PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE
ESCUELA DE INGENIERÍA

EXCALIBUR KEY-GENERATION PROTOCOLS FOR DAG HIERARCHIC DECRYPTION

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

PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE
ESCUELA DE INGENIERÍA

EXCALIBUR KEY-GENERATION PROTOCOLS FOR DAG HIERARCHIC DECRYPTION

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

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 Vrog̃ 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

LIST OF FIGURES

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-rencriptación 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 $(\mathsf{sk}_A, \mathsf{pk}_A)$ 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 Cannetti (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 $\mathcal{X}_B : A \mapsto \{0,1\}$ 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:

$$\mathcal{X}_B(x) = \begin{cases} 1 & if \quad x \in A \\ 0 & if \quad x \notin A \end{cases}$$

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 $e \xleftarrow{\$} S$ denote that e is sampled from S according to the uniform distribution.

A distribution ensamble $X = \{X(\lambda, a)\}_{\lambda \in \mathbb{N}, a \in D}$ 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 λ is taken to be the security parameter. As usual, the complexity of our protocols is measured in terms of the security parameter.

Let μ a negligible function. Two distribution ensembles X and Y are statistically indistinguishables, writed as $X \overset{\text{s}}{\approx} Y$, if for all sufficiently large λ 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 $X \overset{\text{c}}{\approx} Y$, if for every algorithm $\mathcal{D}$ that is probabilistic polynomial-time in it first input, for all sufficiently large λ and all a and all auxiliary information $w \in \{0,1\}^*$ we have

$$|\Pr[\mathcal{D}(\lambda, a, w, x) = 1] - \Pr[D(\lambda, a, w, y) = 1]| < \mu(\lambda)$$

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 $x \in \mathbb{Z}$, $[x \bmod q]$ 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 ζ such that $\zeta^n = 1$. There are exactly n different n-th roots of unity:

$$e^{2k\pi i/n} = \cos\left(\frac{2k\pi}{n}\right) + i \sin\left(\frac{2k\pi}{n}\right),$$

for $0 \leq k < n$ and n a posive integer. A primitive n-th root of unity is an n-th root of unity ζ whose order is n.

The n-th cyclotomic polynomial Φ_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 Φ_{2^k} is equal to $x^{2^{k-1}} + 1$, for k a integer.

For n a power of two, let

$$R_q \stackrel{\text{def}}{=} \frac{\mathbb{Z}_q[x]}{\Phi_{2^n}(x)}$$

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 $\mathbb{Z}_q$.

2.3. Invertible bounded Gaussian distributions in the quotient ring

In the following definition, let Γ_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 $\mathbf{x} \leftarrow \Gamma_B$, and restart if $||\mathbf{x}||_\infty > B$.*

(ii) *Output the polynomial $p \in R_q$ whose coefficients vector is $\lfloor \mathbf{x} \rceil$.*

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

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

Decryption: Given a ciphertext c and secret-key sk, output $m = c \cdot \mathsf{sk} \bmod 2$.

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* $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathrm{Keygen}(1^k)$*,* $m \in \{0,1\}$ *and let* $c \leftarrow \mathrm{Enc}(\mathsf{pk}, m)$*. Let* $\theta \in R_q$ *be* M*-bounded polynomial satisfying* $\theta \mod 2 = 1$*. If* $M < (\frac{1}{71})(\frac{q}{n^2K^2})$ *then*

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

Corollary 2.1. *(The multikey property) Let* $(\mathsf{pk}_1, \mathsf{sk}_1)$ *and* $(\mathsf{pk}_2, \mathsf{sk}_2)$ *be valid pairs of keys,* $m_1, m_2 \in \{0,1\}$ *and let* $c_1 \leftarrow \mathrm{Enc}(\mathsf{pk}_1, m_1)$*,* $c_2 \leftarrow \mathrm{Enc}(\mathsf{pk}_2, m_2)$*. Let* $\tilde{\mathsf{sk}} = \mathsf{sk}_1 \cdot \mathsf{sk}_2 \in R_q$*. Then*

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

provided that K *is small enough.*

3. SECURITY DEFINITIONS

A cryptographic protocol include 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 have 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 involves 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 exactely, 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 choose which parties to corrupt during the computation, based in 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 comming from othe machines within the same protocol instance; and a write-only tape holding the outgoings 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 speciall and predeterminated ITI wich is the fist in 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 attemps 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 λ the security parameter. A functionality is a $(k+1)$-ary function

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

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. $f_i : \mathbb{N} \times (\{0,1\}^*)^k \to \{0,1\}^*$. We only are interesting in functionalities that are computable in polynomial time with respect to the security parameter λ, this means there is a deterministic turing machine with input $\mathbf{x}$ can output $f(\lambda, \mathbf{x})$ in poynomial time with respect the security parameter λ.

A protocol π 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 π a protocol for computing f. The *view* of the i-th party when executing π on input $\mathbf{x} = \{x_1, \ldots, x_k\}$ and security parameter λ, denoted as

$\text{view}_i^\pi(\lambda, \mathbf{x})$, is a tuple

$$(x_i, r_i, m_1^i, \ldots, m_j^i),$$

where r^i is the content of the internal random tape of the i-th party, and $m_1^i, \ldots, m_j^i$ represents the messages sent and received with other parties during the execution of π. For a set $S \subseteq P$ of parties, we set $\text{view}_S^\pi(\lambda, \mathbf{x})$ as the concatenation of each tuple $\text{view}_i^\pi(\lambda, \mathbf{x})$. We also write f_S as the tuple formed of each f_i, for $i \in S$, and $\mathbf{x}_S$ as the tuple formed of each x_i, $i \in S$.

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

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

$$(\mathcal{I}_S(\lambda, \mathbf{x}_S, f_S(\lambda, \mathbf{x})), f(\lambda, \mathbf{x})) \overset{s}{\approx} (\textit{view}_S^\pi(\lambda, \mathbf{x}), \textit{output}^\pi(\lambda, \mathbf{x}))$$

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 π_i is called only after π_{i-1} returns, and additionally, $\rho^{\pi_1, \ldots, \pi_\ell}$ pauses when executing each π_i. Denote by $\rho^{\pi_1 \to \phi_1, \ldots, \pi_\ell \to \phi_\ell}$ the protocol where instead of calling to each π_i, we use an oracle computing the functionality f_i.

PROPOSITION 3.1. *If every π_i securely computes ϕ_i, and $\rho^{\pi_1 \to \phi_1, \ldots, \pi_\ell \to \phi_\ell}$ 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 $s \in R_q$, ψ 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 $\psi \leftarrow \overline{\Upsilon}$ and $s \leftarrow U(R_q)$. Given access to an oracle $\mathcal{O}$ that produces samples in $R_q \times R_q$, distinguish whether $\mathcal{O}$ 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 $\mathsf{pk} = 2g(2f + 1)^{-1} \in R_q$ where $f, g \leftarrow \mathcal{G}_B$, and*

(ii) *a uniformly random polynomial* $u \xleftarrow{\$} R_q$.

4.2. Small factorizations in R_q

Additionally, the semi-honest security rely on the hardness of the following problems. Let ξ_B^l be the Gaussian Product Distribution defined as follows.

Definition 4.3 (Gaussian Product Distribution)**.** *Let ξ_B^l be the distribution that samples polynomials $p_i \leftarrow \mathcal{G}_B^\times$ for $i = 1, \ldots, l$ and outputs $\prod_{i=1}^l p_i$.*

Definition 4.4 (Special factors problem)**.** *Let $\alpha \leftarrow \xi_B^l$ and $\beta \leftarrow \xi_B^m$. The Special Factors Problem is to output α, β 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 factorizarion domain, so for any unit $u \in R_q$ there is a posible factorization $c = u \cdot (u^{-1}c)$. In order to find α, β, 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 $\alpha, \beta \leftarrow \mathcal{G}_B^\times$ and $y \leftarrow \xi_B^l$. Given $u = \alpha \cdot y$ and $v = \alpha \cdot \beta$ and access to the indicator function of $\{\alpha, \beta, y\}$, output α, β 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 $\mathsf{Exc}_{\mathsf{pk}}$ and $\mathsf{Exc}_{\mathsf{sk}}$, with the aim of create a key pair $(\mathsf{sk}_1, \mathsf{pk}_1)$ for participant P_1, based on the set $(\mathsf{sk}_i, \mathsf{pk}_i)$ of all other participants. As we have mentioned, the pair $(\mathsf{sk}_1, \mathsf{pk}_1)$ 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

$$\mathcal{F}^{\binom{n}{1}\text{-OT}} : (\lambda, (x_1, \ldots, x_n), m) \mapsto (\perp, x_m),$$

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

5.2. A Scalar Product protocol

We define the *scalar product* funcionality as

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

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 $\vec{p}^{(1)} = (p_1^{(1)}, \ldots, p_m^{(1)})$.

Note that

$$\sum_{i=1}^{m} p_i^{(b_i)} = (p_1^0, \ldots, p_m^0) \cdot \vec{b}^c + (p_1^1, \ldots, p_m^1) \cdot \vec{b},$$

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 SP_m be a two-party protocol performing $\mathcal{F}^{SP_m}$ 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 $((p_i^{(0)}, p_i^{(1)}) \in R_q^2)_{i=1}^m$. Let κ be such that it is unfeasible to compute 2^κ additions in R_q.

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

1: **procedure** SP_m

2: P_1 samples κ vectors $(\vec{b_i} \xleftarrow{\$} \mathbb{Z}^m)_{i=1}^\kappa$ such that $\sum_{i=1}^{\kappa} \vec{b_i} = \vec{b}$.

3: **for** $i = 1 \ldots \kappa$ **do**

4: P_1 samples a bit σ and two vectors $\vec{a}_0, \vec{a}_1 \xleftarrow{\$} \{0,1\}^m$. 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

$$\begin{aligned} \vec{d_0} &= (p_1^0, \ldots, p_m^0) \cdot \vec{a}_0^c + (p_1^1, \ldots, p_m^1) \cdot \vec{a}_0 \\ \vec{d_1} &= (p_1^0, \ldots, p_m^0) \cdot \vec{a}_1^c + (p_1^1, \ldots, p_m^1) \cdot \vec{a}_1 \end{aligned}$$

7: With a $\binom{2}{1}$-OT protocol, P_1 extracts $\gamma_i := \vec{d_\sigma}$ from P_2.

8: P_1 computes $\gamma = \sum_{i=1}^{\kappa} \gamma_i = (p_1^0, \ldots, p_m^0) \cdot \vec{b}^c + (p_1^1, \ldots, p_m^1) \cdot \vec{b}$.

Remark: This protocol can be restated as a $\binom{2^m}{1}$-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 $x' = \vec{b}$ with a $\binom{2^m}{1}$-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_ℓ begins with a secret element x_ℓ given as input, as well as a uniformly random polynomial r_ℓ. 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_1, and each other P_ℓ generates additive parts $r_{\ell i}$ of r_ℓ, in addition to the random values $s^b_{\ell i}$. For each x_{1i}, participants $P \setminus \{P_1\}$ use (k-1)-MP to perform a secure multiplication of all x_ell, 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 $(x_{1i} \xleftarrow{\$} R_q)_{i=1}^m$, such that $\sum_{i=1}^{m} x_{1i} = x_1$.

3: Player P_2 samples m polynomials $(r_{2i} \xleftarrow{\$} R_q)_{i=1}^m$ 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 $b \xleftarrow{\$} \{0, 1\}$, polynomials $(v_0, v_1) \xleftarrow{\$} R_q^2$ 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 $\binom{2}{1}$-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_ℓ plus the sum of the random values r_ℓ.

***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 π_i of the product $\pi = x \cdot y$, i.e., $\sum \pi_i = (\sum x_i) \cdot (\sum y_i)$. 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 β_i, and all players sample a random polynomial s_i from $\mathcal{G}_B$. These polynomials act as additive shares of P_1's secret, called α (thus P_1 does not know α either). Upon finishing,

Algorithm 3 Multiparty multiplication of k elements in R_q

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

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

procedure k-MP

Player P_1 generates m polynomials $(x_{1i} \xleftarrow{\$} R_q)_{i=1}^m$, such that $\sum_{i=1}^{m} x_{1i} = x_1$.

Each player P_ℓ in $P \backslash \{P_1\}$ samples $(r_{\ell i} \xleftarrow{\$} R_q)_{i=1}^m$ such that $\sum_{i=1}^{m} r_{\ell i} = r_\ell$,

and $2m$ polynomials $(\hat{r}_{\ell i}^b \xleftarrow{\$} R_q)_{i=1}^m$ for $b = 0, 1$. Let $s_{\ell i}^b = r_{\ell i} + \hat{r}_{\ell i}^b$.

for $i = 1, \ldots, m$ **do**

Player P_1 generates a random bit $b \xleftarrow{\$} \{0, 1\}$, and polynomials $(v_0, v_1) \xleftarrow{\$} R_q^2$ such that $v_b = x_{1i}$.

Player P_1 sends (v_0, v_1) to P_2.

for $j = 0, 1$ **do**

Players $P_2, \ldots, P_k$ perform [k-1]-MP$(v_j \cdot x_2, (x_3, s_{3i}^j), \ldots, (x_k, s_{ki}^j))$.

P_2 learns $v_j \cdot \prod_{\ell=2}^{k} x_\ell + \sum_{\ell=3}^{k} s_{\ell i}^j$.

Player P_2 adds s_{2i}^j to this output, obtaining $e_j = v_j \cdot \prod_{\ell=2}^{k} x_\ell + \sum_{\ell=2}^{k} s_{\ell i}^j$

With a $\binom{2}{1}$-OT protocol, player P_1 extracts e_b from P_2.

Note that $e_b = x_{1i} \cdot \prod_{\ell=2}^{k} x_\ell + \sum_{\ell=2}^{k} s_{\ell i}^b$.

Let $\hat{e}_i$ be the polynomials extracted in each of these m steps, and b_i the random bits. P_1 computes $\theta := \sum_{i=1}^{m} \hat{e}_i = \prod_{\ell=1}^{k} x_\ell + \sum_{\ell=2}^{k} r_\ell + \sum_{\ell=2}^{k} \left(\sum_{i=1}^{m} \hat{r}_{\ell i}^{b_i} \right)$.

for $\ell = 2, \ldots, k$ **do**

P_1 extracts $\hat{s}_\ell = \sum_{i=1}^{m} \hat{r}_{\ell i}^{b_i}$ from P_ℓ with $SP(\mathbf{b}, (\mathbf{r}_\ell^0, \mathbf{r}_\ell^1))$,

where $\mathbf{b} = (b_1, \ldots, b_m)$ and $\mathbf{r}_\ell^j = (\hat{r}_{\ell 1}^j, \ldots, \hat{r}_{\ell m}^j)$.

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 $\mathsf{sk}_1 = \alpha \prod_{i=2}^{k} \beta_i$, 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 π_i, such that $\sum_{j=1}^{k} \pi_j = (\sum_{j=1}^{k} x_j) \cdot (\sum_{j=i}^{k} y_j)$.

1: **procedure** k-sMP

2: Each P_i samples $R_i = \{r_{ij} \xleftarrow{R} R_q \mid j = [1, k] \wedge i \neq j\}$.

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 $s_1 \leftarrow \mathcal{G}_B$ and each other participant holds $\beta_i = \mathsf{sk}_i$ and $s_i \leftarrow \mathcal{G}_B$. Let $\alpha = 2(\sum_{i=1}^{k} s_i) + 1$.

Ensure: A public key $\mathsf{pk}_1 = 2g(\alpha \prod_{i=2}^{k} \beta_i)^{-1}$ for P_1.

1: **procedure** $\mathsf{Exc}_{\mathsf{pk}}$

2: Each $P_i \in P$ samples $g_i \leftarrow \mathcal{G}_B$, $r_i \xleftarrow{\$} R_q$ and $t_{ij} \xleftarrow{\$} R_q$, for $j = 1, ..., k$.
Let $r = \sum_{i=1}^{k} r_i$ and $g = \sum_{i=1}^{k} g_i$.

3: All participants perform (k)-MP$(r_1, (\beta_2, t_{21}), \ldots, (\beta_k, t_{k1}))$. Thus,
P_1 learns $r'_1 = r_1 \cdot \prod_{i=2}^{k} \beta_i + \sum_{i=2}^{k} t_{i1}$.

4: **for** $i = 2, \ldots, k$ **do**

5: 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 (β_j, t_{ji}).
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 $r'_i = u_i - \sum_{j=1, j\neq i}^{k} t_{ij}$.

6: 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.

7: P_2 checks: if z is not invertible in R_q, restart the protocol.

8: P_2 computes $2w(z\beta_2)^{-1} = 2g(\alpha \prod_{j=2}^{k} \beta_j)^{-1}$ and publishes it as pk_1.

Public key generation($\mathsf{Exc}_{\mathsf{pk}}$). Protocol $\mathsf{Exc}_{\mathsf{pk}}$ is used to generate the public key for participant P_1. Every participant P_i apart from P_1 holds a key pair $(\mathsf{sk}_i, \mathsf{pk}_i) = (\beta_i, 2h_i\beta_i^{-1})$. Player P_2 plays a special role computing some products. Upon finishing, a public key 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, $r'_i = \prod_{i=2}^{k} \beta_i \cdot r_i$ 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.

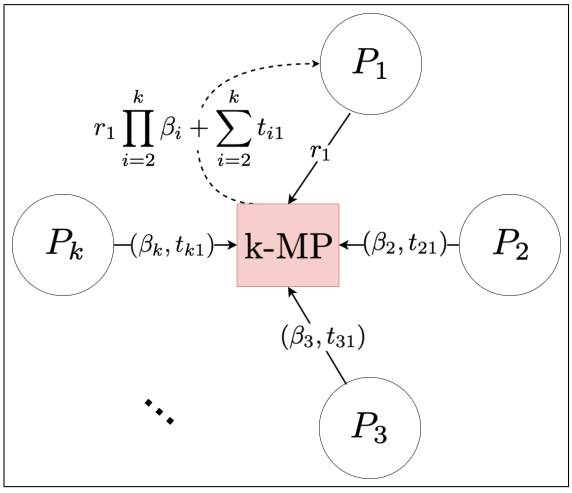

Figure 5.1. $\mathsf{Exc}_{\mathsf{pk}}$: P_1 learn r'_1

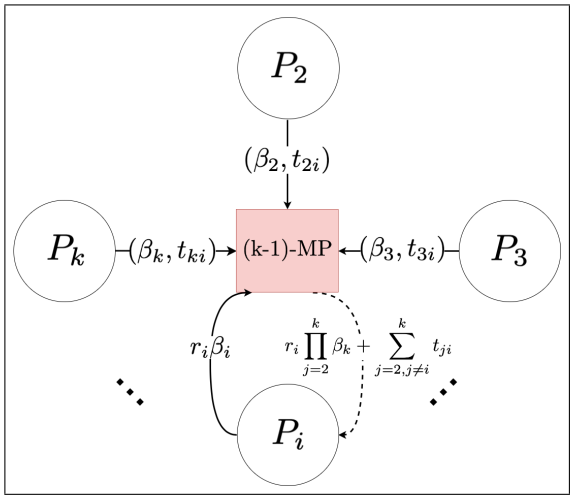

Figure 5.2. $\mathsf{Exc}_{\mathsf{pk}}$: P_i learn u_i

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.

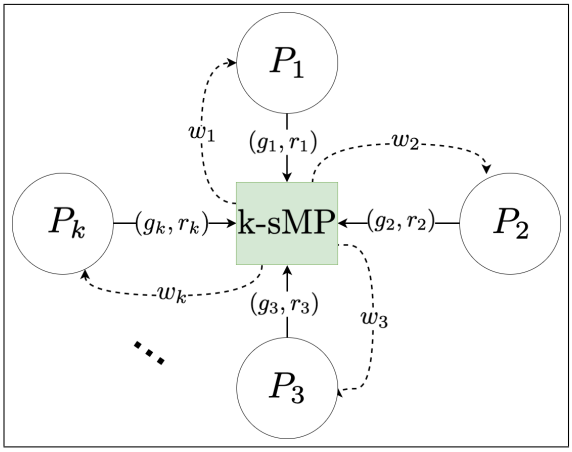

Figure 5.3. $\mathsf{Exc}_{\mathsf{pk}}$: Each P_i learn w_i

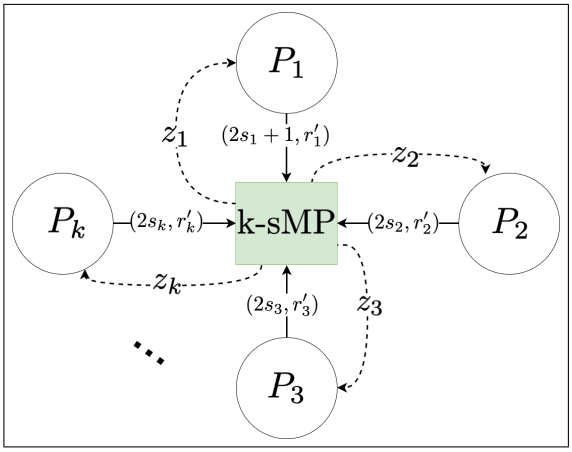

Figure 5.4. $\mathsf{Exc}_{\mathsf{pk}}$: Each P_i learn z_i

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 α^{-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($\mathsf{Exc}_{\mathsf{sk}}$). Protocol $\mathsf{Exc}_{\mathsf{sk}}$ is used to generate the secret key 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 α of the $\mathsf{Exc}_{\mathsf{pk}}$ protocol (hence the need of semi-honest players). Upon finishing, P_1 receives the secret key $\mathsf{sk}_1 = \alpha \prod_{i=2}^{k} \beta_i$.

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 $(\beta_i, s_i) \in R_q^2$.

Ensure: A secret-key $\mathsf{sk}_1 = \alpha \prod_{i=2}^{k} \beta_i$ for P_1.

1: **procedure** $\mathsf{Exc}_{\mathsf{sk}}$

2: Each participant P_i in $P \setminus \{P_1\}$ samples $(r_{ij} \xleftarrow{R} R_q)_{j=2, j\neq i}^{k}$. 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_ℓ holding $(\beta_\ell, r_{\ell i})$.

P_i learns $u_i = 2s_i \prod_{j=2}^{k} \beta_j + \sum_{j=2, j\neq i}^{k} r_{ji}$ and computes

$R_i = u_i - \sum_{j=2, j\neq i}^{k} r_{ij}$.

5: All participants perform k-MP$(2s_1 + 1, (\beta_2, R_2), \ldots, (\beta_k, R_k))$, and P_1 obtains

$\mathsf{sk}_1 := ((2s_1 + 1) \prod_{i=1}^{k} \beta_i) + \sum_{i=2}^{k} \left(2s_i \prod_{j=2}^{k} \beta_j\right) = \alpha \prod_{i=2}^{k} \beta_i \in R_q$

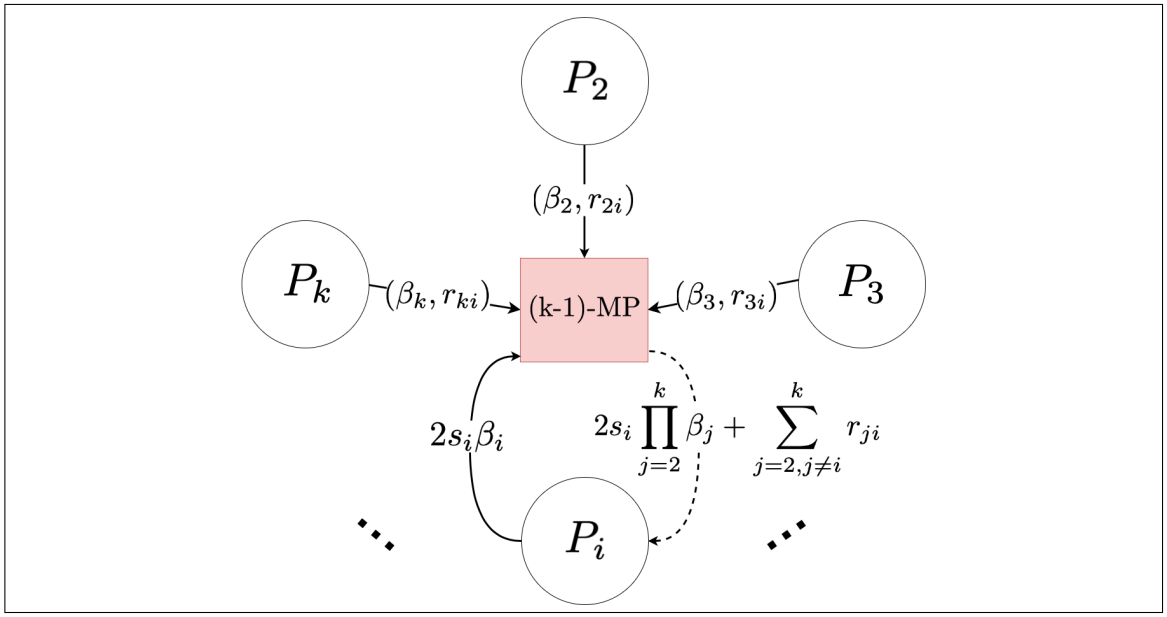

Figure 5.5. $\mathsf{Exc}_{\mathsf{sk}}$: Each P_i learns the product of β_i's plus some noise

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.

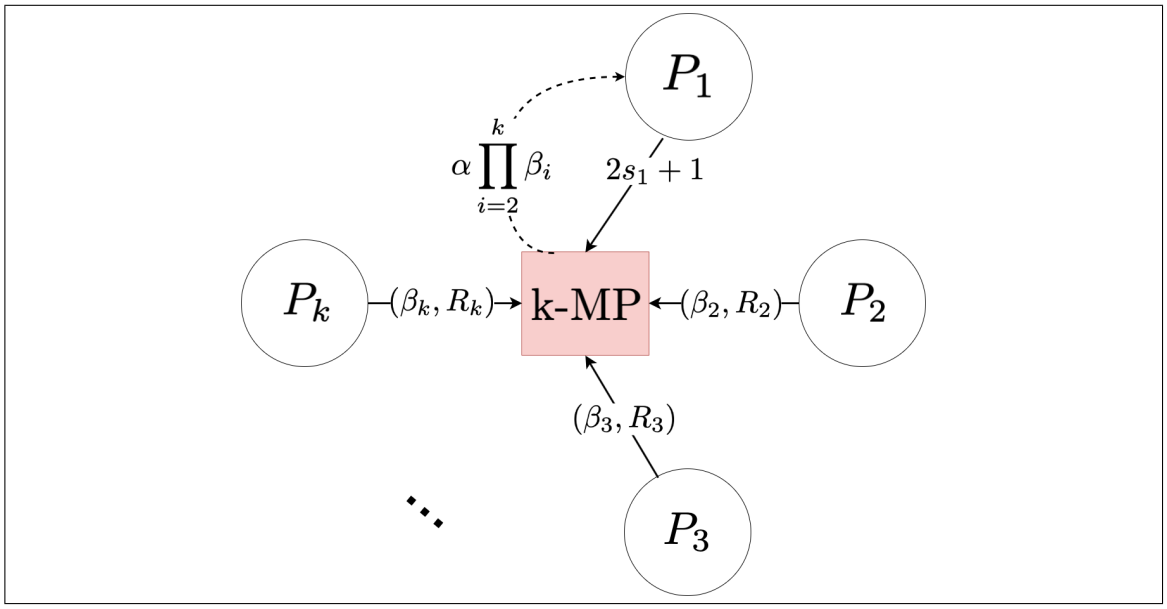

Figure 5.6. $\mathsf{Exc}_{\mathsf{sk}}$: P_1 learns his secret key

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 $\mathsf{Exc}_{\mathsf{pk}}$.

6.1. Extracting keys after the protocol

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

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

and assume that some set of colluded parties S try and deduce secrets. Note that extracting α is a successful attack, as sk_1/α 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 $\mathsf{sk}_2 \cdot \mathsf{sk}_3$ can reformulate the instance defining $\mathsf{sk}' = \mathsf{sk}_2 \cdot \mathsf{sk}_3, p' = p_2 \cdot p_3$ and extract 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 α, g, r, r' or any secret key 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 $\mathcal{G}_B^{\times}$-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 $p_i = \mathsf{pk}^{-1} \in R_q$ for $i \in \{1, \ldots, k\}$. Note that $p_i = g_i \cdot \mathsf{sk}_i$ for

some $g_i \in R_q$. The task of the adversary that receives sk_1 is to extract any element of the set $\{\alpha, \mathsf{sk}_2, \ldots, \mathsf{sk}_k\}$ from the view

$$\begin{cases} \mathsf{sk}_1 = \alpha \cdot \prod_{i=2}^{k} \mathsf{sk}_i, \\ p_1 = g_1 \cdot \alpha \cdot \prod_{i=2}^{k} \mathsf{sk}_i, \\ p_2 = g_2 \cdot \mathsf{sk}_1, \\ \vdots \\ p_k = g_k \cdot \mathsf{sk}_k. \end{cases}$$

CLAIM 6.1. *Extracting α from sk_1 is an instance of the special factors problem.*

PROOF. Let $\beta = \prod_{i=2}^{k} \mathsf{sk}_i$. The task is to extract α 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 $\gamma = \alpha \cdot \prod_{j=2, j\neq i}^{k} \mathsf{sk}_j$. The task is to extract sk_i from $\mathsf{sk}_i \cdot \gamma$. □

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 $\mathsf{sk}_1 = \delta \cdot \mathsf{sk}_i$ for some $\delta \in R_q$ and consider $p_i = g_i \cdot \mathsf{sk}_i$. 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 $\mathsf{sk}'_* = \mathsf{sk}_*/\mathsf{sk}_i, p'_1 = p_1/p_i$ reduces to an equivalent instance of the problem of extracting another secret key. In other words, the view is now

$$\begin{cases} \mathsf{sk}_i \\ \mathsf{sk}'_* = \alpha \cdot \prod_{j=2, j\neq i}^{k} \mathsf{sk}_j, \qquad (*) \\ p_* = g'_* \cdot \alpha \cdot \prod_{j=2, j\neq i}^{k} \mathsf{sk}_j, \\ p_2 = g_1 \cdot \mathsf{sk}_1, \\ \vdots \\ p_k = g_k \cdot \mathsf{sk}_k, \end{cases}$$

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

CLAIM 6.5. *Extracting α, g, r or any β_i from z, w and all public values is an instance of the special factors problem.*

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

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: $\mathsf{k\text{-}MP}$, $\mathsf{k\text{-}sMP}$, $\mathsf{Exc}_{\mathsf{pk}}$, $\mathsf{Exc}_{\mathsf{sk}}$; and then present the proofs of security of this algorithms.

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

$$\mathcal{F}^{\mathsf{k\text{-}MP}} : (\lambda, x_1, (x_2, r_2), \ldots, (x_k, r_k)) \rightarrow \Big(\big(\prod_{i=1}^{k} x_i + \sum_{i=2}^{k} r_i \big), \perp, \perp, \ldots, \perp \Big)$$

Note that P_i 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* $\mathsf{k\text{-}MP}$ *securely computes* $\mathcal{F}^k$.

k-Shared Multiplication Protocol: k-sMP. Let $\mathcal{F}^{\text{k-sMP}}$ the functionality that computes, on input the pairs (x_i, y_i), shares π_i of the product between the sum of x_i and the sum of y_i.

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

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

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

Excalibur Keygen Public Key: $\mathsf{Exc}_{\mathsf{pk}}$. Let $\mathcal{F}^{\mathsf{Exc}_{\mathsf{pk}}}$ a functionality that computes, on input the secret keys β_i, a public key $\mathsf{pk}_1 = 2g(\alpha \prod_{j=2}^{k} \beta_j)^{-1}$ for P_1, where g is a sum of k random elements of $\overline{\mathcal{G}}$ and α is the sum of the secret keys (times two plus one).

$$\mathcal{F}^{\mathsf{Exc}_{\mathsf{pk}}}(\lambda, \perp, \beta_2, \ldots, \beta_k) \rightarrow (\mathsf{pk}_1, \mathsf{pk}_1, \cdots, \mathsf{pk}_1)$$

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

Excalibur Keygen Secret Key: $\mathsf{Exc}_{\mathsf{sk}}$. Let $\mathcal{F}^{\mathsf{Exc}_{\mathsf{sk}}}$ a functionality that computes, on input (β_i, s_i), a secret key $\mathsf{sk}_1 = \alpha \prod_{i=1}^{2} \beta_i$ for player P_1, where α is compute from s_i and is the same of the previous protocol.

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

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

The proof of this result relies heavily on Proposition 3.1, as we need to show first that k-MP securely computes $\mathcal{F}^{\text{k-MP}}$, then use this result to show that k-sMP securely computes $\mathcal{F}^{\text{k-sMP}}$, 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* $\mathcal{F}^{\text{2-MP}}$.

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 $\mathbf{x} = \{x_1, (x_2, r_2)\}$ and $S \subsetneq P$ 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 $\mathcal{I}_S$ such that

$$(\mathcal{I}_S(\lambda, \mathbf{x}_S, \mathcal{F}_S^{\text{2-MP}}(\lambda, \mathbf{x})), \mathcal{F}^{\text{2-MP}}(\mathbf{x})) \overset{s}{\approx} (\text{view}_S^{\text{2-MP}}(\lambda, \mathbf{x}), \text{output}^{\text{2-MP}}(\lambda, \mathbf{x})).$$

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

$$\text{view}_S^{\text{2-MP}}(\lambda, \mathbf{x}) = \begin{cases} x_1, x_{11}, x_{12}, \ldots, x_{1m}, \\ (b_1, v_0^1, v_1^1, \hat{e}_1), \cdots, (b_m, v_0^m, v_1^m, \hat{e}_m). \end{cases}$$

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_j^i are such that $v_{b_i}^i$ 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: λ, x_1, $x_1 \cdot x_2 + r_2$

procedure $\mathcal{I}_{P_1}$
  Sample m random polynomials $(\tilde{x}_{1i} \overset{\$}{\leftarrow} R_q)_{i=1}^m$, such that $\sum_{i=1}^m \tilde{x}_{1i} = x_1$.
  for $i = 1 \ldots m$ **do**
    Sample $\tilde{b}_i \overset{\$}{\leftarrow} \{0, 1\}$ and $(\tilde{v}_0^i, \tilde{v}_1^i) \overset{\$}{\leftarrow} R_q^2$. Set $\tilde{v}_{\tilde{b}_i} = x_{\mathcal{I}i}$.
  Sample m random polynomials $(\tilde{e}_i \overset{\$}{\leftarrow} R_q)_{i=1}^m$, such that $\sum_{i=1}^m \tilde{e}_i = x_1 \cdot x_2 + r_2$.
  Return x_1 together with all the values generated.

We define $\mathcal{I}_S$, a simulator of the view of P_1, in algorithm 7. Its output is

$$\mathcal{I}_S(\lambda, \mathbf{x}) = \left\{ x_1, \tilde{x}_{11}, \tilde{x}_{12}, \ldots, \tilde{x}_{1m}, (\tilde{b}_1, \tilde{v}_0^1, \tilde{v}_1^1, \tilde{e}_1), \cdots, (\tilde{b}_m, \tilde{v}_0^m, \tilde{v}_1^m, \tilde{e}_m). \right.$$

Recall that $\mathcal{F}^{\text{2-MP}}(\mathbf{x})$ and $\text{output}^{\text{2-MP}}(\lambda, \mathbf{x})$ are both equal to $x_1 \cdot x_2 + r_2$. Therefore, we only need to verify that $\mathcal{I}_S(\lambda, \mathbf{x}_S, \mathcal{F}_S^2(\lambda, \mathbf{x})) \overset{s}{\approx} \text{view}_S^{\text{2-MP}}(\lambda, \mathbf{x})$.

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 $\mathcal{I}_S$. 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 $(\tilde{v}_0^i, \tilde{v}_0^i)$ are random polynomials in R_q generated by $\mathcal{I}_S$.

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 $\text{view}_S^{\text{2-MP}}(\lambda, \mathbf{x})$ and $\mathcal{I}_S(\lambda, \mathbf{x}_S, y_1)$ are indistinguishable when $S = \{P_1\}$.

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

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

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

Require: λ, (x_2, r_2).

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

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

$$\mathcal{I}_{\{P_2\}}(\lambda, \mathbf{x}_{P_2}) = \left\{ \; x_2, r_2, \tilde{r}_{21}, \tilde{r}_{22}, \ldots, \tilde{r}_{2m}, (\tilde{v}_0^1, \tilde{v}_1^1, \tilde{e}_0^1, \tilde{e}_1^1), \cdots, (\tilde{v}_0^m, \tilde{v}_1^m, \tilde{e}_0^m, \tilde{e}_1^m) \; \right\}$$

Analogously as before, is it clear that $\mathcal{I}_S(\lambda, \mathbf{x}_S) \overset{s}{\approx} \text{view}_S^{\text{2-MP}}(\lambda, \mathbf{x})$. $\square$

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$^{\text{(k-1)-MP}\to\mathcal{F}^{k-1}, SP_m\to\mathcal{F}^{SP_m}}$ securely computes $\mathcal{F}^{\text{k-MP}}$.

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

$$\text{view}_{P_1}^{\text{k-MP}}(\mathbf{x}, \lambda) = \begin{cases} x_1, \\ x_{11}, x_{12}, \ldots, x_{1m}, \\ (b_1, v_0^1, v_1^1, \hat{e}_1), \ldots, (b_m, v_0^m, v_1^m, \hat{e}_m), \\ \theta, \hat{s}_1, \ldots, \hat{s}_k \end{cases}$$

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 θ is the sum of $\hat{e}_i$ and $\hat{s}_\ell$ the sum of some random values $r_{\ell i}^j$ of player P_ℓ.

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

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

$$\text{view}_{P_\ell}^{\mathsf{k\text{-}MP}}(\mathbf{x}, \lambda) = \begin{cases} x_\ell, r_\ell, \\ r_{\ell 1}, r_{\ell 2}, \ldots, r_{\ell m} \\ (\hat{r}_{\ell 1}^0, \ldots, \hat{r}_{\ell m}^0), (\hat{r}_{\ell 1}^1, \ldots, \hat{r}_{\ell m}^1) \end{cases}$$

Note that the view of P_ℓ is a subset of the view of P_2. The tuple (x_ℓ, r_ℓ) is the party's input, while $r_{\ell i}$ and $\hat{r}_{\ell i}^j$ are uniformly random polynomials.

For the construction of the algorithm $\mathcal{I}_S$, we consider the following four cases:

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

Algorithm 9 Simulator $\mathcal{I}_S^1$ for $\mathsf{k\text{-}MP}$

Require: λ, $\mathbf{x}_S$, $\mathcal{F}_S^{\mathsf{k\text{-}MP}}(\mathbf{x})$.

procedure $\mathcal{I}_S^1$

Generate m polynomials $(\tilde{x}_{1i} \xleftarrow{\$} R_q)_{i=1}^m$, such that $\sum_{i=1}^m \tilde{x}_{1i} = x_1$.

for $i = 1 \ldots m$ **do**

Sample $\tilde{b}_i \xleftarrow{\$} \{0, 1\}$ and $(\tilde{v}_0^i, \tilde{v}_1^i) \xleftarrow{\$} R_q^2$. Set $\tilde{v}_{b_i}^i = \tilde{x}_{1i}$.

Generate m polynomials $(\tilde{r}_{2i} \xleftarrow{\$} R_q)_{i=1}^m$, such that $\sum_{i=1}^m \tilde{r}_{2i} = r_2$.

Generate $2 \cdot m$ polynomials $(\tilde{r}_{2i}^0, \tilde{r}_{2i}^1 \xleftarrow{\$} R_q^2)_{i=1}^m$ and compute $\tilde{s}_2 = \sum_{i=1}^m \tilde{r}_{2i}^{\tilde{b}_i}$.

Compute $(\tilde{v}_0^i \cdot x_2, \tilde{v}_1^i \cdot x_2)_{i=1}^m$.

for each $P_\ell \in S \wedge i = \ell > 2$ **do**

Generate m polynomials $(\tilde{r}_{\ell i} \xleftarrow{\$} R_q)_{i=1}^m$, such that $r_\ell = \sum_{i=1}^m \tilde{r}_{\ell i}$.

Generate $2 \cdot m$ polynomials $(\tilde{r}_{\ell i}^0, \tilde{r}_{\ell i}^1 \xleftarrow{\$} R_q^2)_{i=1}^m$ and compute $\tilde{s}_\ell = \sum_{i=1}^m \tilde{r}_{\ell i}^{\tilde{b}_i}$.

Generate $2 \cdot m$ polynomials $(\tilde{e}_0^i, \tilde{e}_1^i \xleftarrow{\$} R_q)_{i=1}^m$ and compute $\tilde{\theta} = \sum_{i=1}^m \tilde{e}_{b_i}^i$.

For each $P_\ell \notin S$, generate $\tilde{s}_\ell \xleftarrow{\$} R_q$, such that $\tilde{\theta} - \sum_{\ell=2}^k \tilde{s}_\ell = \mathcal{F}_{P_1}^{\mathsf{k\text{-}MP}}(\mathbf{x})$.

Return $\mathbf{x}_S$ 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 $\mathcal{I}_S^1$:

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

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 $\tilde{b}_i$, $\tilde{v}_0^i$, $\tilde{v}_1^i$, $\tilde{r}_{\ell i}$, $\tilde{r}_{\ell i}^0$ and $\tilde{r}_{\ell i}^1$.

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

We conclude that $\mathcal{I}_S(\lambda, \mathbf{x}_S, \mathcal{F}_S^k(\mathbf{x})) \overset{s}{\approx} \text{view}_S^{\mathsf{k\text{-}MP}}(\mathbf{x}, \lambda)$, 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 $\mathcal{I}_S^2$ for k-MP

Require: λ, $\mathbf{x}_S$, $\mathcal{F}_S^k(\mathbf{x})$.

procedure $\mathcal{I}_S^2$

Generate m polynomials $(\tilde{r}_{2i} \xleftarrow{\$} R_q)_{i=1}^m$, such that $\sum_{i=1}^m \tilde{r}_{2i} = r_2$.

Generate $2m$ polynomials $(\tilde{r}_{2i}^0, \tilde{r}_{2i}^1 \xleftarrow{\$} R_q^2)_{i=1}^m$.

Generate polynomials $(\tilde{v}_0^i, \tilde{v}_1^i \xleftarrow{\$} R_q^2)_{i=1}^m$.

for each $P_\ell \in S \wedge \ell > 2$ **do**

Generate m polynomials $(\tilde{r}_{\ell i} \xleftarrow{\$} R_q)_{i=1}^m$, such that $r_\ell = \sum_{i=1}^m \tilde{r}_{\ell i}$.

Generate $2 \cdot m$ polynomials $(\tilde{r}_{\ell i}^0, \tilde{r}_{\ell i}^1 \xleftarrow{\$} R_q^2)_{i=1}^m$.

if $P \setminus \{P_1\} = S$ **then**

Compute $\tilde{e}_j^i = \tilde{v}_j^i \prod_{\ell=2}^k x_\ell + \sum_{\ell=2}^k \tilde{r}_{\ell i} + \sum_{\ell=2}^k \tilde{r}_{\ell i}^j$, for $j = 0, 1$ and $i = 1, \ldots, m$.

else

Generate $2m$ random polynomials $(\tilde{e}_0^i, \tilde{e}_0^i \xleftarrow{\$} R_q)_{i=1}^m$.

Return $\mathbf{x}_S$, $\mathcal{F}_S^k(\mathbf{x})$ together with all the values generated.

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

$$I_S^2(\mathbf{x}, \lambda) = \begin{cases} x_2, r_2, \\ \tilde{r}_{21}, \tilde{r}_{22}, \ldots, \tilde{r}_{2m}, \\ (\tilde{r}_{21}^0, \ldots, \tilde{r}_{2m}^0), (\tilde{r}_{21}^1, \ldots, \tilde{r}_{2m}^1), \\ (\tilde{v}_0^1, \tilde{v}_1^1, \tilde{e}_0^1, \tilde{e}_1^1), \cdots, (\tilde{v}_0^m, \tilde{v}_1^m, \tilde{e}_0^m, \tilde{e}_1^m) \\ \\ \text{for } P_\ell \in S \setminus \{P_1, P_2\} : \\ \quad x_\ell, r_\ell, \tilde{r}_{\ell 1}, \ldots, \tilde{r}_{\ell m}, \\ \quad (\tilde{r}_{\ell 1}^0, \ldots, \tilde{r}_{\ell m}^0), (\tilde{r}_{\ell 1}^1, \ldots, \tilde{r}_{\ell m}^1) \end{cases}$$

(iii) Case $P_1 \in S, P_2 \notin S$

Algorithm 11 Simulator $\mathcal{I}_S^3$ for k-MP

Require: λ, $\mathbf{x}_S$, $\mathcal{F}_S^k(\mathbf{x})$.

procedure $\mathcal{I}_S^3$

  Generate m polynomials $(\tilde{x}_{1i} \xleftarrow{\$} R_q)_{i=1}^m$, such that $\sum_{i=1}^m \tilde{x}_{1i} = x_1$.

  for $i = 1 \dots m$ **do**

    Sample $\tilde{b}_i \xleftarrow{\$} \{0,1\}$ and $(\tilde{v}_0^i, \tilde{v}_1^i) \xleftarrow{\$} R_q^2$. Set $\tilde{v}_{b_i}^i = \tilde{x}_{1i}$.

  for each $P_\ell \in S \wedge \ell > 2$ **do**

    Generate m polynomials $(\tilde{r}_{\ell i} \xleftarrow{\$} R_q)_{i=1}^m$, such that $r_\ell = \sum_{i=1}^m \tilde{r}_{\ell i}$.

    Generate $2 \cdot m$ polynomials $(\tilde{r}_{\ell i}^0, \tilde{r}_{\ell i}^1 \xleftarrow{\$} R_q^2)_{i=1}^m$ and compute $\tilde{s}_\ell = \sum_{i=1}^m \tilde{r}_{\ell i}^{\tilde{b}_i}$.

  Generate $2 \cdot m$ polynomials $(\tilde{e}_0^i, \tilde{e}_1^i \xleftarrow{\$} R_q)_{i=1}^m$ and compute $\tilde{\theta} = \sum_{i=1}^m \tilde{e}_{b_i}^i$.

  For each $P_\ell \notin S$, generate $\tilde{s}_\ell \xleftarrow{\$} R_q$, such that $\tilde{\theta} - \sum_{\ell=2}^k \tilde{s}_\ell = \mathcal{F}_{P_1}^k(\mathbf{x})$.

  Return $\mathbf{x}_S$ together with the values generated.

Now, the output of $\mathcal{I}_S^3$ is of the form:

$$I_S^3(\mathbf{x}, \lambda) = \begin{cases} x_1, \\ \tilde{x}_{11}, \tilde{x}_{12}, \dots, \tilde{x}_{1m}, \\ (\tilde{b}_1, \tilde{v}_0^1, \tilde{v}_1^1, \tilde{e}_1^1), \cdots, (\tilde{b}_m, \tilde{v}_0^m, \tilde{v}_1^1, \tilde{e}_m^m), \\ \tilde{\theta}, \tilde{s}_1, \dots, \tilde{s}_k \\ \\ \text{for } P_\ell \in S \backslash \{P_1, P_2\} : \\ \quad x_\ell, r_\ell, \tilde{r}_{\ell 1}, \dots, \tilde{r}_{\ell m}, \\ \quad (\tilde{r}_{\ell 1}^0, \dots, \tilde{r}_{\ell m}^0), (\tilde{r}_{\ell 1}^1, \dots, \tilde{r}_{\ell m}^1) \end{cases}$$

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

Algorithm 12 Simulator $\mathcal{I}_S^4$ for k-MP

Require: λ, $\mathbf{x}_S$, $\mathcal{F}_S^k(\mathbf{x})$.

procedure $\mathcal{I}_S^4$

  for each $P_\ell \in S$ **do**

    Generate m polynomials $(\tilde{r}_{\ell i} \xleftarrow{\$} R_q)_{i=1}^m$, such that $r_\ell = \sum_{i=1}^m \tilde{r}_{\ell i}$.

    Generate $2 \cdot m$ polynomials $(\tilde{r}_{\ell i}^0, \tilde{r}_{\ell i}^1 \xleftarrow{\$} R_q^2)_{i=1}^m$.

  Return $\mathbf{x}_S$ together with all the values generated.

The output of $\mathcal{I}_S^4$ is of the form:

$$I_S^4(\mathbf{x}, \lambda) = \{x_\ell, r_\ell, \tilde{r}_{\ell 1}, \ldots, \tilde{r}_{\ell m}, \tilde{r}^0_{\ell 1}, \ldots, \tilde{r}^0_{\ell m}, \tilde{r}^1_{\ell 1}, \ldots, \tilde{r}^1_{\ell m} | P_\ell \in S \setminus \{P_1, P_2\}\}$$

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

6.2.2. Proof of proposition 6.3

We prove that k-sMP$^{\text{2-MP}\to\mathcal{F}^{\text{2-MP}}}$ securely computes $\mathcal{F}^{\text{k-sMP}}$. Since we have already shown that k-MP securely computes $\mathcal{F}^{\text{k-MP}}$, then by Proposition 1 we have that k-sMP securely computes $\mathcal{F}^{\text{k-sMP}}$.

Begin by noting that k-sMP is a symmetrical protocol, therefore views of all parties are similar:

$$\text{view}_{P_i}^{\text{k-sMP}^{\text{2-MP}\to\mathcal{F}^{\text{2-MP}}}}(\mathbf{x}, \lambda) = \begin{cases} x_i, y_i, \\ (r_{i1}, r_{i2}, \ldots, r_{i,i-1}), (r_{i,i+1}, \ldots, r_{ik}), \\ (u_{i1}, u_{i2}, \ldots, u_{i,i-1}), (u_{i,i+1}, \ldots, u_{ik}), \end{cases}$$

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 $\mathcal{F}^2(x_i, (y_j, r_{ji}))$ 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

$$\mathcal{I}_S\big(\lambda, \mathbf{x}_S, \mathcal{F}_S^{\text{k-sMP}^{\text{2-MP}\to\mathcal{F}^{\text{2-MP}}}}(\mathbf{x}, \lambda)\big) \overset{s}{\approx} \text{view}_S^{\text{k-sMP}^{\text{2-MP}\to\mathcal{F}^{\text{2-MP}}}}(\mathbf{x}, \lambda)$$

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 $\mathbf{x}_S = \{(x_i, y_i) | P_i \in S\}$ 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 $\mathcal{I}_S$ for k-sMP

Require: λ, $\mathbf{x}_S$, $\mathcal{F}_S^{\text{k-sMP}}(\mathbf{x})$.

procedure $\mathcal{I}_S$
  for $P_i \in S$ **do**
    Samples $\tilde{R}_i = \{\tilde{r}_{ij} \xleftarrow{\$} R_q | j \in [1,k] \wedge i \neq j\}$
  for $P_i \in S$ **do**
    for $P_j \in P \setminus \{P_1\}$ **do**
      if $P_j \in S$ **then**
        Set $\tilde{u}_{ij} = x_i \cdot x_j + \tilde{r}_{ji}$
      else
        Samples $\tilde{u}_{ij} \xleftarrow{\$} R_q$
  Return $\mathbf{x}_S$ 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 $\mathcal{I}_S$ is indistinguishable from the view of S in k-sMP, thus k-sMP securely computes $\mathcal{F}^{\text{k-sMP}}$. ■

6.2.3. Proof of proposition 6.4

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

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

- View of P_1

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

  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 $\mathcal{F}^{\text{k-MP}}$, which computes $r_1 \prod_{i=2}^{k} \beta_i$

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

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

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

  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 (β_i, s_i) is P_i'input. The value u_i is the output of $\mathcal{F}^{\text{k-1-MP}}$ 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

$$\text{view}_{P_2}^{\mathsf{Exc}_{\mathsf{pk}}}(\mathbf{x}, \lambda) = \begin{cases} \beta_2, s_2, g_2, r_2, t_{21}, \ldots, t_{2k} \\ u_2, r'_2, \\ z_1, \ldots, z_k, z, w_1, \ldots, w_k, w \end{cases}$$

  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 $\mathcal{I}_S$ for $\mathsf{Exc}_{\mathsf{pk}}$ in the simulator algorithm 14. The output of $\mathcal{I}_S$ is explained as follows:

  – If $P_1 \in S$, then

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

  – If $P_2 \in S$, then

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

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

$$\{\tilde{\beta}_i, \tilde{s}_i, \tilde{g}_i, \tilde{r}_i, \tilde{t}_{i1}, \ldots, \tilde{t}_{ik}, \tilde{u}_i, \tilde{r}'_i, \tilde{z}_i, \tilde{w}_i\} \in \mathcal{I}_S^{\mathsf{Exc}_{\mathsf{pk}}}(\mathbf{x}, \lambda)$$

Algorithm 14 Simulator for $\mathsf{Exc}_{\mathsf{pk}}$

Require: λ, $\mathbf{x}_S$, $\mathcal{F}_S^{\mathsf{Exc}_{\mathsf{pk}}}(\mathbf{x})$

procedure $\mathcal{I}_S$
  for $P_i \in S$ **do**
    Sample $\tilde{g}_i \xleftarrow{\$} \mathcal{G}_B$, $\tilde{r}_i \xleftarrow{\$} R_q$ and $(\tilde{t}_{ij} \xleftarrow{\$} R_q)_{j=1}^k$.
    Sample $\tilde{w}_i, \tilde{z}_i \xleftarrow{\$} R_q^2$.
    if $P_i = P_1$ **then**
      Sample $\tilde{r}_i \xleftarrow{\$} R_q$.
    else
      Sample $\tilde{u}_i \xleftarrow{\$} R_q$.
      Compute $\tilde{r}'_i = \tilde{u}_i - \sum_{j=1, j\neq i}^k t_{ij}$.
  if $P_2 \in S$ **then**
    For each $P_i \notin S$ sample $\tilde{z}_i, \tilde{w}_i \xleftarrow{\$} R_q^2$.
    Compute $\tilde{z} = \sum_{i=1}^k \tilde{z}_i$. If $\tilde{z}$ is not invertible, restart the algorithm.
    Compute $\tilde{w} = \sum_{i=1}^k \tilde{w}_i$, such that $2\tilde{w} = \mathsf{pk}_1 \cdot \tilde{z} \cdot \beta_2$.
  Return $\mathbf{x}_S$ together with all the values generated.

Since S is a strict subset of P, the elements in the view of S in $\mathsf{Exc}_{\mathsf{pk}}$ 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 $\mathcal{I}_S(\lambda, \mathbf{x}_S, \mathcal{F}_S^{\mathsf{Exc}_{\mathsf{sk}}}(\mathbf{x}))$ is indistinguishable from $\text{view}_S^{\mathsf{Exc}_{\mathsf{sk}}}(\mathbf{x}, \lambda)$ for every S. ■

6.2.4. Proof of proposition 6.5

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

This protocol is symmetrical except for the output, because only P_1 learns sk_1. The rest of the elements in all views are similar and are explained below:

$$\text{view}_{P_i}^{\mathsf{Exc}_{\mathsf{sk}}}(\mathbf{x}, \lambda) = \begin{cases} \beta_i, s_i, \\ (r_{i1}, r_{i2}, \ldots, r_{i,i-1}), (r_{i,i+1}, \ldots, r_{ik}), \\ u_i, R_i \end{cases}$$

The pair (β_i, s_i) is P_i's input and r_{ij} are uniformly random polynomials. The value u_i is the output of $\mathcal{F}^{\mathsf{k\text{-}1\text{-}MP}}$ 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 $\mathcal{I}_S$ for $\mathsf{Exc_{sk}}$ in the simulator algorithm 15. The output of $\mathcal{I}_S$ is:

$$\mathcal{I}_S^{\mathsf{Exc_{sk}}}(\lambda, \mathbf{x}_S, \mathcal{F}_S^{\mathsf{Exc_{sk}}}(\mathbf{x})) = \{\beta_i, s_i, \tilde{r}_{i1}, (\tilde{r}_{i2}, \ldots, \tilde{r}_{i,i-1}), (\tilde{r}_{i,i+1}, \ldots, \tilde{r}_{ik}), \tilde{u}_i, \tilde{R}_i\}_{P_i \in S}$$

Algorithm 15 Simulator for $\mathsf{Exc_{sk}}$

Require: λ, $\mathbf{x}_S$, $\mathcal{F}_S^{\mathsf{Exc_{sk}}}(\mathbf{x})$.

1: **procedure** $\mathcal{I}_S$
2: **for** $P_i \in S$ **do**
3: Sample $\{\tilde{r}_{ij} \xleftarrow{\$} R_q \mid j = [1, k] \wedge i \neq j\}$ and $\tilde{u}_i \xleftarrow{\$} R_q$.
4: Compute $\tilde{R}_i = \tilde{u}_i - \sum_{j=2, j\neq 1}^{k} \tilde{r}_{ij}$.
5: Return $\mathbf{x}_S$ together with all the values generated.

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

$$\mathcal{I}_S(\lambda, \mathbf{x}_S, \mathcal{F}_S^{\mathsf{Exc_{sk}}}(\mathbf{x})) \overset{s}{\approx} \text{view}_S^{\mathsf{Exc_{sk}}}(\mathbf{x}, \lambda). \qquad \blacksquare$$

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 $(\mathsf{sk}, \mathsf{pk})$ 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 θ (resp. π) 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 approximatively* $(2m)^{k-1}\pi + (2m)^{k-1}\theta$. *The computational cost of performing* k-sMP *is approximatively* $mk(k-1)(2\pi + \theta)$.

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 $\binom{2}{1}$-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 $(2m)^{k-1}$ $\binom{2}{1}$-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* $\mathsf{Exc}_{\mathsf{pk}}$ *and* $\mathsf{Exc}_{\mathsf{sk}}$ *between* k *parties is* $O(u_k)$*, that is,* $O((2m)^{k-1})$ *products in* R_q *and* $O((2m)^{k-1})$ $\binom{2}{1}$-OT *protocols.*

PROOF. In $\mathsf{Exc}_{\mathsf{pk}}$ (algorithm 5), parties perform one k-MP and $(k-1)$ instances of (k-1)-MP. Also, in $\mathsf{Exc}_{\mathsf{sk}}$ (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 $\mathcal{G}_B$, and each β_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 α is as above and γ 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 proove 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 out 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 form 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

Asharov, G., Lindell, Y., Schneider, T., & Zohner, M. (2013, 11). *More efficient oblivious transfer and extensions for faster secure computation.*

Canetti, R. (2000). Security and composition of multiparty cryptographic protocols. *Journal of CRYPTOLOGY*, *13*(1), 143–202.

Canetti, R. (2001). Universally composable security: A new paradigm for cryptographic protocols. In *Foundations of computer science, 2001. proceedings. 42nd ieee symposium on* (pp. 136–145).

Canetti, R. (2006, September). Security and composition of cryptographic protocols: A tutorial (part i). *SIGACT News*, *37*(3), 67–92. Retrieved from `http://doi.acm.org/10.1145/1165555.1165570` doi: 10.1145/1165555.1165570

Goldreich, O., et al. (2005). Foundations of cryptography–a primer. *Foundations and Trends® in Theoretical Computer Science*, *1*(1), 1–116.

Goubin, L., & Vial Prado, F. J. (2016). Blending fhe-ntru keys – the excalibur property. In *Progress in cryptology – indocrypt 2016.* Springer International Publishing.

Ishai, Y., Kilian, J., Nissim, K., & Petrank, E. (2003, 06). *Extending oblivious transfers efficiently.*

Li, S.-D., & Dai, Y.-Q. (2005, Mar 01). Secure two-party computational geometry. *Journal of Computer Science and Technology*, *20*(2), 258–263. Retrieved from `https://doi.org/10.1007/s11390-005-0258-z` doi: 10.1007/s11390-005-0258-z

Lindell, Y. (2017). How to simulate it–a tutorial on the simulation proof technique. In

Tutorials on the foundations of cryptography (pp. 277–346). Springer.

López-Alt, A., Tromer, E., & Vaikuntanathan, V. (2012). On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In *Proceedings of the forty-fourth annual acm symposium on theory of computing* (pp. 1219–1234). New York, NY, USA: ACM.

Stehlé, D., & Steinfeld, R. (2011). Making ntru as secure as worst-case problems over ideal lattices. In *Eurocrypt*.

Thomae, E., & Wolf, C. (2012). Solving underdetermined systems of multivariate quadratic equations revisited. In M. Fischlin, J. Buchmann, & M. Manulis (Eds.), *Public key cryptography – pkc 2012* (pp. 156–171). Berlin, Heidelberg: Springer Berlin Heidelberg.