, m_i^j\) represents the messages sent and received with other parties during the execution of \(\pi\). For a set \(S \subseteq P\) of parties, we set \(\text{view}\pi^S(\lambda, x)\) as the concatenation of each tuple \(\text{view}\pi^i(\lambda, x)\).

We also write \(f_S\) as the tuple formed of each \(f_i\), for \(i \in S\), and \(x_S\) as the tuple formed of each \(x_i, i \in S\).

The output of the \(i\)-th party when executing \(\pi\) on input \(x = \{x_1, \ldots, x_k\}\) and security parameter \(\lambda\) is denoted as \(\text{output}\pi(\lambda, x)\).

**Definition 3.2.** Let \(P\) be a set of \(k\) parties, and \(f = (f_1, \ldots, f_k)\) a functionality. We say that \(\pi\) securely computes \(f\) in the presence of semi-honest adversaries if for every set \(S \subseteq P\) of colluded parties there is a PPT algorithm \(I_S\) such that

\((I_S(\lambda, x_S, f_S(\lambda, x)), f(\lambda, x)) \approx (\text{view}\pi^S(\lambda, x), \text{output}\pi(\lambda, x))\)

We now give a proposition that allows us to prove security for protocols that involve executing other protocols as non-concurrent sub-routines. Let \(\pi_1, \ldots, \pi_\ell\) be protocols computing functionalities \(\phi_1, \ldots, \phi_\ell\), respectively. Let \(\rho^{\pi_1, \ldots, \pi_\ell}\) be a protocol computing a functionality \(g\) that makes use of \(\pi_1, \ldots, \pi_\ell\) as subroutine in a non-concurrent fashion, so that \(\pi_i\) is called only after \(\pi_{i-1}\) returns, and additionally, \(\rho^{\pi_1, \ldots, \pi_\ell}\) pauses when executing each \(\pi_i\). Denote by \(\rho^{\pi_1 \rightarrow \phi_1, \ldots, \pi_\ell \rightarrow \phi_\ell}\) the protocol where instead of calling to each \(\pi_i\), we use an oracle computing the functionality \(f_i\).

**Proposition 3.1.** If every \(\pi_i\) securely computes \(\phi_i\), and \(\rho^{\pi_1 \rightarrow \phi_1, \ldots, \pi_\ell \rightarrow \phi_\ell}\) securely computes \(g\), then \(\rho^{\pi_1, \ldots, \pi_\ell}\) securely computes \(g\). (Lindell, 2017)

Note that the restriction that sub-protocols are invoked non-concurrently is key for stating this result in such a simplified way, instead of using the more complete, but more complex framework presented in (Canetti, 2001). We do highlight that the restriction of
Canetti’s framework into our scenario yields security requirements equivalent to that of Definition 3.2 (see (Canetti, 2006), §5.2).
4. HARDNESS ASSUMPTIONS

Security of our protocols against semi-honest adversaries is based on two well-known assumptions: The Ring Learning With Errors problem (RLWE) and The Decisional Small Polynomial Ratio problem (DSPR); and the difficulty of new factorization problems in $R_q$ that extend those from (Goubin & Vial Prado, 2016). We first describe the DSPR assumption.

4.1. Decisional Small Polynimial Ratio Assumption

The Ring Learning With Errors problem and the Decisional Small Problem Ratio, detailed in definitions 4.1 and 4.2, relate to the security of the underlying NTRU encryption scheme as described in (Stehlé & Steinfeld, 2011; López-Alt et al., 2012).

Let $s \in R_q$, $\psi$ a distribution in $R_q$ and $A_{s,\psi}$ the distribution obtained by sampling the pair $(a, as + e)$ with $(a, e) \leftarrow U(R_q) \times \psi$. The RLWE problem shows that is difficult to distinguish between a sampling from $U(R_q \times R_q)$ and from $A_{s,\psi}$.

**Definition 4.1** (Ring-LWE, from (Stehlé & Steinfeld, 2011)). The Ring Learning with Errors Problem is as follows. Let $\psi \leftarrow \Upsilon$ and $s \leftarrow U(R_q)$. Given access to an oracle $O$ that produces samples in $R_q \times R_q$, distinguish whether $O$ outputs samples from $A_{s,\psi}$ or from $U(R_q \times R_q)$. The distinguishing advantage should be negligible over the randomness of the input, the randomness of the samples and the internal randomness of the algorithm.

The DSPR showed that the public-key distribution is statistically close to uniform in the ring.

**Definition 4.2** (Decisional Small Polynomial Ratio assumption, from (Stehlé & Steinfeld, 2011)). For some parameters $q, n, B$, it is computationally hard to distinguish between the following two distributions over $R_q$:

(i) A polynomial $pk = 2g(2f + 1)^{-1} \in R_q$ where $f, g \leftarrow G_B$, and
(ii) a uniformly random polynomial $u \overset{\$}{\leftarrow} R_q$.

4.2. Small factorizations in $R_q$

Additionally, the semi-honest security rely on the hardness of the following problems. Let $\xi_{lB}$ be the Gaussian Product Distribution defined as follows.

**Definition 4.3** (Gaussian Product Distribution). Let $\xi_{lB}$ be the distribution that samples polynomials $p_i \leftarrow G_B^x$ for $i = 1, \ldots, l$ and outputs $\prod_{i=1}^{l} p_i$.

**Definition 4.4** (Special factors problem). Let $\alpha \leftarrow \xi_{lB}$ and $\beta \leftarrow \xi_{mB}$. The Special Factors Problem is to output $\alpha, \beta$ with the knowledge of $c = \alpha \cdot \beta$ and access to the indicator function of $\{\alpha, \beta\}$.

In other words, the task is to find the correct factorization of $c$. Recall that $R_q$ is not a unique factorization domain, so for any unit $u \in R_q$ there is a possible factorization $c = u \cdot (u^{-1} c)$. In order to find $\alpha, \beta$, the solver must query the indicator function to corroborate that his solution is the correct one. In our construction, secret keys play the role of the individual factors of $c$, and the indicator function consists in encrypt-decrypt key-guessing routines. When $l = m = 1$, this is the small factors problem from (Goubin & Vial Prado, 2016).

**Definition 4.5** (Special GCD problem). Let $\alpha, \beta \leftarrow G_B^\times$ and $y \leftarrow \xi_{lB}$. Given $u = \alpha \cdot y$ and $v = \alpha \cdot \beta$ and access to the indicator function of $\{\alpha, \beta, y\}$, output $\alpha, \beta$ and $y$.

**Definition 4.6** (Special Factors Assumption). For some set of parameters, it is computationally hard to solve the Special Factors or the Special GCD problems.

As noted in (Goubin & Vial Prado, 2016), Special GCD reduces to a version of DSPR, and the SF problem may be expressed as a quadratic system of equations in $\mathbb{Z}_q$ in the underdetermined setting, which is considered secure (Thomae & Wolf, 2012). Moreover, as in (Goubin & Vial Prado, 2016) we put it as a conjecture that the additional cyclic structure provided by $R_q$ does not help an attacker to solve this system.
5. MPC KEY GENERATION PROTOCOLS

This chapter introduce the key-generator multiparty protocols $\text{Exc}_{pk}$ and $\text{Exc}_{sk}$, with the aim of create a key pair $(\text{sk}_1, \text{pk}_1)$ for participant $P_1$, based on the set $(\text{sk}_i, \text{pk}_i)$ of all other participants. As we have mentioned, the pair $(\text{sk}_1, \text{pk}_1)$ can decrypt any message encrypted with the public key of any other participant and $P_1$ cannot discover any of the other secret keys.

For this purpose, we define first the Oblivious Transfer functionality and a Scalar Product functionality.

5.1. Oblivious Transfer protocol

The one-out-of-n oblivious transfer protocol is a distributed two-party algorithm in which a party $P_1$ has $n$ secret values and transfer one of them to $P_2$, without know what element has been sended.

The oblivious transfer functionality is defined as

$$\mathcal{F}^{(n)}_{\text{OT}} : (\lambda, (x_1, \ldots, x_n), m) \mapsto (\bot, x_m),$$

where $m$ is an integer between 1 and $n$. Let $(\binom{n}{1})_{\text{OT}}$ a protocol that securely computes $\mathcal{F}^{(n)}_{\text{OT}}$.

5.2. A Scalar Product protocol

We define the scalar product functionality as

$$\mathcal{F}^{SP_m} : (\vec{b}, (\vec{p}^{(0)}, \vec{p}^{(1)})) \mapsto (\sum_{i=1}^{m} p_i^{(b_i)}, \bot),$$

where $\vec{b} \in \{0, 1\}^m$ is a vector of bits defined as $\vec{b} = (b_0, \ldots, b_m)$, and $\vec{p}^{(0)}, \vec{p}^{(1)} \in R_q^m$ are vectors of polynomials in $R_q$ defined as $\vec{p}^{(0)} = (p_1^{(0)}, \ldots, p_m^{(0)})$ and $\vec{p}^{(1)} = (p_1^{(1)}, \ldots, p_m^{(1)})$.  

15
Note that
\[ \sum_{i=1}^{m} p_i^{(b_i)} = (p_0^1, \ldots, p_m^1) \cdot \vec{b}^c + (p_1^1, \ldots, p_m^1) \cdot \vec{b}, \]
where \( \vec{b}^c = (\bar{b}_1, \ldots, \bar{b}_m) \) is the binary complement of \( \vec{b} \).

The two-party protocol performing this functionality is presented in definition 5.1 and outlined in the algorithm 1.

**Definition 5.1.** Let \( \mathcal{SP}_m \) be a two-party protocol performing \( \mathcal{F}^{\mathcal{SP}_m} \) for \( m \in \mathbb{N} \), as follows. Party \( P_1 \) has a sequence of \( m \) bits ordered in a binary vector \( \vec{b} = (b_1, \ldots, b_m) \).

For each \( i = 1, \ldots, m \), party \( P_2 \) has a pair of polynomials \( (p_i^{(0)}, p_i^{(1)}) \) of \( R_q \). In the end, party \( P_1 \) only learns \( \gamma = p_1^{(b_1)} + p_2^{(b_2)} + \cdots + p_m^{(b_m)} \) and party \( P_2 \) learns nothing.

Note that when \( m = 1 \) this is simply a \((\binom{2}{1})\)-OT (1-out-of-2 oblivious transfer). The construction of this protocol is straightforward and based on (Li & Dai, 2005).

**Algorithm 1 Scalar product protocol**

**Require:** \( P_1 \) holds \( \vec{b} = (b_1, \ldots, b_m) \in \{0, 1\}^m \) and \( P_2 \) holds \( 2m \) polynomials \( (p_i^{(0)}, p_i^{(1)}) \in R_q^m \). Let \( \kappa \) be such that it is unfeasible to compute \( 2^\kappa \) additions in \( R_q \).

**Ensure:** \( P_1 \) learns \( (p_0^0, \ldots, p_m^0) \cdot \vec{b}^c + (p_1^1, \ldots, p_m^1) \cdot \vec{b} \) and \( P_2 \) learns nothing.

1: **procedure** \( \mathcal{SP}_m \)
2: \( P_1 \) samples \( \kappa \) vectors \( (\vec{b}_i \leftarrow \mathbb{Z}_2^m)_{i=1}^{\kappa} \) such that \( \sum_{i=1}^{\kappa} \vec{b}_i = \vec{b} \).
3: **for** \( i = 1 \ldots \kappa \) **do**
4: \( P_1 \) samples a bit \( \sigma \) and two vectors \( \vec{a}_0, \vec{a}_1 \leftarrow \{0, 1\}^m \). \( P_1 \) sets \( a_\sigma \leftarrow \vec{b}_i \).
5: \( P_1 \) sends the pair \( (\vec{a}_1, \vec{a}_2) \) to \( P_2 \).
6: \( P_2 \) computes
   \[ \vec{d}_0 = (p_0^0, \ldots, p_m^0) \cdot \vec{a}_0 + (p_1^1, \ldots, p_m^1) \cdot \vec{a}_0 \]
   \[ \vec{d}_1 = (p_0^1, \ldots, p_m^1) \cdot \vec{a}_1 + (p_1^1, \ldots, p_m^1) \cdot \vec{a}_1 \]
7: **With** a \((\binom{2}{1})\)-OT protocol, \( P_1 \) extracts \( \gamma_\sigma := \vec{d}_\sigma \) from \( P_2 \).
8: \( P_1 \) computes \( \gamma = \sum_{i=1}^{\kappa} \gamma_i = (p_0^0, \ldots, p_m^0) \cdot \vec{b}^c + (p_1^1, \ldots, p_m^1) \cdot \vec{b} \).

**Remark:** This protocol can be restated as a \((\binom{2m}{1})\)-OT protocol, as follows. For each \( x \in \{0, 1\}^m \), party \( P_2 \) computes a mapping \( x \mapsto \sum_{i=1}^{m} p_i^{x[i]} \) where \( x[i] \) is the \( i \)-th bit of \( x \).
Then, party $P_1$ extracts the polynomial corresponding to $x' = \bar{b}$ with a $(2^m)$-OT protocol. We point out that this is highly inefficient, because $P_2$ needs to compute $O(2^m)$ additions in $R_q$.

### 5.3. Secure MPC protocols for multiplication in $R_q$

The building blocks of our key-generating scheme are two protocols, that we name $k$-Multiplication Protocol and $k$-Shared Multiplication Protocol. Both of these protocols share the goal of performing a multiparty multiplication of elements in $R_q$, but differ in the final output learned by the participants.

**2-Multiplication Protocol.** This two-party protocol is defined in (Goubin & Vial Prado, 2016), and the goal is compute the product of two elements in $R_q$. The participant $P_1$ begins with $x_1$ as input, and $P_2$ with the pari $x_2, r_2$. At the end, $P_1$ learns $x_1 \cdot x_2 + r_2$, but $P_2$ learn nothing.

The detail of the protocol is present in Algorithm 2. The participant $P_1$ generate randomly $m$ additive parts $x_{1i}$ of $x_1$, anagously $P_2$ generate $m$ additive parts $r_{2i}$ of $r_2$. For each polynomial $x_{1i}$ they compute $x_{1i} \cdot x_2 + r_{2i}$, and at the end $P_1$ can compute $\sum x_{1i} \cdot x_2 + r_2 = x_1 \cdot x_2 + r_2$.

**k-Multiplication Protocol** (k-MP). We use this protocol to multiply $k$ elements in our ring. Every participant $P_\ell$ begins with a secret element $x_\ell$ given as input, as well as a uniformly random polynomial $r_\ell$. Upon finishing, participant $P_1$ learns $\prod_{\ell=1}^k x_\ell + \sum_{\ell=2}^k r_\ell$, and the rest of the participants learn nothing.

Algorithm 3 contains the detail of this protocol, and Algorithm 2 provides the base case with two parties. The idea is similar to 2-MP protocol. First, participant $P_1$ generate the additive parts $x_{1i}$ of $x_0$, and each other $P_\ell$ generates additive parts $r_{\ell i}$ of $r_\ell$, in addition to the random values $s_{\ell i}^k$. For each $x_{1i}$, participants $P \setminus \{P_1\}$ use (k-1)-MP to perform a secure multiplication of all $x_{\ell i}$, and this multiplication is carefully masked with additive
Algorithm 2 Two-party $R_q$ multiplication 2-MP

Require: Player $P_1$ holds $x_1 \in R_q$ and $P_2$ holds a pair $(x_2, r_2) \subset R_q^2$. Let $m \in \mathbb{N}$ be such that it is unfeasible to compute $2^m$ additions in $R_q$.

Ensure: Player $P_1$ learns $x_1 \cdot x_2 + r_2$.

1: procedure $k$-MP
2: Player $P_1$ generates $m$ polynomials $(x_1i \leftarrow R_q)_{i=1}^m$, such that $\sum_{i=1}^m x_{1i} = x_1$.
3: Player $P_2$ samples $m$ polynomials $(r_{2i} \leftarrow R_q)_{i=1}^m$ such that $\sum_{i=1}^m r_{2i} = r_2$.
4: for $i = 1, \ldots, m$ do
5: Player $P_1$ generates a random bit $b \leftarrow \{0, 1\}$, polynomials $(v_0, v_1) \leftarrow R_q$ and sets $v_b = x_{1i}$.
6: Player $P_1$ sends $(v_0, v_1)$ to $P_2$.
7: Player $P_2$ computes $(e_0, e_1) = (v_0 \cdot x_2 + r_{2i}, v_1 \cdot x_2 + r_{2i})$.
8: With a $(\binom{k}{1})$-OT protocol, player $P_1$ extracts $e_b$ from $P_2$.
9: Let $\hat{e}_1, \ldots, \hat{e}_m$ be the polynomials extracted by $P_1$ in each of the $m$ steps. Player $P_1$ computes $\sum_{i=1}^m \hat{e}_i = x_1 \cdot x_2 + r_2$.

uniform noise for participant $P_1$. In the end, $P_1$ performs $SP_m$ with each other participant with the goal of cancelling this noise, and obtain the product of $x_\ell$ plus the sum of the random values $r_\ell$.

$k$-Shared Multiplication Protocol ($k$-sMP). In this protocol every participant starts with a pair of additive shares $(x_i, y_i)$ of elements $x, y \in R_q$, and in the end learns an additive share $\pi_i$ of the product $\pi = x \cdot y$, i.e., $\pi_i = (\sum x_i) \cdot (\sum y_i)$. The details of this protocol are shown in Algorithm 4. Players perform $k(k - 1)$ pair-wise multiplications of shares using 2-MP (steps 3-5). The random noise added by 2-MP serves us to mask the value of the correct shares, and it is then cancelled out when adding up all polynomials (step 6).

5.4. Excalibur key generation protocols

In our key-generating protocols, players $P_i \in \{P_2, \ldots, P_k\}$ start with their secret keys $\beta_i$, and all players sample a random polynomial $s_i$ from $G_B$. These polynomials act as additive shares of $P_1$’s secret, called $\alpha$ (thus $P_1$ does not know $\alpha$ either). Upon finishing,
Algorithm 3: Multiparty multiplication of $k$ elements in $\mathbb{R}_q$

**Require:** A number of players $k \geq 3$. Player $P_1$ holds $x_1 \in \mathbb{R}_q$ and each other player $P_i$ holds a pair $(x_i, r_i) \subset \mathbb{R}_q^2$. Let $m \in \mathbb{N}$ be such that it is unfeasible to compute $2^m$ additions in $\mathbb{R}_q$.

**Ensure:** Player $P_1$ learns $\prod_{\ell=1}^{k} x_\ell + \sum_{\ell=2}^{k} r_\ell$.

1: procedure $k$-MP
2: Player $P_1$ generates $m$ polynomials $(x_{1i} \overset{\$}{\leftarrow} \mathbb{R}_q)_{i=1}^{m}$, such that $\sum_{i=1}^{m} x_{1i} = x_1$.
3: Each player $P_i$ in $\mathbb{P} \setminus \{P_1\}$ samples $(r_{\ell i} \overset{\$}{\leftarrow} \mathbb{R}_q)_{i=1}^{m}$ such that $\sum_{i=1}^{m} r_{\ell i} = r_\ell$.
4: for $i = 1, \ldots, m$ do
5: Player $P_1$ generates a random bit $b \overset{\$}{\leftarrow} \{0, 1\}$, and polynomials $(v_0, v_1) \overset{\$}{\leftarrow} \mathbb{R}_q^2$ such that $v_b = x_{1i}$.
6: Player $P_1$ sends $(v_0, v_1)$ to $P_2$.
7: for $j = 0, 1$ do
8: Players $P_2, \ldots, P_k$ perform $[k-1]$-MP($v_j \cdot x_2, (x_3, s_{2,3}^j), \ldots, (x_k, s_{k,3}^j)$).
9: Player $P_2$ adds $s_{2,3}^j$ to this output, obtaining $e_j = v_j \cdot \prod_{\ell=2}^{k} x_\ell + \sum_{\ell=2}^{k} s_{\ell i}^j$.
10: With a $\binom{2}{1}$-OT protocol, player $P_1$ extracts $e_b$ from $P_2$.
11: Note that $e_b = x_{1i} \cdot \prod_{\ell=2}^{k} x_\ell + \sum_{\ell=2}^{k} s_{\ell i}^b$.
12: for $\ell = 2, \ldots, k$ do
13: $P_1$ extracts $\hat{s}_\ell = \sum_{i=1}^{m} \hat{r}_{\ell i}^b$ from $P_i$ with $SP(b, (r_\ell^b, r_\ell^1))$, where $b = (b_1, \ldots, b_m)$ and $r_\ell^b = (\hat{r}_{\ell 1}^b, \ldots, \hat{r}_{\ell m}^b)$.
14: Finally, $P_1$ computes $\theta - \sum_{\ell=2}^{k} \hat{s}_\ell = \prod_{\ell=1}^{k} x_\ell + \sum_{\ell=2}^{k} r_\ell$.

Participant $P_1$ learns the secret key $sk_1 = \alpha \prod_{i=2}^{k} \beta_i$, as well as its public key $pk_1$. On the other hand, all other participants only learn $pk_1$. As advised in (Goubin & Vial Prado, 2016), parties generate the public key first, and $P_1$ commits to it.
Algorithm 4 Multiparty shared multiplication of $k$ elements in $R_q$

Require: Each participant $P_i$ holds a pair $(x_i, y_i)$ of elements from $R_q$.

Ensure: Each $P_i \in P$ learns an element $\pi_i$, such that $\sum_{j=1}^{k} \pi_j = (\sum_{j=1}^{k} x_j) \cdot (\sum_{j=1}^{k} y_j)$.

1: procedure $k$-sMP
2: Each $P_i$ samples $R_i = \{ r_{ij} \leftarrow R_q \mid j = [1, k] \land i \neq j \}$.
3: for $i = 1, \ldots, k$ do
4: for $j = 1, \ldots, k, j \neq i$ do
5: $P_i, P_j$ perform 2-MP($x_i, (y_j, r_{ji})$). Thus $P_i$ learns $u_{ij} = x_i \cdot y_j + r_{ji}$
6: Each participant $P_i$ computes $\pi_i = x_i y_i + \sum_{j=1, j \neq i}^{k} u_{ij} - \sum_{r \in R_i} r$.

Algorithm 5 Excalibur Public Key Generation

Require: Participant $P_1$ holds an element $s_1 \leftarrow G_B$ and each other participant holds $\beta_i = sk_i$ and $s_i \leftarrow G_B$. Let $\alpha = 2(\sum_{i=1}^{k} s_i) + 1$.

Ensure: A public key $pk_i = 2g(\alpha \prod_{i=2}^{k} \beta_i)^{-1}$ for $P_1$.

1: procedure $\text{Exc}_{pk}$
2: Each $P_i \in P$ samples $g_i \leftarrow G_B, r_i \leftarrow R_q$ and $t_{ij} \leftarrow R_q$, for $j = 1, \ldots, k$.
3: Let $r = \sum_{i=1}^{k} r_i$ and $g = \sum_{i=1}^{k} g_i$.
4: All participants perform $(k)$-MP($r_1, (\beta_2, t_{21}), \ldots, (\beta_k, t_{k1})$). Thus,$P_1$ learns $r'_1 = r_1 \cdot \prod_{i=2}^{k} \beta_i + \sum_{i=2}^{k} t_{i1}$.
5: for $i = 2, \ldots, k$ do
6: $P_i$ and the rest of participants in $P \setminus \{ P_1, P_i \}$ perform $(k-1)$-MP. $P_i$ gives $r_i, \beta_i$ as input, and each other player $P_j \in P \setminus \{ P_1, P_i \}$ gives $(\beta_j, t_{ji})$.
7: $P_i$ learns $u_i = r_i \cdot \prod_{j=2}^{k} \beta_k + \sum_{j=2, j \neq i}^{k} t_{ji}$ and computes $r'_i = u_i - \prod_{j=1, j \neq i}^{k} t_{ji}$.
8: With $g_i, r_i$ and $s_i, r'_i$, all players perform Shared k-MP twice to obtain shares of $w = g \cdot r$ and $z = \alpha \cdot r' = \alpha \prod_{i=2}^{k} \beta_i \cdot r$.

Each participant reveal their shares to $P_2$, thus $P_2$ learns $z, w$.
9: $P_2$ checks: if $z$ is not invertible in $R_q$, restart the protocol.
10: $P_2$ computes $2w(z\beta_2)^{-1} = 2g(\alpha \prod_{j=2}^{k} \beta_j)^{-1}$ and publishes it as $pk_1$. 

20
Public key generation (Exc\textsubscript{pk}). Protocol \texttt{Exc}_{pk} is used to generate the public key for participant \( P_1 \). Every participant \( P_i \) apart from \( P_1 \) holds a key pair \((\text{sk}_i, \text{pk}_i) = (\beta_i, 2h_i\beta_i^{-1})\). Player \( P_2 \) plays a special role computing some products. Upon finishing, a public key \( \text{pk}_1 \) is broadcast to everyone. This public key is a polynomial of the form \( 2g(\alpha \prod_{i=2}^{k} \beta_i)^{-1} \), for additively shared elements \( \alpha = 2(\sum_{i=1}^{k} s_i) + 1 \) and \( g = \sum_{i=1}^{k} g_i \).

The protocol is shown as Algorithm 5. It begins with the \( k \) participants sampling a gaussian share \( g_i \), and random elements \( r_i, t_{ij} \) used to additively mask polynomials, as in protocol 4. Each participant \( P_i \) learn \( r'_i \), such that, \( r'_i = \prod_{k_i=2}^{i} \beta_i \cdot r_i \) with \( r' = \sum_{i=1}^{k} r'_i \) and \( r = \sum_{i=1}^{k} r_i \). This is shown in lines 3-5 and figures 5.1, 5.2.

**Figure 5.1.** \texttt{Exc}_{pk}: \( P_1 \) learn \( r'_1 \)

**Figure 5.2.** \texttt{Exc}_{pk}: \( P_i \) learn \( u_i \)

Participants perform Shared k-MP to obtain shares of \( w = g \cdot r \) and \( z = \alpha \cdot r' \), i.e. each participant \( P_i \) obtain \( w_i \) and \( z_i \) such that \( w = \sum w_i \) and \( z = \sum z_i \). This is shown in line 6 and figures 5.3,5.4.
Once the joint secret $\prod_{i=2}^{k} \beta_i$ is shared, $P_2$ has the task of inverting it in the ring, multiplying by $\alpha^{-1}, g$ and broadcasting. To avoid $P_2$ extracting or using these secrets, they are separated into multiplicative factors that do not leak secrets (or, more precisely, such that extracting secrets from them needs to solve SF or Special GCD problems).

**Secret key generation** ($\text{Exc}_{sk}$). Protocol $\text{Exc}_{sk}$ is used to generate the secret key $\text{sk}_1$ for participant $P_1$, given secret keys $\beta_2, \ldots, \beta_k$ of the other participants. This protocol needs the same additive share of $\alpha$ of the $\text{Exc}_{pk}$ protocol (hence the need of semi-honest players).

Upon finishing, $P_1$ receives the secret key $\text{sk}_1 = \alpha \prod_{i=2}^{k} \beta_i$.

The protocol is shown as Algorithm 6, with each participant $P_i$ sampling random elements $r_{ij}$. Each participant learns $u_i = 2s_i \prod_{j=2}^{k} \beta_j + \sum_{j=2, j\neq i}^{k} r_{ji}$ from a $k$-1-MP and then computing $R_i$. This is shown in lines 3,4 and figure 5.5.
Algorithm 6 Excalibur Secret Key Generation

**Require:** Let $\alpha = 2(\sum_{i=1}^{k} s_i) + 1$ be the same additive share as in protocol 5: Participant $P_1$ holds $s_1 \in R_q$ and each other participant holds a pair $(\beta_i, s_i) \in R^2_q$.

**Ensure:** A secret-key $sk_1 = \alpha \prod_{i=2}^{k} \beta_i$ for $P_1$.

1: procedure $Exc_{sk}$
2: Each participant $P_i$ in $P \setminus \{P_1\}$ samples $(r_{ij} \leftarrow R_q)_{j=2,j\neq i}$. Let $r_i = \sum_{j=2,j\neq i}^{k} r_{ij}$.
3: for $i = 2, \ldots, k$ do
4:   $P_i$ and the rest of players from $P \setminus \{P_1, P_i\}$ perform $(k$-1$)$-MP, with $P_i$ holding $2s_i \beta_i$ and each other $P_\ell$ holding $(\beta_\ell, r_{\ell i})$.
5:   $P_i$ learns $u_i = 2s_i \prod_{j=2}^{k} \beta_j + \sum_{j=2,j\neq i}^{k} r_{ji}$ and computes
6:      $R_i = u_i - \sum_{j=2,j\neq i}^{k} r_{ij}$.
7: All participants perform $k$-MP$(2s_1 + 1, (\beta_2, R_2), \ldots, (\beta_k, R_k))$, and $P_1$ obtains
8: $sk_1 := ((2s_1 + 1) \prod_{i=1}^{k} \beta_i) + \sum_{i=2}^{k} (2s_i \prod_{j=2}^{k} \beta_j) = \alpha \prod_{i=2}^{k} \beta_i \in R_q$.

Figure 5.5. $Exc_{sk}$: Each $P_i$ learns the product of $\beta_i$’s plus some noise.
Finally, $P_1$ learns his secret key from a k-1-MP using the values $R_i$ computed before. This is shown in line 5 and figure 5.6.

Figure 5.6. $\text{Exc}_{sk}$: $P_1$ learns his secret key
6. SECURITY ANALYSIS

In this section we inspect the security of the proposed scheme against semi-honest adversaries. Throughout this section, parties \{P_1, \ldots, P_k\} participate in the protocol, player \(P_1\) receives the powerful key at the end, and \(P_2\) has the special role of inverting a polynomial in protocol \(\text{Exc}_{p_k}\).

6.1. Extracting keys after the protocol

Recall that \(P_1\) is provided an Excalibur key pair of the form

\[
(\text{sk}_1, \text{pk}_1) = (\alpha \cdot \prod_{i=2}^{k} \text{sk}_i, 2g \cdot \text{sk}_1^{-1}) \in R_q \times R_q,
\]

and assume that some set of colluded parties \(S\) try and deduce secrets. Note that extracting \(\alpha\) is a successful attack, as \(\text{sk}_1/\alpha\) can be sold as a valid NTRU key decrypting messages intended to all parties excepting party \(P_1\). Also, extracting a product of secret keys is also an attack even if individual keys are unknown, because of the inherited decryption rights. Without loss of generality, we assume that the attacker intends to extract an individual secret key, since keys can be grouped together in the view and equations are equivalent, but with different size parameters. For instance, an attacker extracting \(\text{sk}_2 \cdot \text{sk}_3\) can reformulate the instance defining \(\text{sk}' = \text{sk}_2 \cdot \text{sk}_3, p' = p_2 \cdot p_3\) and extract \(\text{sk}'\) from a wider Gaussian distribution.

**PROPOSITION 6.1.** Let \(S \subsetneq \{P_1, \ldots, P_k\}\) be a set of colluded parties. The problem of extracting \(\alpha, g, r, r'\) or any secret key \(\text{sk}_j\) of a party \(P_j \notin S\) from public values, views of the protocol and secret keys of parties in \(S\) reduces to instances of \(G_{\ell_2}^\times\)-GCD or Special Factors problems. The same holds for the problem of extracting a product of secret keys of honest parties.

The ring elements available to the uncolluded adversary are given by the output secret key, and public keys. Let \(p_i = \text{pk}_i^{-1} \in R_q\) for \(i \in \{1, \ldots, k\}\). Note that \(p_i = g_i \cdot \text{sk}_i\) for
some $g_i \in R_q$. The task of the adversary that receives $\sk_1$ is to extract any element of the set $\{\alpha, \sk_2, \ldots, \sk_k\}$ from the view

\[
\begin{aligned}
\sk_1 &= \alpha \cdot \prod_{i=2}^{k} \sk_i, \\
p_1 &= g_1 \cdot \alpha \cdot \prod_{i=2}^{k} \sk_i, \\
p_2 &= g_2 \cdot \sk_1, \\
&\vdots \\
p_k &= g_k \cdot \sk_k.
\end{aligned}
\]

**CLAIM 6.1.** Extracting $\alpha$ from $\sk_1$ is an instance of the special factors problem.

**PROOF.** Let $\beta = \prod_{i=2}^{k} \sk_i$. The task is to extract $\alpha$ from $\alpha \cdot \beta$. □

**CLAIM 6.2.** Extracting $\sk_i$ for $i \in \{2, \ldots, k\}$ from $\sk_1$ is an instance of the special factors problem.

**PROOF.** Let $\gamma = \alpha \cdot \prod_{j=2, j \neq i}^{k} \sk_j$. The task is to extract $\sk_i$ from $\sk_i \cdot \gamma$. □

**CLAIM 6.3.** Extracting $\sk_i$ for $i \in \{2, \ldots, k\}$ from the whole view is an instance of $\mathcal{G}_B^\times$-GCD problem, for some bound $B$.

**PROOF.** Write $\sk_1 = \delta \cdot \sk_i$ for some $\delta \in R_q$ and consider $p_i = g_i \cdot \sk_i$. There are no other equations involving $\sk_i$ or $g_i$, therefore solving for $\sk_i$ is exactly solving $\mathcal{G}_B^\times$-GCD. □

**CLAIM 6.4.** Extracting secret keys from the whole view and information from collusion with other parties are $\mathcal{G}_B^\times$-GCD or special factors problems.

**PROOF.** If the attacker learns $\sk_i$ for $i \in \{2, \ldots, k\}$ by collusion, then defining $\sk_i' = \sk_i / \sk_i, p_i' = p_i / p_i$ reduces to an equivalent instance of the problem of extracting another secret key. In other words, the view is now...
\[
\begin{align*}
\text{sk}_i \\
\text{sk}'_i = \alpha \cdot \prod_{j=2,j\neq i}^{k} \text{sk}_j, \quad (*) \\
p_* = g'_* \cdot \alpha \cdot \prod_{j=2,j\neq i}^{k} \text{sk}_j, \\
p_2 = g_1 \cdot \text{sk}_1, \\
\vdots \\
p_k = g_k \cdot \text{sk}_k,
\end{align*}
\]

and the only equations involving another secret-key \text{sk}_l for \(l \neq i\) are (*) and \(p_l = g_l \cdot \text{sk}_l\), defining an instance of \(G^*_B\)-GCD. The same holds for a larger set of colluded parties.

\[\square\]

CLAIM 6.5. Extracting \(\alpha, g, r\) or any \(\beta_i\) from \(z, w\) and all public values is an instance of the special factors problem.

PROOF. The task is to extract \(\alpha, g, r\) from \(w = g \cdot r\) and \(z = \alpha \cdot r'\) with \(z' = r \prod_{i=2}^{k} \beta_i\).

\[\square\]

6.2. Extracting secrets during the protocols

We address here the security of all our algorithms against semi-honest adversaries, during and after the execution. We define the corresponding functionalities to the algorithms presented in the previous chapter: \(k\)-MP, \(k\)-sMP, \(\text{Exc}_{pk}\), \(\text{Exc}_{sk}\), and then present the proofs of security of this algorithms.

\(k\)-Multiplication Protocol: \(k\)-MP. Let \(F^{k\text{-MP}}\) the functionality that computes, on input the pairs \((x_i, r_i)\), the product of \(x_i\) plus the sum of \(r_i\).

\[
F^{k\text{-MP}} : (\lambda, (x_1, (x_2, r_2), \ldots, (x_k, r_k))) \rightarrow \left(\prod_{i=1}^{k} x_i + \sum_{i=2}^{k} r_i, \perp, \perp, \ldots, \perp\right)
\]

Note that \(P_1\) has \(x_i\) and each other \(P_i\) has \((x_i, r_i)\) as input. Only \(P_1\) receive an output.

PROPOSITION 6.2. The protocol \(k\)-MP securely computes \(F^k\).
**k-Shared Multiplication Protocol:** Let $\mathcal{F}^{k\text{-sMP}}$ be the functionality that computes, on input the pairs $(x_i, y_i)$, shares $\pi_i$ of the product between the sum of $x_i$ and the sum of $y_i$.

$\mathcal{F}^{k\text{-sMP}} : (\lambda, (x_1, y_2), (x_2, y_2), \ldots, (x_k, y_k)) \rightarrow (\pi_1, \pi_2, \ldots, \pi_k)$

where $\sum_{i=1}^{k} \pi_i = (\sum_{i=1}^{k} x_i) \cdot (\sum_{i=1}^{k} y_i)$.

**Proposition 6.3.** The protocol $k$-sMP securely computes $\mathcal{F}^{k\text{-sMP}}$.

**Excalibur Keygen Public Key:** Let $\mathcal{F}^{\text{Exc}_{pk}}$ be a functionality that computes, on input the secret keys $\beta_i$, a public key $pk_1 = 2g(\alpha \prod_{j=2}^{k} \beta_j)^{-1}$ for player $P_1$, where $g$ is a sum of $k$ random elements of $\mathcal{G}$ and $\alpha$ is the sum of the secret keys (times two plus one).

$\mathcal{F}^{\text{Exc}_{pk}}(\lambda, \perp, \beta_2, \ldots, \beta_k) \rightarrow (pk_1, pk_1, \ldots, pk_1)$

**Proposition 6.4.** The protocol $\text{Exc}_{pk}$ securely computes $\mathcal{F}^{\text{Exc}_{pk}}$.

**Excalibur Keygen Secret Key:** Let $\mathcal{F}^{\text{Exc}_{sk}}$ be a functionality that computes, on input $(\beta_i, s_i)$, a secret key $sk_1 = \alpha \prod_{i=1}^{k} \beta_i$ for player $P_1$, where $\alpha$ is compute from $s_i$ and is the same of the previous protocol.

$\mathcal{F}^{\text{Exc}_{sk}}(\lambda, s_1, (\beta_2, s_2), \ldots, (\beta_k, s_k)) \rightarrow (\alpha \prod_{j=2}^{k} \beta_j, \perp, \ldots, \perp)$

**Proposition 6.5.** The protocol $\text{Exc}_{sk}$ securely computes $\mathcal{F}^{\text{Exc}_{sk}}$.

The proof of this result relies heavily on Proposition 3.1, as we need to show first that $k$-MP securely computes $\mathcal{F}^{k\text{-MP}}$, then use this result to show that $k$-sMP securely computes $\mathcal{F}^{k\text{-sMP}}$, and so forth. The proof of $k$-MP is also interesting because we need to perform an induction on $k$. In (Goubin & Vial Prado, 2016) authors discussed intuitive proofs of the base case of the above proposition, namely $k = 2$. This motivates us to present a simulation-based proof of the following proposition.
Proposition 6.6 (Simulation-based proof of (Goubin & Vial Prado, 2016), §7.1).
The protocol 2-MP securely computes $F^{2\text{-MP}}$.

Proof. We first point out that the views of the protocol are semantically secure, that is, they do not leak any secrets from the protocol if our SF assumption holds. This is straightforward to see and is detailed in the proof of (Goubin & Vial Prado, 2016), §7.1.

As in section 3.2 let $x = \{x_1, (x_2, r_2)\}$ and $S \subseteq P$ be a set of corrupted parties. Note that, as $k = 2$, we have $S \in \{\emptyset, \{P_1\}, \{P_2\}\}$. Now, according to definition 3.2, for every possible set $S$ we construct a PPT algorithm $I_S$ such that

$$(I_S(\lambda, x_S, F^{2\text{-MP}}_S(\lambda, x)), F^{2\text{-MP}}(x)) \approx (\text{view}^{2\text{-MP}}_S(\lambda, x), \text{output}^{2\text{-MP}}(\lambda, x)).$$

Case 1: $S = \{P_1\}$. The view of corrupt $P_1$ in 2-MP protocol is:

$$\text{view}^{2\text{-MP}}_S(\lambda, x) = \begin{cases} x_1, x_{11}, x_{12}, \ldots, x_{1m}, \\ (b_1, v^1_0, v^1_1, \hat{e}_1), \ldots, (b_m, v^m_0, v^m_1, \hat{e}_m). \end{cases}$$

Recall that $x_1$ is $P_1$’s input. The values $x_{1i}$ are random polynomials such that they add up to $x_1$. The random bits $b_i$ and the random polynomials $v^i_j$ are such that $v^i_j b_i$ is equal to $x_{1i}$. Finally, $\hat{e}_i$ is the output of the oblivious transfer functionality and $\sum_{i=1}^m \hat{e}_i = x_1 \cdot x_2 + r_2$.

Algorithm 7 Simulator for 2-MP corresponding to $S = \{P_1\}$

Require: $\lambda, x_1, x_1 \cdot x_2 + r_2$

1: procedure $I_{P_1}$
2: Sample $m$ random polynomials $(\bar{x}_{1i} \leftarrow R_q)_{i=1}^m$, such that $\sum_{i=1}^m \bar{x}_{1i} = x_1$.
3: for $i = 1 \ldots m$ do
4: Sample $\bar{b}_i \leftarrow \{0, 1\}$ and $(\bar{v}^i_0, \bar{v}^i_1) \leftarrow R_q$. Set $\bar{v}_{\bar{b}_i} = x_{2i}$.
5: Sample $m$ random polynomials $(\bar{e}_i \leftarrow R_q)_{i=1}^m$, such that $\sum_{i=1}^m \bar{e}_i = x_1 \cdot x_2 + r_2$.
6: Return $x_1$ together with all the values generated.

We define $I_S$, a simulator of the view of $P_1$, in algorithm 7. Its output is

$$(I_S(\lambda, x) = \begin{cases} x_1, \bar{x}_{11}, \bar{x}_{12}, \ldots, \bar{x}_{1m}, (\bar{b}_1, \bar{v}^1_0, \bar{v}^1_1, \bar{e}_1), \ldots, (\bar{b}_m, \bar{v}^m_0, \bar{v}^m_1, \bar{e}_m). \end{cases}$$
Recall that $F^{2\text{-MP}}(\mathbf{x})$ and output$^{2\text{-MP}}(\lambda, \mathbf{x})$ are both equal to $x_1 \cdot x_2 + r_2$. Therefore, we only need to verify that $I_S(\lambda, \mathbf{x}_S, F^{2\text{-MP}}_S(\lambda, \mathbf{x})) \approx \text{view}^{2\text{-MP}}_S(\lambda, \mathbf{x})$.

First, both views share $x_1$. The polynomials $x_{11}, x_{12}, \ldots, x_{1m}$ are uniformly generated by $P_1$ in 2-MP. On the other hand, $\tilde{x}_{11}, \tilde{x}_{12}, \ldots, \tilde{x}_{1m}$ are uniformly generated by $I_S$. Also, we have that $\sum_{i=1}^{m} x_{1i} = \sum_{i=1}^{m} \tilde{x}_{1i} = x_1$, yielding that these sets of polynomials are indistinguishable.

In the same fashion, each $b_i$ is a random bit and $(v_0^i, v_1^i)$ are random polynomials in $R_q$ chosen by $P_1$. On the other hand, $\tilde{b}_i$ is a random bit and $(\tilde{v}_0^i, \tilde{v}_1^i)$ are random polynomials in $R_q$ generated by $I_S$.

Finally $\tilde{e}_i$ is chosen at random, while $\hat{e}_i$ equals $x_{1i} \cdot x_2 + r_{2i}$. Note that this last value is indistinguishable from uniform because of the additive uniformly random polynomial $r_{2i}$ selected by the honest player $P_2$. We conclude that view$^{2\text{-MP}}_S(\lambda, \mathbf{x})$ and $I_S(\lambda, \mathbf{x}_S, y_1)$ are indistinguishable when $S = \{P_1\}$.

**Case 2: $S = \{P_2\}$**. The view of $P_2$ in 2-MP protocol is

$$\text{view}^{2\text{-MP}}_S(\lambda, \mathbf{x}) = \left\{ x_2, r_2, r_{21}, r_{22}, \ldots, r_{2m}, (v_0^1, v_1^1, e_0^1, e_1^1), \ldots, (v_0^m, v_1^m, e_0^m, e_1^m) \right\}$$

**Algorithm 8** Simulator for 2-MP corresponding to $S = \{P_2\}$

**Require**: $\lambda$, $(x_2, r_2)$.

1: procedure $I_{P_2}$
2:   Generate $m$ random polynomials $(\tilde{r}_{2i} \leftarrow \mathcal{R}_q)_{i=1}^m$ such that $\sum_{i=1}^{m} \tilde{r}_{2i} = r_2$.
3:   for $i = 1 \ldots m$ do
4:     Generate random polynomials $(\tilde{v}_0^i, \tilde{v}_1^i) \leftarrow \mathcal{R}_q^2$ and compute
5:       $(\tilde{e}_0^i, \tilde{e}_1^i) = (\tilde{v}_0^i \cdot x_2 + r_{2i}, \tilde{v}_1^i \cdot x_2 + r_{2i})$.
6:   Return $(x_2, r_2)$ together with all the values generated.

We define $I_S$ in algorithm 8. Note that $F_{P_2}^{2\text{-MP}}(\lambda, x_1, (x_2, r_2))$ is empty. The output of $I_S$ is:

$$I_{(P_2)}(\lambda, \mathbf{x}_{P_2}) = \left\{ x_2, r_2, \tilde{r}_{21}, \tilde{r}_{22}, \ldots, \tilde{r}_{2m}, (\tilde{v}_0^1, \tilde{v}_1^1, \tilde{e}_0^1, \tilde{e}_1^1), \ldots, (\tilde{v}_0^m, \tilde{v}_1^m, \tilde{e}_0^m, \tilde{e}_1^m) \right\}$$
Analogously as before, is it clear that $\mathcal{I}_S(\lambda, x_S) \approx \text{view}^{2\text{MP}}_S(\lambda, x)$. □

6.2.1. Proof of proposition 6.2

From now on we say that $k$-MP uses a functionality $\mathcal{F}^{SP_m}$ for the scalar product as in algorithm $SP_m$.

We proceed with an inductive argument. First, we assume that for all $k'$ such that $2 \leq k' < k$, $k'$-MP securely computes $\mathcal{F}^{k'}$. The inductive step is to show that $k$-MP ($k$-1)-MP → $\mathcal{F}^{k-1}, SP_m$ → $\mathcal{F}^{SP_m}$ securely computes $\mathcal{F}^{k-\text{MP}}$.

We begin by describing the views of parties $P_1$ (the key receiver), $P_2$, and $P_\ell$ for $\ell > 2$.

$$
\text{view}^{k-\text{MP}}_{P_1}(x, \lambda) = \begin{cases} 
    x_1, \\
    x_{11}, x_{12}, \ldots, x_{1m}, \\
    (b_1, v_1^0, v_1^1, \hat{e}_1), \ldots, (b_m, v_m^0, v_m^1, \hat{e}_m), \\
    \theta, \hat{s}_1, \ldots, \hat{s}_k
\end{cases}
$$

The elements $x_{1i}, b_i, v_j^i$ and $\hat{e}_i$ are as in the proof of proposition 6.6. On the other hand, the polynomial $\theta$ is the sum of $\hat{e}_i$ and $\hat{s}_\ell$ the sum of some random values $r_{\ell i}^j$ of player $P_\ell$.

$$
\text{view}^{k-\text{MP}}_{P_2}(x, \lambda) = \begin{cases} 
    x_2, r_2, \\
    r_{21}, r_{22}, \ldots, r_{2m}, \\
    (r_{21}^0, \ldots, r_{2m}^0), (\bar{r}_{21}^0, \ldots, \bar{r}_{2m}^0), \\
    (v_0^1, v_1^1, e_0^1, e_1^1), \ldots, (v_m^1, v_m^1, e_0^m, e_1^m)
\end{cases}
$$

In $P_2$’s view, the polynomials $r_{2i}, \bar{r}_{2i}^j$ are uniformly random values in $R_q$. 

31
\[
\text{view}_P^{k\text{-MP}}(x, \lambda) = \begin{cases} 
  x_\ell, r_\ell, \\
  r_{\ell 1}, r_{\ell 2}, \ldots, r_{\ell m} \\
  (\hat{r}^0_{\ell 1}, \ldots, \hat{r}^0_{\ell m}), (\hat{r}^1_{\ell 1}, \ldots, \hat{r}^1_{\ell m})
\end{cases}
\]

Note that the view of \(P_\ell\) is a subset of the view of \(P_2\). The tuple \((x_\ell, r_\ell)\) is the party’s input, while \(r_{\ell i}\) and \(\hat{r}^i_{\ell i}\) are uniformly random polynomials.

For the construction of the algorithm \(I_S\), we consider the following four cases:

(i) Case \(P_1, P_2 \in S\)

**Algorithm 9** Simulator \(I_S^1\) for \(k\text{-MP}\)

**Require:** \(\lambda, x_S, F_S^{k\text{-MP}}(x)\).

1: \textbf{procedure} \(I_S^1\)

2: Generate \(m\) polynomials \((\bar{x}_{1i} \leftarrow R_q)_{i=1}^m\), such that \(\sum_{i=1}^m \bar{x}_{1i} = x_1\).

3: \textbf{for} \(i = 1 \ldots m\) \textbf{do}

4: \hspace{1em} Sample \(\hat{b}_i \leftarrow \{0, 1\}\) and \((\overline{v}_0^i, \overline{v}_1^i) \leftarrow R_{2q}\). Set \(\overline{v}^i_{\hat{b}_i} = \bar{x}_{1i}\).

5: Generate \(m\) polynomials \((\overline{r}_{2i} \leftarrow R_q)_{i=1}^m\), such that \(\sum_{i=1}^m \overline{r}_{2i} = r_2\).

6: Generate \(2 \cdot m\) polynomials \((\overline{r}^0_{2i}, \overline{r}^1_{2i} \leftarrow R_{2q})_{i=1}^m\) and compute \(\overline{s}_2 = \sum_{i=1}^m \overline{r}^i_{2i}\).

7: Compute \((\overline{v}_0^i \cdot x_2, \overline{v}_1^i \cdot x_2)_{i=1}^m\).

8: \textbf{for} each \(P_\ell \in S \land \hat{i} = \ell > 2\) \textbf{do}

9: \hspace{1em} Generate \(m\) polynomials \((\overline{r}_{\ell i} \leftarrow R_q)_{i=1}^m\), such that \(r_\ell = \sum_{i=1}^m \overline{r}_{\ell i}\).

10: \hspace{1em} Generate \(2 \cdot m\) polynomials \((\overline{r}^0_{\ell i}, \overline{r}^1_{\ell i} \leftarrow R_{2q})_{i=1}^m\) and compute \(\overline{s}_\ell = \sum_{i=1}^m \overline{r}^i_{\ell i}\).

11: Generate \(2 \cdot m\) polynomials \((\overline{c}^i_{0}, \overline{c}^1_{0} \leftarrow R_q)_{i=1}^m\) and compute \(\overline{\theta} = \sum_{i=1}^m \overline{c}^i_{0}\).

12: For each \(P_\ell \notin S\), generate \(\overline{s}_\ell \leftarrow R_q\), such that \(\overline{\theta} - \sum_{\ell=2}^k \overline{s}_\ell = F_1^{k\text{-MP}}(x)\).

13: Return \(x_S\) together with all the values generated.
Rearranging values (and, for the sake of presentation, repeating some of them) we have that the following output of $\mathcal{I}_S^1$:

$$
\mathcal{I}_S^1(x, \lambda) = \begin{cases} 
  x_1, \\
  x_{11}, x_{12}, \ldots, x_{1m}, \\
  (\tilde{b}_1, \tilde{v}_0^1, \tilde{e}_1^1), \ldots, (\tilde{b}_m, \tilde{v}_0^m, \tilde{e}_m^1), \\
  \tilde{\theta}, \tilde{s}_1, \ldots, \tilde{s}_k \\
  x_2, r_2, \\
  \tilde{r}_{21}, \tilde{r}_{22}, \ldots, \tilde{r}_{2m}, \\
  (\tilde{r}_0^{21}, \ldots, \tilde{r}_0^{2m}), (\tilde{r}_1^{21}, \ldots, \tilde{r}_1^{2m}), \\
  (\tilde{v}_0^1, \tilde{e}_0^1, \tilde{e}_1^1), \ldots, (\tilde{v}_m^1, \tilde{e}_0^m, \tilde{e}_1^m), \\
  \text{for } P_\ell \in S \setminus \{P_1, P_2\} : \\
  x_\ell, r_\ell, \tilde{r}_\ell^{01}, \ldots, \tilde{r}_\ell^{0m}, \\
  (\tilde{r}_\ell^{10}, \ldots, \tilde{r}_\ell^{1m}), (\tilde{r}_\ell^{11}, \ldots, \tilde{r}_\ell^{1m}) 
\end{cases}
$$

First of all, inputs are the same for the ideal functionalities and the views of the protocol. Elements $\tilde{x}_{1i}$ are randomly generated in $\mathcal{I}_S^1$, just like $x_{1i}$ in the protocol, and are therefore indistinguishable. The same happens with $\tilde{b}_i, \tilde{v}_0^i, \tilde{e}_i^1, \tilde{\theta}, \tilde{s}_k, \tilde{r}_\ell^{0i}$ and $\tilde{r}_\ell^{1i}$.

The element $\tilde{e}_i^1$ is a random element in $R_q$, and again it is indistinguishable from $\tilde{e}_i^1 = v_{b_i}^i \prod_{\ell=2}^k x_\ell + \sum_{\ell=2}^k s_{\ell i}^k$. Finally, $\tilde{\theta} - \sum_{\ell=2}^k \tilde{s}_\ell$ equals $F_{F_{P_1}}^k(x)$, just as in protocol k-MP.

We conclude that $\mathcal{I}_S(\lambda, x_S, F^k_S(x)) \approx \text{view}_S^{k-MP}(x, \lambda)$, when $\{P_1, P_2\} \subseteq S$.

The remaining three cases are shown just as above; we only write the output of the algorithms for completeness.

(ii) Case $P_1 \notin S, P_2 \in S$
Algorithm 10 Simulator $\mathcal{I}^2_S$ for k-MP

Require: $\lambda, x_S, F^2_S(x)$.

1: procedure $\mathcal{I}^2_S$
2:   
3:   Generate $m$ polynomials $(\tilde{r}_{2i} \leftarrow R_q)_{i=1}^m$, such that $\sum_{i=1}^m \tilde{r}_{2i} = r_2$.
4:   Generate $2m$ polynomials $(\tilde{r}_{0i}^0, \tilde{r}_{2i}^1 \leftarrow R_q)_{i=1}^m$.
5:   Generate polynomials $(\tilde{v}_0^i, \tilde{v}_1^i \leftarrow R_q)_{i=1}^m$.
6:   for each $P_\ell \in S \land \ell > 2$ do
7:     Generate $m$ polynomials $(\tilde{r}_{\ell i} \leftarrow R_q)_{i=1}^m$, such that $r_\ell = \sum_{i=1}^m \tilde{r}_{\ell i}$.
8:     Generate $2 \cdot m$ polynomials $(\tilde{r}_{\ell i}^0, \tilde{r}_{\ell i}^1 \leftarrow R_q)_{i=1}^m$.
9:     if $P \setminus \{P_1\} = S$ then
10:        Compute $\tilde{e}_j^i = \tilde{v}_j^i \prod_{\ell=2}^k x_\ell + \sum_{\ell=2}^k \tilde{r}_{\ell i} + \sum_{\ell=2}^k \tilde{r}_j^\ell$, for $j = 0, 1$ and $i = 1, \ldots, m$.
11:    else
12:        Generate $2m$ random polynomials $(\tilde{e}_0^i, \tilde{e}_1^i \leftarrow R_q)_{i=1}^m$.
13:   Return $x_S, F^2_S(x)$ together with all the values generated.

Rearranging the output of $\mathcal{I}^2_S$ we have:

\[
I^2_S(x, \lambda) = \begin{cases} 
\begin{aligned}
x_2, r_2, \\
\tilde{r}_{21}, \tilde{r}_{22}, \ldots, \tilde{r}_{2m}, \\
(\tilde{r}_0^1, \ldots, \tilde{r}_0^m), (\tilde{r}_1^1, \ldots, \tilde{r}_1^m), \\
(\tilde{v}_0^1, \tilde{v}_1^1, \tilde{e}_0^1, \tilde{e}_1^1), \ldots, (\tilde{v}_0^m, \tilde{v}_1^m, \tilde{e}_0^m, \tilde{e}_1^m)
\end{aligned}
\end{cases}
\]

for $P_\ell \in S \setminus \{P_1, P_2\} :

\begin{aligned}
x_\ell, r_\ell, \tilde{r}_{\ell 1}, \ldots, \tilde{r}_{\ell m}, \\
(\tilde{r}_0^\ell, \ldots, \tilde{r}_0^m), (\tilde{r}_1^\ell, \ldots, \tilde{r}_1^m)
\end{aligned}

(iii) Case $P_1 \in S, P_2 \not\in S$
Algorithm 11 Simulator $T^3_S$ for k-MP

Require: $\lambda, x_S, F^S_k(x)$.

1: procedure $T^3_S$
2: Generate $m$ polynomials $(\tilde{x}_{1i} \sim R_q)_{i=1}^m$, such that $\sum_{i=1}^m \tilde{x}_{1i} = x_1$.
3: for $i = 1 \ldots m$ do
4: Sample $\tilde{b}_i \sim \{0, 1\}$ and $(\tilde{v}_i^0, \tilde{v}_i^1) \sim R_q^{2}$. Set $\tilde{v}_i^{\tilde{b}_i} = \tilde{x}_{1i}$.
5: for each $P^\ell \in S \land \ell > 2$ do
6: Generate $m$ polynomials $(\tilde{r}_\ell i \sim R_q)_{i=1}^m$, such that $r_\ell = \sum_{i=1}^m \tilde{r}_\ell i$.
7: Generate $2 \cdot m$ polynomials $(\tilde{c}_\ell i^0, \tilde{c}_\ell i^1 \sim R_q)_{i=1}^m$ and compute $\tilde{s}_\ell = \sum_{i=1}^m \tilde{c}_\ell i^{\tilde{b}_i}$.
8: For each $P^\ell \notin S$, generate $\tilde{s}_\ell \sim R_q$, such that $\tilde{s}_\ell = \sum_{\ell=1}^{k} \tilde{r}_\ell i^{\tilde{b}_i}$.
9: Return $x_S$ together with all the values generated.

Now, the output of $T^3_S$ is of the form:

$$I^3_S(x, \lambda) = \begin{cases} 
  x_1, \\
  \tilde{x}_{11}, \tilde{x}_{12}, \ldots, \tilde{x}_{1m}, \\
  (\tilde{b}_1, \tilde{v}_1^0, \tilde{v}_1^1), \ldots, (\tilde{b}_m, \tilde{v}_m^0, \tilde{v}_m^1), \\
  \tilde{\theta}, \tilde{s}_1, \ldots, \tilde{s}_k 
\end{cases}$$

for $P^\ell \in S \setminus \{P_1, P_2\}$:

$$x_\ell, r_\ell, \tilde{r}_\ell 1, \ldots, \tilde{r}_\ell m, \\
(\tilde{r}_\ell^0, \tilde{r}_\ell^1, \ldots, \tilde{r}_\ell^m)$$

(iv) Case $P_1 \notin S$ and $P_2 \notin S$

Algorithm 12 Simulator $T^4_S$ for k-MP

Require: $\lambda, x_S, F^S_k(x)$.

1: procedure $T^4_S$
2: for each $P^\ell \in S$ do
3: Generate $m$ polynomials $(\tilde{r}_\ell i \sim R_q)_{i=1}^m$, such that $r_\ell = \sum_{i=1}^m \tilde{r}_\ell i$.
4: Generate $2 \cdot m$ polynomials $(\tilde{r}_\ell i^0, \tilde{r}_\ell i^1 \sim R_q)_{i=1}^m$.
5: Return $x_S$ together with all the values generated.
The output of $I^4_S$ is of the form:

$$I^4_S(x, \lambda) = \{x_\ell, r_\ell, \tilde{r}_\ell, \ldots, \tilde{r}_{\ell m}, \tilde{r}_{\ell 1}, \ldots, \tilde{r}_{\ell m}, P \in S \setminus \{P_1, P_2\}\}$$

As before, we conclude that $I^4_S(\lambda, x_S, F^k_{2MP}(x)) \approx \text{view}^{k_{-MP}}_S(x, \lambda)$ for all $S \subseteq P$, which was to be shown. In conclusion, the protocol $k$-MP securely computes $F^{k_{-MP}}$. ■

6.2.2. Proof of proposition 6.3

We prove that $k$-$sMP^{2-MP} \rightarrow F^{2-MP}$ securely computes $F^{k-sMP}$. Since we have already shown that $k$-MP securely computes $F^{k_{-MP}}$, then by Proposition 1 we have that $k$-$sMP$ securely computes $F^{k-sMP}$. Begin by noting that $k$-$sMP$ is a symmetrical protocol, therefore views of all parties are similar:

$$\text{view}^{k-sMP^{2-MP} \rightarrow F^{2-MP}}_{P_i}(x, \lambda) = \left\{\begin{array}{l}
x_i, y_i, \\
(r_{i1}, r_{i2}, \ldots, r_{i, i-1}), (r_{i, i+1}, \ldots, r_{ik}), \\
(u_{i1}, u_{i2}, \ldots, u_{i, i-1}), (u_{i, i+1}, \ldots, u_{ik}) \end{array}\right.$$ 

The pair $(x_i, y_i)$ is the input of $P_1$ and $r_{ij}$ are sampled uniformly random from $R_q$. The value $u_{ij}$ is the output of the function $F^2(x_i, (y_j, r_{ji}))$ for $P_i$. We define a PPT algorithm $\mathcal{I}_S$ in algorithm 13.

Note that the output of the protocol is identical to the output of the functionality. Then we just need to show that

$$\mathcal{I}_S(\lambda, x_S, F^{k-sMP^{2-MP} \rightarrow F^{2-MP}}_S(x, \lambda)) \approx \text{view}^{k-sMP^{2-MP} \rightarrow F^{2-MP}}_{S}(x, \lambda)$$

Let us recall that the view $S$ of the protocol is the concatenation of the individual views of $P_i \in S$ and that the inputs $x_S = \{(x_i, y_i) | P_i \in S\}$ of algorithm $\mathcal{I}_S$ are identical to the inputs of these views. Hence, elements $r_{ij}$ are obtained uniformly from $R_q$ in both the algorithm and the protocol, and are therefore indistinguishable.
Algorithm 13 Simulator $I_S$ for k-sMP

Require: $\lambda$, $x_S$, $F^{k\text{-sMP}}_S(x)$.

1: procedure $I_S$
2: for $P_i \in S$ do
3:   Samples $\tilde{R}_i = \{\tilde{r}_{ij} \leftarrow R_q | j \in [1,k] \land i \neq j\}$
4: for $P_i \in S$ do
5:   for $P_j \in P \setminus \{P_i\}$ do
6:     if $P_j \in S$ then
7:       Set $\tilde{u}_{ij} = x_i \cdot x_j + \tilde{r}_{ji}$
8:     else
9:       Samples $\tilde{u}_{ij} \leftarrow R_q$
10: return $x_S$ together with the values generated.

Now, elements $u_{ij} = x_i \cdot x_j + r_{ji}$ are indistinguishable from a uniformly random polynomial in $R_q$, unless we have information about $x_j$ or $r_{ji}$. Therefore, for each $P_i$, if $P_j$ is not in $S$, the uniform $\tilde{u}_{ij}$ in this algorithm is indistinguishable from the one generated by the protocol. If $P_j$ belongs to $S$ then in both cases it is computed from $x_i \cdot x_j + r_{ji}$, and thus for any $S$ these values are also indistinguishable.

We conclude that $I_S$ is indistinguishable from the view of $S$ in k-sMP, thus k-sMP securely computes $F^{k\text{-sMP}}$. ■

6.2.3. Proof of proposition 6.4

We prove that the protocol $\text{Exc}_{pk}$, that uses functionalities $F^{k\text{-MP}}$ and $F^{k\text{-sMP}}$, securely computes $F^{\text{Exc}_{pk}}$.

The views of different parties are very similar to each other and are shown bellow:

- View of $P_1$

$$\text{view}_{P_1}^{\text{Exc}_{pk}}(x, \lambda) = \begin{cases} s_1, g_1, r_1, t_{11}, \ldots, t_{1k} \\ r'_1, z_1, w_1 \end{cases}$$

Recall that $s_1$ is $P_1$'s input, and $g_1, r_1, (t_{11}, \ldots, t_{1k})$ are uniformly generated polynomials. The value $r'_1$ is the output of $F^{k\text{-MP}}$, which computes $r_1 \prod_{i=2}^k \beta_i$. 

37
plus some random values. The pair $z_1, w_1$ are the outputs of $F^{k-sMP}$ shared functionality to compute $z = \alpha \cdot r'$ and $w = g \cdot r$, respectively.

- View of $P_i$, with $i \neq 1, 2$

$$\text{view}_{P_i}^{\text{Exc}}(x, \lambda) = \begin{cases} 
\beta_i, s_i, g_i, t_{i1}, \ldots, t_{ik} \\
u_i, r_i' \\
z_i, w_i
\end{cases}$$

This view has many elements that are analogous of the ones in the view of $P_1$, hence we only mention new ring elements. The pair $(\beta_i, s_i)$ is $P_i$'s input. The value $u_i$ is the output of $F^{k-1-MP}$ functionality to compute $r_i \prod_{j=2}^{k} \beta_j$ plus some random elements and $r_i'$ is an additive random masking for $u_i$.

- View of $P_2$

$$\text{view}_{P_2}^{\text{Exc}}(x, \lambda) = \begin{cases} 
\beta_2, s_2, g_2, t_{21}, \ldots, t_{2k} \\
u_2, r_2' \\
z_1, \ldots, z_k, z, w_1, \ldots, w_k, w
\end{cases}$$

This view is similar to the last views. The pairs $(z_i, w_i)$ are sent by $P_i$ to $P_2$. Finally, $z$ is the sum of $z_i$ elements and $w$ the sum of $w_i$ elements.

We define $I_S$ for $\text{Exc}_{pk}$ in the simulator algorithm 14. The output of $I_S$ is explained as follows:

- If $P_1 \in S$, then

$$\{\tilde{s}_1, \tilde{g}_1, \tilde{r}_1, \tilde{t}_{11}, \ldots, \tilde{t}_{1k}, \tilde{u}_2, \tilde{r}_2', \tilde{z}_1, \tilde{w}_1, \} \in I_S^{\text{Exc}_{pk}}(x, \lambda)$$

- If $P_2 \in S$, then

$$\{\tilde{\beta}_2, \tilde{s}_2, \tilde{g}_2, \tilde{r}_2, \tilde{t}_{21}, \ldots, \tilde{t}_{2k}, \tilde{u}_2, \tilde{r}_2', \tilde{z}_1, \ldots, \tilde{z}_k, z, \tilde{w}_1, \ldots, \tilde{w}_k, w\} \in I_S^{\text{Exc}_{pk}}(x, \lambda)$$

- If $P_i \in S$ and $P_i \neq P_1, P_2$, then

$$\{\tilde{\beta}_i, \tilde{s}_i, \tilde{g}_i, \tilde{r}_i, \tilde{t}_{i1}, \ldots, \tilde{t}_{ik}, \tilde{u}_i, \tilde{r}_i', \tilde{z}_i, \tilde{w}_i\} \in I_S^{\text{Exc}_{pk}}(x, \lambda)$$
Algorithm 14 Simulator for $\text{Exc}_{\text{pk}}$

Require: $\lambda, x_S, F^\text{Exc}_{\text{pk}}(x)$

1: procedure $I_S$
2: for $P_i \in S$ do
3: \hspace{1em} Sample $\tilde{g}_i \leftarrow \mathcal{G}_B$, $\tilde{r}_i \leftarrow \mathcal{R}_q$ and $(\tilde{t}_{ij}, \tilde{r}_{ij})_{j=1}^k$.
4: \hspace{1em} Sample $\tilde{w}_i, \tilde{z}_i \leftarrow \mathcal{R}_q^2$.
5: \hspace{1em} if $P_i = P_1$ then
6: \hspace{2em} Sample $\tilde{u}_i \leftarrow \mathcal{R}_q$.
7: \hspace{2em} Compute $\tilde{r}'_i = \tilde{u}_i - \sum_{j=1, j \neq i}^k t_{ij}$.
8: \hspace{1em} else
9: \hspace{2em} Sample $\tilde{u}_i \leftarrow \mathcal{R}_q$.
10: \hspace{2em} Compute $\tilde{r}'_i = \tilde{u}_i - \sum_{j=1, j \neq i}^k t_{ij}$.
11: \hspace{1em} if $P_2 \in S$ then
12: \hspace{2em} For each $P_i \notin S$ sample $\bar{z}_i$, $\bar{w}_i \leftarrow \mathcal{R}_q^2$.
13: \hspace{2em} Compute $\bar{z} = \sum_{i=1}^k \bar{z}_i$. If $\bar{z}$ is not invertible, restart the algorithm.
14: \hspace{2em} Compute $\bar{w} = \sum_{i=1}^k w_i$, such that $2\bar{w} = \text{pk}_1 \cdot \bar{z} \cdot \beta_2$.
15: \hspace{1em} Return $x_S$ together with all the values generated.

Since $S$ is a strict subset of $P$, the elements in the view of $S$ in $\text{Exc}_{\text{pk}}$ protocol are independent of each other or are masked additively with uniformly random elements, making them indistinguishable from uniform.

With the same ideas of the previous proofs, we can show that $I_S(\lambda, x_S, F^\text{Exc}_{\text{sk}}(x))$ is indistinguishable from $\text{view}^\text{Exc}_{\text{sk}}(x, \lambda)$ for every $S$. ■

6.2.4. Proof of proposition 6.5

Analogously as before, we prove that $\text{Exc}_{\text{sk}}$ protocol, that uses $F^{k\text{-MP}}$ functionality, securely computes $F^\text{Exc}_{\text{sk}}$.

This protocol is symmetrical except for the output, because only $P_1$ learns $\text{sk}_1$. The rest of the elements in all views are similar and are explained below:

$$\text{view}^\text{Exc}_{\text{sk}}(x, \lambda) = \begin{cases} \beta_i, s_i, \\ (r_{i1}, r_{i2}, \ldots, r_{i,i-1}), (r_{i,i+1}, \ldots, r_{ik}), \\ u_i, R_i \end{cases}$$
The pair \((\beta_i, s_i)\) is \(P_i\)'s input and \(r_{ij}\) are uniformly random polynomials. The value \(u_i\) is the output of the \(F^{k-1\cdot MP}\) functionality to compute \(2s_i \prod_{j=2}^{k} \beta_j\) plus some uniform elements and \(R_i\) is an additive masking of \(u_i\) by \(P_i\).

We define \(I_S\) for \(\mathsf{Exc_{sk}}\) in the simulator algorithm 15. The output of \(I_S\) is:

\[
I_S^{\mathsf{Exc_{sk}}} (\lambda, x_S, F_S^{\mathsf{Exc_{sk}}} (x)) = \{ \beta_i, s_i, \tilde{r}_{i1}, (\tilde{r}_{i2}, \ldots, \tilde{r}_{i,i-1}), (\tilde{r}_{i,i+1}, \ldots, \tilde{r}_{ik}), \tilde{u}_i, \tilde{R}_i \}_{P_i \in S}
\]

**Algorithm 15** Simulator for \(\mathsf{Exc_{sk}}\)

<table>
<thead>
<tr>
<th>Require:</th>
<th>(\lambda, x_S, F_S^{\mathsf{Exc_{sk}}} (x)).</th>
</tr>
</thead>
<tbody>
<tr>
<td>1:</td>
<td><strong>procedure</strong> (I_S)</td>
</tr>
<tr>
<td>2:</td>
<td><strong>for</strong> (P_i \in S) <strong>do</strong></td>
</tr>
<tr>
<td>3:</td>
<td>Sample ({ \tilde{r}_{ij} \overset{$}{\leftarrow} R_q \mid j = [1, k] \land i \neq j }) and (\tilde{u}_i \overset{$}{\leftarrow} R_q).</td>
</tr>
<tr>
<td>4:</td>
<td>Compute (\tilde{R}<em>i = \tilde{u}<em>i - \sum</em>{j=2,j \neq 1}^{k} \tilde{r}</em>{ij}).</td>
</tr>
<tr>
<td>5:</td>
<td><strong>Return</strong> (x_S) together with all the values generated.</td>
</tr>
</tbody>
</table>

The values \(\tilde{r}_{ij}\) are uniformly sampled, as in the protocol. As in previous proofs, \(\tilde{u}_i\) is random and indistinguishable from \(u_i = 2s_i \prod_{j=2}^{k} \beta_j + \sum_{j=1,j \neq i}^{k} r_{ji}\), because \(r_{ji}\) elements are uniformly sampled by at least one honest party. Therefore, \(\tilde{R}_i\) is indistinguishable from \(R_i\). Given this, we conclude that

\[
I_S (\lambda, x_S, F_S^{\mathsf{Exc_{sk}}} (x)) \overset{\$}{\approx} \text{view}^{\mathsf{Exc_{sk}}} (x, \lambda).
\]
7. PARAMETERS AND COMPLEXITY

The parameters \( n, q, B \) control the security of the underlying FHE-NTRU encryption scheme, and the hardness of our new problems in \( R_q \): Decisional Small Polynomial Ratio assumption and Special Factors assumption. We consider them fixed and according to the suggested values in (Stehlé & Steinfeld, 2011; Goubin & Vial Prado, 2016; López-Alt et al., 2012) for at least \( \lambda = 128 \) bits of security.

The computational complexity of our key-generation protocol amounts to \( O((2\lambda)^{k-1}) \) instances of \( \binom{2}{1} \)-OT and \( O((2\lambda)^{k-1}) \) multiplications in \( R_q \). As a heuristic estimation, in order to securely generate an Excalibur key pair between 4 participants and with 128 bits of security (this is, create a key pair that inherits decryption of three parties), there is the need to perform approximatively \( 2^{24} \) OT’s and \( 2^{24} \) products in \( R_q \), which is feasible for secure \( n, q \). With FFT or Karatsuba methods, polynomial multiplication can be carried out in time \( \tilde{O}(n,q) \), and oblivious transfers can be efficiently performed using techniques as OT extensions. For instance, (Asharov et al., 2013) reports computation of 700,000 \( \binom{2}{1} \)-OT per second over Wi-Fi, and (Ishai, Kilian, Nissim, & Petrank, 2003) reduces an OT to three cryptographic hash computations. In a regular, commercially available laptop, \( 2^{24} \) products in \( R_q \) with \( n = 512 \) and \( \log_2(q) \approx 256 \) took us around fifteen minutes (in C++ with the bignum library GMP (https://gmplib.org/)).

Although there are relatively simple efficiency improvements to our protocols, on future work we will focus on attaining security against malicious adversaries before addressing efficiency concerns. We point out that, while our protocols may not be efficient enough for practical applications with a large number of parties, once key-generation procedures are finished, the resulting keys behave as regular NTRU keys without extra complexity other than coefficient size (which does not dramatically affect the efficiency of the NTRU scheme, and is analized in (Stehlé & Steinfeld, 2011)).
7.1. Algorithmic Complexity

In this section we develop expressions for the computational complexity of our key generation protocols. Let $n, q, B$ be secure NTRU parameters, $m$ be such that it is unfeasible to compute $2^m$ additions in $R_q$, and $k$ parties are involved in the key generation procedure.

As we show below, an Excalibur key pair $(sk, pk)$ can be generated in $O((2m)^k)$ products in $R_q$ and $O((2m)^{k-1})$ basic $(\binom{2}{1})$-OT protocols. While this is certainly prohibitive for a large amount of parties and reasonable security, with fast polynomial multiplication and OT-extension techniques it is possible to generate a key pair with $k = 4$ and $m = 128$ in some minutes. Let us also mention that this key acts as other keys of the system, that is, after key generation is completed, no extra complexity is to be expected for encryption, decryption or homomorphic procedures (other than coefficient size, whose impact in complexity is analyzed in (López-Alt et al., 2012)).

**Definition 7.1.** Let $\theta$ (resp. $\pi$) be the computational cost of performing a $(\binom{2}{1})$-OT protocol (resp. performing a multiplication in $R_q$).

**Proposition 7.1.** The computational cost of performing $k$-MP is approximately $(2m)^{k-1}\pi + (2m)^{k-1}\theta$. The computational cost of performing $k$-sMP is approximately $mk(k-1)(2\pi + \theta)$.

**Proof.** First, note that the computational cost of performing $SP_m$ (with $\kappa = m$) is $m\theta$ (see algorithm 1 and note that the scalar product is not expressed in terms of full $R_q$ products), and the cost of performing 2-MP is $(2\pi + \theta)m$. Let $u_k$ be the computational cost of performing $k$-MP. Given the description of the protocol in algorithm 3, we have the following recurrence:

$$\begin{cases} 
  u_k = 2mu_{k-1} + km\theta, \\
  u_2 = (2\pi + \theta)m.
\end{cases}$$
To see this, note that parties first perform $2m$ instances of (k-1)-MP, then $m \left( \begin{array}{c} k \\ 1 \end{array} \right)$-OT extractions, and finally $(k - 1)$ scalar products $SP_m$. The solution to this equation for $k \geq 3$ is given by

$$u_k = (2m)^{k-2}u_2 + m\theta \sum_{i=3}^{k} i(2m)^{k-i},$$

and therefore the cost of k-MP is approximately $(2m)^{k-1}$ products in $R_q$ and $(2m)^{k-1}$ $(\begin{array}{c} k \\ 1 \end{array})$-OT protocols.

Let now $v_k$ be the computational cost of performing k-sMP. Parties perform $k(k - 1)$ instances of 2-MP (algorithm 4), therefore we have

$$v_k = mk(k - 1)(2\pi + \theta).$$

PROPOSITION 7.2. The cost of performing both $\text{Exc}_{pk}$ and $\text{Exc}_{sk}$ between k parties is $O(u_k)$, that is, $O((2m)^{k-1})$ products in $R_q$ and $O((2m)^{k-1})(\begin{array}{c} k \\ 1 \end{array})$-OT protocols.

PROOF. In $\text{Exc}_{pk}$ (algorithm 5), parties perform one k-MP and $(k - 1)$ instances of (k-1)-MP. Also, in $\text{Exc}_{sk}$ (algorithm 6) parties perform $(k - 1)$ instances of (k-1)-MP and one final k-MP. The leading term of computational cost in both cases is therefore $O((2m)^{k-1})$ products and $((2m)^{k-1})$ oblivious transfers.

Remark: With $m = 128$ bits of security against brute force additions in $R_q$, four parties need to compute around $2^{24}$ products in $R_q$ and $2^{24}$ 1-out-of-2 oblivious transfer protocols.
8. CONCLUSIONS

Our work extends the original Excalibur key-generating protocols for an arbitrary hierarchy of keys, and presents formal simulation-based proofs for the security of these protocols.

While we have defined our protocols with respect to a participant $P_1$ that aims to obtain a key that decrypts messages of participants $P_2, \ldots, P_k$, we can immediately extend these for any DAG-like hierarchy, as follows. Starting from the leaves, which already have their key pairs, first generate the keys of their parents. For a parent with $k$ children, these keys are of the form $\alpha \prod \beta_i$, with $\alpha = 2(\sum s_i) + 1$ a sum of $k$ elements sampled from $G_B$, and each $\beta_i$ the secret key of one of the leaves. In turn, these keys are used to generate the keys for nodes at higher levels, and so forth. Note that keys generated in this fashion are of the form $\alpha \prod \gamma_j$, where $\alpha$ is as above and $\gamma$ is itself a product of secret keys of lower levels (which are either leaves or keys of the same form). Thus, secret keys for members of higher hierarchies are again products of elements distributing according to a gaussian distribution, so all of our security proofs can be extended for more complex hierarchies; we only need to update our hardness assumptions so that they hold with wider gaussian distributions, that is, bounded by $2kB + 1$ instead of $B$, where $k$ is the outdegree of the hierarchy.

8.1. Future work

As we said early, we prove that our protocols are secure if they run in isolation, but if run alongside other protocols they could not be. It remains to prove that our protocols are secure for Universal Composition framework.

Another potential future contribution is improve efficiency to our protocols. And finally, it remains to see the problem of key-generation in the presence of malicious or adaptive adversaries. In particular, we note that this case is not immediate from our results, as Definition 3.2 and Proposition 3.1 must be tightened when considering the malicious case,
because tampering with intermediate values may affect the input of other protocols, even if they involve honest players only.
REFERENCES


