Excalibur Key-Generation Protocols For DAG Hierarchic Decryption

Louis Goubin[1], Geraldine Monsalve[2,3], Juan Reutter[2,3], and Francisco Vial-Prado[2,3]

[1] Laboratoire de Mathématiques de Versailles, UVSQ, Université Paris-Saclay, France
[2] DCC, Pontificia Universidad Católica de Chile, Santiago de Chile
[3] IMFD Chile www.imfd.cl

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. Using a novel approach, Goubin and Vial-Prado (2016) take advantage of the Multikey FHE-NTRU encryption scheme to establish decryption rights at key-generation time, thus preventing leakage of all secrets involved (even by powerful key-holders). Their algorithms are intended for two parties, and can be composed to form chains of users with inherited decryption rights. In this article, we provide new protocols for generating *Excalibur* keys under any DAG-like hierarchy, 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.

1 Introduction

In some public-key cryptography applications, parties own decryption rights over ciphertexts according to some hierarchic structure. For instance, in a mail redirection scenario it may be required that Alice is able to decrypt all of Bob's ciphertexts, and not conversely. 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 [4] proposed two-party computation protocols that securely perform a key generation procedure of the celebrated NTRU-based Multikey-FHE [8]. 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 [8]. 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 re-encryption scheme, addressing a re-encryption paradigm in fully homomorphic encryption scenarios, as pointed out in [4].

In this article we extend these key-generation protocols and provide multi-party computation protocols in the advised cyclotomic polynomial ring of [9] 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 MPC applications, our protocols require the composition of several routines. In all generality, this poses additional security restrictions, as a composition of two secure protocols is not necessarily secure (some examples are given in [3]). This has been the subject of extensive research, and we highlight the Universal Composability (UC) framework proposed by Cannetti [2] in which any two UC-secure protocols may be arbitrarily composed ensuring the inheritance of security. In our case, only non-concurrent composition is performed, and as pointed out in [3], security is proven directly with simulation-based proofs.

Our contributions. This article provides new protocols that extend those from [4], 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 protocols 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 [4]. 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 OT 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 [1]). We are confident that there is much to optimize in these procedures, opening another interesting angle for future work.

Overview of the article. We begin in section 2 by revisiting necessary concepts to construct our protocols. 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

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 $\mathbb{Z}_q$. The indicator function of a set S is any function that outputs 1 if the preimage is in S and 0 otherwise. For a distribution χ that samples from some set S, $e \leftarrow \chi$ means that e is sampled from S with the distribution χ, and $e \xleftarrow{\$} S$ means that e is sampled from S according to the uniform distribution. We denote tuples of elements with bold letters. For n a power of 2, let $R_q \stackrel{\text{def}}{=} \mathbb{Z}_q[X]/(X^n+1)$ be the cyclotomic ring of polynomials modulo X^n+1 and coefficients in $\mathbb{Z}_q$.

Invertible bounded Gaussian distributions in the quotient ring. We first point out that the ring $R_q = \mathbb{Z}_q[X]/(X^n+1)$ is not a unique factorization domain as X^n+1 is generally not irreducible in $\mathbb{Z}_q$. Units of this ring with small coefficients are used as NTRU secret keys. In fact, for $q = 1 \bmod 2n$ as in the advised modifications by Stehlé and Steinfeld [9], X^n+1 splits into n linear factors in $\mathbb{Z}_q$, ensuring a large key space. See [9,4] for more details. In the following definition, for a real number $r > 0$, let Γ_r be the Gaussian distribution on $\mathbb{R}^n$ centered about 0 and with standard-deviation r.

Definition 1 (Bounded Discrete Gaussian distribution over R_q). *For a real number $0 < B \ll q$, 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:*

1. *Sample vector $\mathbf{x} \leftarrow \Gamma_B$, and restart if $||\mathbf{x}||_\infty > B$.*
2. *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.

FHE-NTRU encryption and the multikey property. The Multikey FHE scheme presented in [8] uses a modified version of NTRU (N^{th}-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 $q = 1 \bmod 2n$ 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 $\mathsf{sk} \leftarrow 2f+1$ until sk is invertible. Sample $g \leftarrow \mathcal{G}_B^\times$ and define $\mathsf{pk} \leftarrow 2g \cdot \mathsf{sk}^{-1}$.

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

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

The linearity of the decryption equation allowed authors of [8] 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. As noted in [4], this linearity also implies that a secret key with small extra multiplicative factors (such as other secret keys) is able to correctly decrypt, i.e. a polynomial multiplication in R_q of a small number of secret keys acts as a regular key and inherits the decryption rights of all its factors. For two parties in [4], the R_q product of secret keys is performed in a secure MPC fashion in order to attain the Excalibur property.

3 Security definitions

We present the definitions we use in showing our key-generating protocols are secure. We distinguish two adversarial settings: the *semi-honest* case in which players cooperate with the execution of the protocol and sample from the correct distributions, but may collude and try to learn secrets from their shared views; and the *malicious* case, in which adversaries are not guaranteed to follow the protocol. In this paper, we only consider semi-honest adversaries, leaving the malicious case for future work.

3.1 Simulation-based MPC security against semi-honest adversaries

Fix a set of $P = \{P_1, \ldots, P_k\}$ of parties. Following e.g. [7], our notion of security is based on the idea of emulating *functionalities*, which are k-ary functions $f : (\{0,1\}^*)^k \to (\{0,1\}^*)^k$. To describe a functionality f we usually write $f = (f_1, \ldots, f_k)$, where each f_i is a random k-ary function that outputs a string.

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 $\text{view}_i^\pi(\mathbf{x}, \lambda)$, 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 $\text{view}_S^\pi(\mathbf{x}, \lambda)$ as the concatenation of each tuple $\text{view}_i^\pi(\mathbf{x}, \lambda)$. 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 $\text{output}^\pi(\mathbf{x}, \lambda)$.

Definition 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*

$$(\mathcal{I}_S(1^\lambda, \mathbf{x}_S, f_S(\mathbf{x}, \lambda)), f(\mathbf{x}, \lambda)) \overset{s}{\approx} (\mathit{view}_S^\pi(\mathbf{x}, \lambda), \mathit{output}^\pi(\mathbf{x}, \lambda))$$

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 $\phi_1, \ldots, \phi_\ell$, and let $\rho^{\pi_1, \ldots, \pi_\ell}$ be a protocol computing a functionality g that makes use of $\pi_1, \ldots, \pi_\ell$ 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. Using the ideas in e.g. [7], we can show the following.

Proposition 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.*

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 all of the machinery proposed in [2]. We do highlight that the restriction of Canetti's framework into our scenario yields security requirements equivalent to that of Definition 2 (see [3], §5.2).

4 Hardness assumptions

Security of our protocols against semi-honest adversaries is based on two well-known assumptions (RLWE and DSPR), and the difficulty of new factorization problems in R_q that extend those from [4].

Definition 3 (Decisional Small Polynomial Ratio assumption, from [9]). *For some parameters q, n, B, it is computationally hard to distinguish between the following two distributions over R_q: (1) A polynomial $\mathsf{pk} = 2g(2f+1)^{-1} \in R_q$ where $f, g \leftarrow \mathcal{G}_B$, and (2) a uniformly random polynomial $u \xleftarrow{\$} R_q$.*

Definition 4 (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 5 (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 UFD, 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. 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 [4].

Definition 6 (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 7 (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 [4], 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 [10]. Moreover, as in [4] 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

Our key-generating protocols assume a set of participants $P = \{P_1, \ldots, P_k\}$, and the objective is to 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.

We assume the existence of a protocol SP_m for computing a certain scalar product, that works as follows. Party A holds m bits $\boldsymbol{b} = (b_1, \ldots, b_m)$ and party B holds two vectors $\boldsymbol{r}^{(0)} = (r_1^{(0)}, \ldots, r_m^{(0)})$ and $\boldsymbol{r}^{(1)} = (r_1^{(1)}, \ldots, r_m^{(1)})$. At the end of the protocol, A learns $\sum_{i=1}^{m} r_i^{(b_i)}$ and B 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 [6]. It is outlined in appendix A.

5.1 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.

Our *k-Multiplication Protocol* is a nontrivial extension of algorithm 2-MP, from [4]. We need this algorithm for defining our protocols, so we recall it bellow.

Algorithm 1 Two-party R_q multiplication 2-MP, from [4].

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 $\sum_{i=1}^m \hat{e}_i = x_1 \cdot x_2 + r_2$.

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 2 contains the detail of this protocol, and Algorithm 1 provides the base case. The idea is to use (k-1)-MP to perform a secure multiplication of

Algorithm 2 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 $\prod_{\ell=1}^{k} x_\ell + \sum_{\ell=2}^{k} r_\ell$.

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: 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 $s_{\ell i}^b = r_{\ell i} + \hat{r}_{\ell i}^b$.

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

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

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$(v_j \cdot x_2, (x_3, s_{3i}^j), \ldots, (x_k, s_{ki}^j))$. P_2 learns $v_j \cdot \prod_{\ell=2}^{k} x_\ell + \sum_{\ell=3}^{k} s_{\ell i}^j$.

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

10: With a $\binom{2}{1}$-OT protocol, player P_1 extracts e_b from P_2. Note that $e_b = x_{1i} \cdot \prod_{\ell=2}^{k} x_\ell + \sum_{\ell=2}^{k} s_{\ell i}^b$.

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

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

13: P_1 extracts $\hat{s}_\ell = \sum_{i=1}^m \hat{r}_{\ell i}^{b_i}$ from P_ℓ with $SP(\boldsymbol{b}, (\boldsymbol{r}_\ell^0, \boldsymbol{r}_\ell^1))$, where $\boldsymbol{b} = (b_1, \ldots, b_m)$ and $\boldsymbol{r}_\ell^j = (\hat{r}_{\ell 1}^j, \ldots, \hat{r}_{\ell m}^j)$.

14: Finally, P_1 computes $\theta - \sum_{\ell=2}^{k} \hat{s}_\ell = \prod_{\ell=1}^{k} x_\ell + \sum_{\ell=2}^{k} r_\ell$.

all but participant's P_1 ring elements (see step 8). In turn, the multiplication for participant P_1 is carefully masked with additive uniform noise in order to avoid input leaking. In the end (Step 12), P_1 performs SP_m with each other participant with the goal of cancelling noise.

***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 3. 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).

5.2 Excalibur key generation protocols

In out key-generating protocols, players $P_2, \ldots, P_k$ 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, participant P_1 learns the secret key $\mathsf{sk}_1 = \alpha \prod_{i=2}^{k} \beta_i$, as well as

Algorithm 3 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 $\sum_{j=1}^{k} \pi_j = (\sum_{j=1}^{k} x_j) \cdot (\sum_{j=i}^{k} y_j)$.

procedure k-sMP
  Each P_i samples $R_i = \{r_{ij} \xleftarrow{R} R_q \mid j = [1,k] \wedge i \neq j\}$.
  for $i = 1, \ldots, k$ **do**
    for $j = 1, \ldots, k$, $j \neq i$ **do**
      P_i, P_j perform 2-MP$(x_i, (y_j, r_{ji}))$. Thus P_i learns $u_{ij} = x_i \cdot y_j + r_{ji}$
  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 4 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.

procedure $\mathsf{Exc}_{\mathsf{pk}}$
  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 $g = \sum_{i=1}^{k} g_i$.
  All participants perform (k)-MP$(r_1, (\beta_2, t_{21}), \ldots, (\beta_k, t_{k1}))$. Thus,
  P_1 learns $r'_1 = r_1 \cdot \prod_{i=2}^{k} \beta_i + \sum_{i=2}^{k} t_{i1}$.
  for $i = 2, \ldots, k$ **do**
    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 $r'_i = u_i - \sum_{j=1, j\neq i}^{k} t_{ij}$.
  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.
  P_2 checks: if z is not invertible in R_q, restart the protocol.
  P_2 computes $2w(z\beta_2)^{-1} = 2g(\alpha \prod_{j=2}^{k} \beta_j)^{-1}$ and publishes it as pk_1.

its public key pk_1. On the other hand, all other participants only learn pk_1. As advised in [4], parties generate the public key first, and P_1 commits to it.

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 4. It begins with participants sampling a gaussian share g_i, and random elements r_i, t_{ij} used to additively mask polynomials, as in protocol 3. 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).

Algorithm 5 Excalibur Secret Key Generation

Require: Let $\alpha = 2(\sum_{i=1}^{k} s_i) + 1$ be the same additive share as in protocol 4: 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 $r_{ij} \xleftarrow{R} R_q$, with $j = 2, \ldots, k$ and $j \neq i$. 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_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 $\mathsf{sk}_1 := ((2s_1 + 1) \prod_{i=1}^{k} \beta_i) + \sum_{i=2}^{k} \left(2s_i \prod_{j=2}^{k} \beta_j\right) = \alpha \prod_{i=2}^{k} \beta_i \in R_q$

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 5, and again it uses our multiplication protocols together with carefully selected random noise.

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 P' 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 multikey property. The following proposition is proven on appendix C.1.

Proposition 2. *Let $P' \subsetneq \{P_1, \ldots, P_k\}$ be a set of colluded parties. The problem of extracting α, g, r, r' or any secret key sk_j of a party $P_j \notin P'$ from public values, views of the protocol and secret keys of parties in P' 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.*

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.

Definition 8. *Consider the following functionalities. All variables are in R_q:*

$$\begin{array}{ll}
\mathcal{F}^{\mathsf{k\text{-}MP}}: & (x_1,(x_2,r_2),\ldots,(x_k,r_k)) \mapsto ((\prod_{i=1}^{k} x_i + \sum_{i=2}^{k} r_i), _, _, \ldots, _), \\
\mathcal{F}^{\mathsf{k\text{-}sMP}}: & ((x_1,y_2),\ldots,(x_k,y_k)) \mapsto (\pi_1,\ldots,\pi_k) \text{ where } \sum_i \pi_i = (\sum_i x_i)\cdot(\sum_i y_i); \\
\mathcal{F}^{\mathsf{Exc_{pk}}}: & (s_1,(\beta_2,s_2),\ldots,(\beta_k,s_k)) \mapsto (\mathsf{pk}_1,\mathsf{pk}_1,\cdots,\mathsf{pk}_1); \\
\mathcal{F}^{\mathsf{Exc_{sk}}}: & (s_1,(\beta_2,s_2),\ldots,(\beta_k,s_k)) \mapsto (\alpha \prod_{j=2}^{k} \beta_j, _, \cdots, _).
\end{array}$$

Proposition 3. *For $\rho \in \{\mathsf{k\text{-}MP}, \mathsf{k\text{-}sMP}, \mathsf{Exc_{pk}}, \mathsf{Exc_{sk}}\}$, ρ securely computes $\mathcal{F}^{\rho}$.*

The proof of this result relies heavily on Proposition 1, as we need to show first that k-MP securely computes $\mathcal{F}^{\mathsf{k\text{-}MP}}$, then use this result to show that k-sMP securely computes $\mathcal{F}^{\mathsf{k\text{-}sMP}}$, and so forth. The proof of k-MP is also interesting because we need to perform an induction on k. In [4], 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 $\rho = \mathsf{2\text{-}MP}$ here, and the complete proof proposition 3 is deferred to appendix C.

Proposition 4 (Simulation-based proof of [4], §7.1). *The protocol* 2-MP *securely computes $\mathcal{F}^{\mathsf{2\text{-}MP}}$.*

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 [4], §7.1.

As in section 3.1 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 2, for every possible set S we construct a PPT algorithm $\mathcal{I}_S$ such that

$$(\mathcal{I}_S(1^\lambda, \mathbf{x}_S, \mathcal{F}_S^{\mathsf{2\text{-}MP}}(\mathbf{x})), \mathcal{F}^{\mathsf{2\text{-}MP}}(\mathbf{x})) \overset{\mathrm{s}}{\approx} (\mathrm{view}_S^{\mathsf{2\text{-}MP}}(\mathbf{x},\lambda), \mathrm{output}^{\mathsf{2\text{-}MP}}(\mathbf{x},\lambda)).$$

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

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

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 $\sum_{i=1}^{m} \hat{e}_i = x_1 \cdot x_2 + r_2$.

We define $\mathcal{I}_S$, a simulator of the view of P_1, in algorithm 6. Its output is

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

Recall that $\mathcal{F}^{\mathsf{2\text{-}MP}}(\mathbf{x})$ and $\mathrm{output}^{\mathsf{2\text{-}MP}}(\mathbf{x},\lambda)$ are both equal to $x_1 \cdot x_2 + r_2$. Therefore, we only need to verify that $\mathcal{I}_S(1^\lambda, \mathbf{x}_S, \mathcal{F}_S^2(\mathbf{x})) \overset{\mathrm{s}}{\approx} \mathrm{view}_S^{\mathsf{2\text{-}MP}}(\mathbf{x},\lambda)$.

Algorithm 6 Simulator for 2-MP corresponding to $S = \{P_1\}$

Require: 1^λ, x_1, $x_1 \cdot x_2 + r_2$

procedure $\mathcal{I}_{P_1}$
- Sample m random 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 $\tilde{v}_{\tilde{b}_i} = x_{\mathcal{I}i}$.
- Sample m random polynomials $(\tilde{e}_i \xleftarrow{\$} R_q)_{i=1}^m$, 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.

First, both views share x_1. The polynomials $x_{11}, x_{12}, \dots, x_{1m}$ are uniformly generated by P_1 in 2-MP. On the other hand, $\tilde{x}_{11}, \tilde{x}_{12}, \dots, \tilde{x}_{1m}$ 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 $\text{view}_S^{\text{2-MP}}(\mathbf{x}, \lambda)$ and $\mathcal{I}_S(1^\lambda, \mathbf{x}_S, y_1)$ are statistically indistinguishable when $S = \{P_1\}$.

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

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

Algorithm 7 Simulator for 2-MP corresponding to $S = \{P_2\}$

Require: 1^λ, (x_2, r_2).

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

We define $\mathcal{I}_S$ in algorithm 7. Note that $\mathcal{F}_{P_2}^{\text{2-MP}}(x_1, (x_2, r_2))$ is empty. The output of $\mathcal{I}_S$ is:

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

Analogously as before, is it clear that $\mathcal{I}_S(1^\lambda, \mathbf{x}_S) \overset{\text{s}}{\approx} \text{view}_S^{\text{2-MP}}(\mathbf{x}, \lambda)$. □

6.3 Parameters and efficiency

The parameters n, q, B control the semantic (and multikey-homomorphic) security of the underlying NTRU encryption scheme, and the hardness of our new

problems in R_q of section 4. We consider them fixed and according to the suggested values in [9,4,8] 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 (see appendix B for detailed computations). 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, [1] reports computation of 700,000 $\binom{2}{1}$-OT per second over Wi-Fi, and [5] 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 [9]).

7 Conclusion

Our paper extends the original Excalibur key-generation protocols for an arbitrary hierarchy of keys, and presents formal 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.

The problem of key-generation in the presence of malicious adversaries is an interesting direction for future work. In particular, we note that this case is not immediate form our results, as Definition 2 and Proposition 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).

Acknowledgements

We would like to thank the anonymous reviewers for their comments. This work was supported by Instituto Milenio Fundamentos de los Datos, Vicuña Mackenna 4860, Santiago, Chile, and Fondecyt Chile (project number 1170866). The fourth author would like to thank Claudio Orlandi for his insight and for providing references on simulation-based proofs, and Martín Ugarte for his helpful comments.

References

1. Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. More efficient oblivious transfer and extensions for faster secure computation, 11 2013.
2. Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Foundations of Computer Science, 2001. Proceedings. 42nd IEEE Symposium on*, pages 136–145. IEEE, 2001.
3. Ran Canetti. Security and composition of cryptographic protocols: A tutorial (part i). *SIGACT News*, 37(3):67–92, September 2006.
4. Louis Goubin and Francisco José Vial Prado. Blending FHE-NTRU keys – the excalibur property. In *Progress in Cryptology – INDOCRYPT 2016*. Springer International Publishing, 2016.
5. Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending oblivious transfers efficiently, 06 2003.
6. Shun-Dong Li and Yi-Qi Dai. Secure two-party computational geometry. *Journal of Computer Science and Technology*, 20(2):258–263, Mar 2005.
7. Yehuda Lindell. How to simulate it–a tutorial on the simulation proof technique. In *Tutorials on the Foundations of Cryptography*, pages 277–346. Springer, 2017.
8. Adriana López-Alt, Eran Tromer, and Vinod Vaikuntanathan. 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*, STOC '12, pages 1219–1234, New York, NY, USA, 2012. ACM.
9. Damien Stehlé and Ron Steinfeld. Making ntru as secure as worst-case problems over ideal lattices. In *EUROCRYPT*, 2011.
10. Enrico Thomae and Christopher Wolf. Solving underdetermined systems of multivariate quadratic equations revisited. In Marc Fischlin, Johannes Buchmann, and Mark Manulis, editors, *Public Key Cryptography – PKC 2012*, pages 156–171, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

A Scalar product protocol SP_m

In our k-Multiplication protocol (algorithm 2), parties rely on a multiparty scalar product protocol as a subroutine to cancel additive noise.

Definition 9. *For $m \in \mathbb{N}$, let SP_m be a two-party protocol performing the following. Party A has a sequence of bits ordered in a binary vector $\boldsymbol{b} = (b_1, \ldots, b_m)$. For each $i = 1, \ldots, m$, party B has a pair of polynomials $(p_i^{(0)}, p_i^{(1)})$ of R_q. In the end, party A learns $\gamma = p_1^{(b_1)} + p_2^{(b_2)} + \cdots + p_m^{(b_m)}$ and nothing more. Party B learns nothing.*

We refer to this functionality as a *scalar product*[4], since it computes

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

where $\boldsymbol{b}^c = (\bar{b}_1, \ldots, \bar{b}_m)$ is the binary complement of $\boldsymbol{b}$. The protocol is outlined in algorithm 8 below.

Algorithm 8 Scalar product protocol

Require: Alice holds $\boldsymbol{b} = (b_1, \ldots, b_m) \in \{0,1\}^m$. Bob 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: Alice learns $(p_1^0, \ldots, p_m^0) \cdot \boldsymbol{b}^c + (p_1^1, \ldots, p_m^1) \cdot \boldsymbol{b}$, Bob learns nothing.

1: **procedure** SP_m
2: Alice samples κ vectors $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_\kappa \xleftarrow{\$} \mathbb{Z}^m$ such that $\boldsymbol{b}_1 + \cdots + \boldsymbol{b}_\kappa = \boldsymbol{b}$.
3: **for** $i = 1 \ldots \kappa$ **do**
4: Alice samples a bit σ and two vectors $\boldsymbol{a}_0, \boldsymbol{a}_1 \xleftarrow{\$} \{0,1\}^m$. She sets $a_\sigma \leftarrow \boldsymbol{b}_i$.
5: Alice sends the pair $(\boldsymbol{a}_1, \boldsymbol{a}_2)$ to Bob.
6: Bob computes

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

7: With a $\binom{2}{1}$-OT protocol, Alice extracts $\gamma_i := d_\sigma$ from Bob.
8: Alice computes $\gamma = \sum_{i=1}^{\kappa} \gamma_i = (p_1^0, \ldots, p_m^0) \cdot \boldsymbol{b}^c + (p_1^1, \ldots, p_m^1) \cdot \boldsymbol{b}$.

Remark: This protocol can be restated as a $\binom{2^m}{1}$-OT protocol, as follows. For each $x \in \{0,1\}^m$, party B 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 A extracts the polynomial corresponding to $x' = \boldsymbol{b}$ with a $\binom{2^m}{1}$-OT protocol. We point out that this is highly inefficient, because B needs to compute $O(2^m)$ additions in R_q.

[4] In the vector space R_q^m. Recall that $R_q \simeq \mathbb{F}_{q^n}$, the field of characteristic q^n

B Algorithmic complexity

In this appendix we develop expressions for the computational complexity of our key generation protocols. In this section, 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 [8]).

Definition 10. *Let θ (resp. π) be the computational cost of performing a $\binom{2}{1}$-OT protocol (resp. performing a multiplication in R_q).*

Proposition 5. *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 8 from appendix A 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 2, 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 3), therefore we have

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

□

Proposition 6. *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 4), parties perform one $\mathsf{k\text{-}MP}$ and $(k-1)$ instances of $(\mathsf{k\text{-}1})\text{-}\mathsf{MP}$. Also, in $\mathsf{Exc}_{\mathsf{sk}}$ (algorithm 5) parties perform $(k-1)$ instances of $(\mathsf{k\text{-}1})\text{-}\mathsf{MP}$ and one final $\mathsf{k\text{-}MP}$. 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.

C Security proofs

C.1 Proof of proposition 2

Recall that the output of the proposed protocol is a key-pair of the form

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

where for $i = 2, \ldots, k$, $(\mathsf{sk}_i, \mathsf{pk}_i) \leftarrow \mathsf{NTRU\text{-}Keygen}()$ and $\alpha = 2(f_1 + \cdots + f_k) + 1$ for polynomials f_j sampled from $\mathcal{G}_B^\times$. 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

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

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.

Claim. 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. 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. 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. Extracting secret keys from the whole view and informationfrom 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. 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 $z' = r \prod_{i=2}^{k} \beta_i$. □

C.2 Proof of proposition 3

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

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

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

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

The elements x_{1i}, b_i, v_j^i and $\hat{e}_i$ are as in the proof of proposition 4. 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.

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

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

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

Note that the view of P_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 $\hat{r}^j_{\ell i}$ are uniformly random polynomials.

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

1. Case $P_1, P_2 \in S$

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

Require: 1^λ, $\mathbf{x}_S$, $\mathcal{F}^{\mathsf{k\text{-}MP}}_S(\mathbf{x})$.

procedure $\mathcal{I}^1_S$
  Generate m polynomials $(\tilde{x}_{1i} \xleftarrow{\$} R_q)^m_{i=1}$, such that $\sum^m_{i=1} \tilde{x}_{1i} = x_1$.
  for $i = 1 \ldots m$ **do**
    Sample $\tilde{b}_i \xleftarrow{\$} \{0,1\}$ and $(\tilde{v}^i_0, \tilde{v}^i_1) \xleftarrow{\$} R^2_q$. Set $\tilde{v}^i_{b_i} = \tilde{x}_{1i}$.
  Generate m polynomials $(\tilde{r}_{2i} \xleftarrow{\$} R_q)^m_{i=1}$, such that $\sum^m_{i=1} \tilde{r}_{2i} = r_2$.
  Generate $2 \cdot m$ polynomials $(\tilde{r}^0_{2i}, \tilde{r}^1_{2i} \xleftarrow{\$} R^2_q)^m_{i=1}$ and compute $\tilde{s}_2 = \sum^m_{i=1} \tilde{r}^{\tilde{b}_i}_{2i}$.
  Compute $(\tilde{v}^i_0 \cdot x_2, \tilde{v}^i_1 \cdot x_2)^m_{i=1}$.
  for each $P_\ell \in S \wedge i = \ell > 2$ **do**
    Generate m polynomials $(\tilde{r}_{\ell i} \xleftarrow{\$} R_q)^m_{i=1}$, such that $r_\ell = \sum^m_{i=1} \tilde{r}_{\ell i}$.
    Generate $2 \cdot m$ polynomials $(\tilde{r}^0_{\ell i}, \tilde{r}^1_{\ell i} \xleftarrow{\$} R^2_q)^m_{i=1}$ and compute $\tilde{s}_\ell = \sum^m_{i=1} \tilde{r}^{\tilde{b}_i}_{\ell i}$.
  Generate $2 \cdot m$ polynomials $(\tilde{e}^i_0, \tilde{e}^i_1 \xleftarrow{\$} R_q)^m_{i=1}$ and compute $\tilde{\theta} = \sum^m_{i=1} \tilde{e}^i_{b_i}$.
  For each $P_\ell \notin S$, generate $\tilde{s}_\ell \xleftarrow{\$} R_q$, such that $\tilde{\theta} - \sum^k_{\ell=2} \tilde{s}_\ell = \mathcal{F}^{\mathsf{k\text{-}MP}}_{P_1}(\mathbf{x})$.
  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 $\mathcal{I}^1_S$:

$$\mathcal{I}^1_S(\mathbf{x}, \lambda) = \begin{cases} x_1, \\ \tilde{x}_{11}, \tilde{x}_{12}, \ldots, \tilde{x}_{1m}, \\ (\tilde{b}_1, \tilde{v}^1_0, \tilde{v}^1_1, \tilde{e}^1_1), \cdots, (\tilde{b}_m, \tilde{v}^m_0, \tilde{v}^1_1, \tilde{e}^m_m), \\ \tilde{\theta}, \tilde{s}_1, \ldots, \tilde{s}_k \\ \\ x_2, r_2, \\ \tilde{r}_{21}, \tilde{r}_{22}, \ldots, \tilde{r}_{2m}, \\ (\tilde{r}^0_{21}, \ldots, \tilde{r}^0_{2m}), (\tilde{r}^1_{21}, \ldots, \tilde{r}^1_{2m}), \\ (\tilde{v}^1_0, \tilde{v}^1_1, \tilde{e}^1_0, \tilde{e}^1_1), \cdots, (\tilde{v}^m_0, \tilde{v}^m_1, \tilde{e}^m_0, \tilde{e}^m_1), \\ \\ \text{for } P_\ell \in S \backslash \{P_1, P_2\} : \\ \quad x_\ell, r_\ell, \tilde{r}_{\ell 1}, \ldots, \tilde{r}_{\ell m}, \\ \quad (\tilde{r}^0_{\ell 1}, \ldots, \tilde{r}^0_{\ell m}), (\tilde{r}^1_{\ell 1}, \ldots, \tilde{r}^1_{\ell m}) \end{cases}$$

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 $\mathcal{I}_S(1^\lambda, \mathbf{x}_S, \mathcal{F}_S^k(\mathbf{x})) \overset{\text{s}}{\approx} \text{view}_S^{\mathsf{k\text{-}MP}}(\mathbf{x}, \lambda)$, 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.

2. Case $P_1 \notin S, P_2 \in S$

Algorithm 10 Simulator $\mathcal{I}_S^2$ for k-MP

Require: 1^λ, $\mathbf{x}_S$, $\mathcal{F}_S^k(\mathbf{x})$.

procedure $\mathcal{I}_S^2$
  Generate m polynomials $(\tilde{r}_{2i} \overset{\$}{\leftarrow} R_q)_{i=1}^m$, such that $\sum_{i=1}^m \tilde{r}_{2i} = r_2$.
  Generate $2m$ polynomials $(\tilde{r}_{2i}^0, \tilde{r}_{2i}^1 \overset{\$}{\leftarrow} R_q^2)_{i=1}^m$.
  Generate polynomials $(\tilde{v}_0^i, \tilde{v}_1^i \overset{\$}{\leftarrow} R_q^2)_{i=1}^m$.
  for each $P_\ell \in S \wedge \ell > 2$ **do**
    Generate m polynomials $(\tilde{r}_{\ell i} \overset{\$}{\leftarrow} R_q)_{i=1}^m$, such that $r_\ell = \sum_{i=1}^m \tilde{r}_{\ell i}$.
    Generate $2 \cdot m$ polynomials $(\tilde{r}_{\ell i}^0, \tilde{r}_{\ell i}^1 \overset{\$}{\leftarrow} R_q^2)_{i=1}^m$.
  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 $(\tilde{e}_0^i, \tilde{e}_0^i \overset{\$}{\leftarrow} R_q)_{i=1}^m$.
  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:

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

3. Case $P_1 \in S, P_2 \notin S$

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

Require: 1^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 $\tilde{v}_{\tilde{b}_i}^i = \tilde{x}_{1i}$.
  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 $\tilde{\theta} = \sum_{i=1}^m \tilde{e}_{\tilde{b}_i}^i$.
  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:

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

4. Case $P_1 \notin S$ and $P_2 \notin S$

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

Require: 1^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:

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

As before, we conclude that $\mathcal{I}_S(1^\lambda, \mathbf{x}_S, \mathcal{F}_S^k(\mathbf{x})) \overset{s}{\approx} \text{view}_S^{\text{k-MP}}(\mathbf{x}, \lambda)$ for all $S \subsetneq P$, which was to be shown. In conclusion, the protocol k-MP securely computes $\mathcal{F}^{\text{k-MP}}$. ■

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

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

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

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.

Algorithm 13 Simulator $\mathcal{I}_S$ for k-sMP

Require: 1^λ, $\mathbf{x}_S$, $\mathcal{F}_S^{\mathsf{k\text{-}sMP}}(\mathbf{x})$.

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.

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

$$\mathcal{I}_S\big(1^\lambda, \mathbf{x}_S, \mathcal{F}_S^{\mathsf{k\text{-}sMP}^{\mathsf{2\text{-}MP}\to\mathcal{F}^{\mathsf{2\text{-}MP}}}}(\mathbf{x},\lambda)\big) \overset{s}{\approx} \text{view}_S^{\mathsf{k\text{-}sMP}^{\mathsf{2\text{-}MP}\to\mathcal{F}^{\mathsf{2\text{-}MP}}}}(\mathbf{x},\lambda)$$

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.

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 $\mathcal{F}^{\mathsf{k\text{-}sMP}}$. ■

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

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

- View of P_1

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

  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 $\mathcal{F}^{\mathsf{k\text{-}MP}}$, which computes $r_1 \prod_{i=2}^{k} \beta_i$ plus some random values. The pair z_1, w_1 are the outputs of $\mathcal{F}^{\mathsf{k\text{-}sMP}}$ shared functionality to compute $z = \alpha \cdot r'$ and $w = g \cdot r$, respectively.
- View of P_i, with $i \neq 1, 2$

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

  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 $\mathcal{F}^{\mathsf{k\text{-}1\text{-}MP}}$ 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

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

  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 $\mathsf{Exc_{pk}}$ in the simulator algorithm 14. The output of $\mathcal{I}_S$ is explained as follows:
  - If $P_1 \in S$, then

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

  - If $P_2 \in S$, then

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

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

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

  Since S is a strict subset of P, the elements in the view of S in $\mathsf{Exc_{pk}}$ 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 $\mathcal{I}_S(1^\lambda, \mathbf{x}_S, \mathcal{F}_S^{\mathsf{Exc_{sk}}}(\mathbf{x}))$ is indistinguishable from $\text{view}_S^{\mathsf{Exc_{sk}}}(\mathbf{x}, \lambda)$ for every S. ■

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

Require: 1^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.

$\mathsf{Exc}_{\mathsf{sk}}$: Analogously as before, we prove that $\mathsf{Exc}_{\mathsf{sk}}$ protocol, that uses $\mathcal{F}^{\mathsf{k\text{-}MP}}$ 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 $\mathsf{Exc}_{\mathsf{sk}}$ in the simulator algorithm 15. The output of $\mathcal{I}_S$ is:

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

Algorithm 15 Simulator for $\mathsf{Exc}_{\mathsf{sk}}$

Require: 1^λ, $\mathbf{x}_S$, $\mathcal{F}_S^{\mathsf{Exc}_{\mathsf{sk}}}(\mathbf{x})$.

procedure $\mathcal{I}_S$
  for $P_i \in S$ **do**
    Sample $\{\tilde{r}_{ij} \xleftarrow{\$} R_q \mid j = [1, k] \wedge i \neq j\}$ and $\tilde{u}_i \xleftarrow{\$} R_q$.
    Compute $\tilde{R}_i = \tilde{u}_i - \sum_{j=2, j\neq 1}^k \tilde{r}_{ij}$.
  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

$$\mathcal{I}_S(1^\lambda, \mathbf{x}_S, \mathcal{F}_S^{\mathsf{Exc}_{\mathsf{sk}}}(\mathbf{x})) \stackrel{\mathrm{s}}{\approx} \mathrm{view}_S^{\mathsf{Exc}_{\mathsf{sk}}}(\mathbf{x}, \lambda). \qquad \blacksquare$$