Bachelor's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Mathematical Sciences

Sander Støle Hageli

Post-quantum cryptography

Bachelor's thesis in Mathematical Sciences
Supervisor: Kristian Gjøsteen
December 2021

NTNU
Norwegian University of
Science and Technology

Sander Støle Hageli

Post-quantum cryptography

Bachelor's thesis in Mathematical Sciences
Supervisor: Kristian Gjøsteen
December 2021

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Mathematical Sciences

NTNU
Norwegian University of
Science and Technology

Abstract

In this bachelor's thesis, we are taking a look at CRYSTALS Kyber, a CCA2 quantum-resistance Key Encapsulation Mechanism, how it is constructed and its security. We are also looking at the Module Learning With Error Problem and how it relates to Kyber.

With quantum computers having the possibility of becoming powerful enough in the near future to efficiently factoring integers and finding solutions to the discrete logarithm problem, and by the fact that NIST made a call for defining a new standard for public-key encryption, there has been an increased interest in post-quantum cryptography.

One of the finalists to NIST's call for proposals are CRYSTALS Kyber: a CCA2-secure KEM, whose security is based on the hardness of solving the Module Learning With Error Problem. Kyber is the result of a slightly tweaked Fujisaki–Okamoto transform of a CPA-secure scheme called Kyber.CPA.

Kyber.CPA consists of three algorithms: a key generation, an encryption and a decryption algorithm. The outputs of these algorithms are compressed to make the public-key and ciphertext smaller with respect to the amount of bits required to store them.

Contents

1 Introduction

Many of the cryptosystems in use today base their security on hard mathematical problems such as integer factorization or the discrete logarithm problem. While there are no efficient methods to solve these problems with today's classical computers, a powerful enough quantum computer may solve these problems in polynomial time. This might be done using Shor's algorithm [1].

Today's quantum computers are not powerful enough to solve the problems mentioned above, but if they do one day become powerful enough, then it is advantageous to already have secure systems against them. This is why NIST, the U.S. National Institute of Standards and Technology, wanted to define a new post-quantum standard for public-key encryption [2]. They started a process to evaluate different quantum-resistant public-key algorithms. One of the most promising type of problems to base these algorithms on are lattice-based ones.

One of the finalists to NIST's call for proposals are CRYSTALS, the "Cryptographic Suite for Algebraic Lattices" [3]. CRYSTALS Kyber is their public-key Key Encapsulation Mechanism, which is IND-CCA2 secure, and its security is based on the hardness of solving the Module Learning With Error Problem. It is a generalization of the Learning With Error Problem, where instead of only restricting the problem to real numbers, vectors of polynomials in a quotient ring are used.

The construction of Kyber is built upon a CPA-secure scheme, called Kyber.CPA. The algorithms in Kyber.CPA are made to compress their output. This sacrifices some correctness, but it makes the public-key and ciphertext smaller with respect to the amount of bits required to store them. This error in correctness could be made very small by choosing the right parameters.

Kyber.CPA is transformed into Kyber via a slightly tweaked Fujisaki–Okamoto transform. Making it possible for two parties to share a secret-key between each other over a public network. The security of Kyber can be expressed in terms of the security of Kyber.CPA, both with classical and quantum adversaries, even though this expression changes from a classical to a quantum adversary.

There are reductions, in both directions, between the Module Learning With Error Problem and the Module Shortest Independent Vector Problem. One of these reductions is a quantum reduction, making the problems equally hard for a quantum computer to solve. It is believed that the latter of these problems is hard to solve, and so the Module Learning With Error Problem is also believed to be hard to solve.

2 Definitions and notations

Vector notation Here non-bold lowercase letters represent scalars or polynomials, bold lowercase letters represent vectors, while bold uppercase letters represent matrices. A vector $\mathbf{a}$ with elements from a set S will be written as $\mathbf{a} \in S^n$, while $n \times m$ matrices $\mathbf{A}$ with elements from a set S will be written as $\mathbf{A} \in M_{n \times m}(S)$. All vectors will be column vectors.

Randomly chosen notation When the notation $a \leftarrow S$ is used, it should be interpreted as a being chosen uniformly random from the set S. If $\mathbf{a}$ is a vector (or $\mathbf{A}$ is a matrix), then $\mathbf{a} \leftarrow S^n$ ($\mathbf{A} \leftarrow M_{n \times m}(S)$), denotes that all the elements in $\mathbf{a}$ ($\mathbf{A}$) are uniformly random chosen from S.

Groups and rings The ring R will be the quotient ring $\mathbb{Z}[X]/(X^n+1)$ of polynomials with integer coefficients. R_q will denote the quotient ring $\mathbb{Z}_q[X]/(X^n+1)$ of polynomial with coefficients from $\mathbb{Z}_q$, the group of integers modulo q.

Binomial distribution The set B_ν is the collection $\{\Sigma_{i=1}^{\nu}(a_i - b_i) \mid (a_i, b_i) \leftarrow \{0,1\}^2\}$. So when $u \leftarrow B_\nu$, it means that $u = \Sigma_{i=1}^{\nu}(a_i - b_i)$, where $(a_i, b_i) \leftarrow \{0,1\}^2$ for all i. If $u \in R$, then $u \leftarrow B_\nu$ means that each coefficient of u is chosen from B_ν.

Norm For an element $w = w_0 + w_1 X + w_2 X^2 + ... + w_{n-1} X^{n-1} \in R_q$, the norm will be defined as $\|w\|_\infty = \max_i(\|w_i\|_\infty)$, where all $w_i \in \mathbb{Z}_q$. The norm of an element $w \in \mathbb{Z}_q$ will be $\|w\|_\infty = |w \bmod^{\pm} q|$, where $w \bmod^{\pm} q$ denotes the integer w' in the range $(-\frac{q}{2}, \frac{q}{2}]$ if q is even, and $[-\frac{q-1}{2}, \frac{q-1}{2}]$ if q is odd, such that w' is congruent to w modulo q.

Rounding The notation $\lceil a \rfloor$ means to round a to the nearest integer. $\lceil \cdot \rceil$ is the ceiling function that rounds up, and $\lfloor \cdot \rfloor$ is the floor function that rounds down.

Cryptosystem A cryptosystem is a tuple consisting of three algorithms: A key generation algorithm, an encryption algorithm, and a decryption algorithm.

Correctness A cryptosystem is δ-correct if the decryption, with the secret-key, of the encryption, with the corresponding public-key, of a message return the same message in $\delta \cdot 100\%$ of the cases. If $\delta = 1$ then the cryptosystem is simply called correct.

XOF An extendable output function (XOF) is a hash function that maps a finite amount of bits into an infinite amount of bits. The output can be interpreted as bits describing a vector, or the output can be bits describing a matrix.

IND-CPA security In a public-key Indistinguishable Chosen Plaintext Attack (IND-CPA) secure setting, an adversary sends two messages to a challenger. The challenger only encrypts a random of these two messages and sends this back to the adversary. The cryptosystem is called IND-CPA secure if $Adv_{cpa}(\mathcal{A}) = |Pr(Adversary\ guessing\ correctly) - \frac{1}{2}|$ is negligible, with respect to a security parameter λ, for any efficient adversary $\mathcal{A}$.

IND-CCA2 security In a public-key Indistinguishable Chosen Ciphertext Attack 2 (IND-CCA2) secure setting, an adversary and challenger go through all the same steps as in the IND-CPA setting. In addition, the adversary may get the decryption of any ciphertext that is not the challenge ciphertext. The cryptosystem is called IND-CPA secure if $|Pr(Adversary\ guessing\ correctly) - \frac{1}{2}|$ is negligible with respect to a security parameter λ.

Random Oracle Model In a Random Oracle Model (ROM), when the oracle is asked for a query, it outputs a random element. If the oracle is asked for the same query again, the output will be the same random element.

3 Construction

When constructing a cryptosystem, the system will consist of a key generation, an encryption and a decryption algorithm. It is preferable that this system is as correct as possible, while also making it as secure and efficient as possible.

3.1 Compressing and Decompressing

In the key generation algorithm, the public-key will be compressed. This is to make the key size smaller with respect to the amount of bits required to store the information about the key. The output of the encryption and decryption algorithms will also be compressed to make their output smaller. For a prime q, an integer $x \in \mathbb{Z}_q$ and an integer $d < \lceil \log_2(q) \rceil$, the compression and decompression functions will be defined as follows:

$$Compress_q(x, d) = \left\lceil \frac{2^d}{q} x \right\rfloor \bmod 2^d \tag{1}$$

$$Decompress_q(x, d) = \left\lceil \frac{q}{2^d} x \right\rfloor \tag{2}$$

The compression function will take an element, x, from $\mathbb{Z}_q$ into $\mathbb{Z}_{2^d}$. Since $d < \lceil \log_2(q) \rceil$ and d being an integer, then $d < \log_2(q)$. So $\mathbb{Z}_{2^d}$ is a proper subset of $\mathbb{Z}_q$, for any prime q. The decompression function is intended as an inverse of the compression function. It takes an element, x, from $\mathbb{Z}_{2^d}$ into $\mathbb{Z}_q$, with d and q being the same as for the compression function. When applying the compress and decompress functions to elements in R_q or R_q^k, the functions are applied to each coefficient individually.

The compress function can be interpreted as scaling an $x \in \mathbb{Z}_q$ by some $c < 1$, making cx a decimal number. The rounding of cx will consequently remove the decimals of cx. When decompressing this compressed x, $\lceil cx \rfloor$ is scaled by the multiplicative inverse of c. This means that some information about x is lost. A parallel can by drawn from base-ten, where an integer can be represented as $x = a_0 1 + a_1 10 + 1_2 10^2 + ...$, when scaling x by $\frac{1}{10^d}$, it would result in $a_0 \frac{1}{10^d} + a_1 \frac{1}{10^{d-1}} + ... + a_{d-1} \frac{1}{10} + a_d 1 + a_{d+1} 10 + ...$ Rounding this number would give $a_d 1 + a_{d+1} 10 + ...$, and scaling it back results in $a_d 10^d + a_{d+1} 10^{d+1} + ...$ So, by scaling, rounding and scaling back, the d first digits of x is lost. While this is the case in base-ten, the d first digits are not lost in the same way when scaling by $\frac{2^d}{q}$.

The relation between the error of compressing and decompressing, and the values of d and q, is described by the following lemma.

Lemma 1. *Let q be a prime, $x \in \mathbb{Z}_q$ and d an integer such that $d < \lceil \log_2(q) \rceil$. Let $x' = Decompress_q(Compress_q(x, d), d)$. Then*

$$\left| x - x' \ mod^{\pm} q \right| \leq \left\lceil \frac{q}{2^{d+1}} \right\rfloor.$$

Proof. The rounding of any $y \in \mathbb{Q}$ can be written as $\lceil y \rfloor = y + c$, for a $c \in (-\frac{1}{2}, \frac{1}{2}]$, where c is the distance between y and the nearest integer. Then

$$\begin{aligned}
\left| x - \left\lceil \frac{q}{2^d} \left(\left\lceil \frac{2^d}{q} x \right\rfloor \bmod 2^d \right) \right\rfloor \bmod^{\pm} q \right| &= \left| x - \left\lceil \frac{q}{2^d} \left(\frac{2^d}{q} x + c \bmod 2^d \right) \right\rfloor \bmod^{\pm} q \right| \\
&= \left| x - \left\lceil \frac{q}{2^d} \left(\frac{2^d}{q} x + c - n 2^d \right) \right\rfloor \bmod^{\pm} q \right| \\
&= \left| x - \left\lceil x + \frac{cq}{2^d} - nq) \right\rfloor \bmod^{\pm} q \right| \\
&= \left| x - \left(x + \frac{cq}{2^d} - nq + e \right) \bmod^{\pm} q \right| \\
&= \left| \frac{cq}{2^d} - nq + e \bmod^{\pm} q \right| = \left| \frac{cq}{2^d} + e \bmod^{\pm} q \right| \\
&= \left| \frac{cq}{2^d} + e \right|.
\end{aligned}$$

Here, e is the difference between $x + \frac{cq}{2^d} - nq$ and the nearest integer, and since x and nq are integers, e will consequently be the difference between $\frac{cq}{2^d}$ and the nearest integer. This will result in

$$\left| \frac{cq}{2^d} + e \right| = \left\lceil \frac{cq}{2^d} \right\rfloor \leq \left\lceil \frac{q}{2^{d+1}} \right\rfloor$$

since $|c|$ is at most $\frac{1}{2}$. □

3.2 Algorithms

In the following algorithms, q will be a prime and the function f will be an extended output function, meaning that depending on the input, the output of the function may be on different forms, i.e., the output may be a vector, or a matrix.

The first algorithm defined will be the key generation algorithm, *Gen*, where d_t will be a positive integer less than $\lceil \log_2(q) \rceil$, and the function f is a XOF.

Key generation algorithm (1)
Input: (No input)

1: $s_1, s_2 \leftarrow \{0,1\}^{256}$
2: $\mathbf{A} = f(s_1) \in M_{k \times k}(R_q)$
3: $(\mathbf{s}, \mathbf{e}) = f(s_2) \in B_\nu^k \times B_\nu^k$
4: $\mathbf{t} = Compress_q(\mathbf{As} + \mathbf{e}, d_t)$
5: return $((\mathbf{t}, s_1), \mathbf{s})$

Here $(\mathbf{t}, s_1)$ is used as a public-key, while $\mathbf{s}$ is the private-key. The public-key consists of $\mathbf{t}$, which is a compression of $\mathbf{As} + \mathbf{e}$, and the seed s_1 for generating the matrix $\mathbf{A}$. It is more efficient to return the 256-bit seed value for the matrix instead of the whole $k \times k$ matrix.

The second algorithm will be the encryption algorithm, *Enc*. Let d_t be as above, and let d_u and d_v be positive integers less than $\lceil \log_2(q) \rceil$.

Encryption algorithm (2)
Input: (public-key=$(\mathbf{t}, s_1)$, message=m)

1: $r \leftarrow \{0,1\}^{256}$
2: $\mathbf{t} = Decompress_q(\mathbf{t}, d_t)$
3: $\mathbf{A} = f(s_1) \in M_{k \times k}(R_q)$
4: $(\mathbf{r}, \mathbf{e}_1, e_2) = f(r) \in B_\nu^k \times B_\nu^k \times B_\nu$
5: $\mathbf{u} = Compress_q(\mathbf{A}^T\mathbf{r} + \mathbf{e}_1, d_u)$
6: $v = Compress_q(\mathbf{t}^T\mathbf{r} + e_2 + \lceil \frac{q}{2} \rfloor m, d_v)$
7: return $(\mathbf{u}, v)$

The output $(\mathbf{u}, v)$ is the ciphertext.

In the decryption algorithm, *Dec*, as for *Enc*, let d_t, d_u and d_v be as above.

Decryption algorithm (3)
Input: (secret-key=$\mathbf{s}$, ciphertext=$(\mathbf{u}, v)$)

1: $\mathbf{u} = Decompress_q(\mathbf{u}, d_u)$
2: $v = Decompress_q(v, d_v)$
3: return $Compress_q(v - \mathbf{s}^T\mathbf{u}, 1)$

These three algorithms, $\Pi = (Gen, Enc, Dec)$, are the Kyber.CPA cryptosystem. Since the value d used in compressing is 1, the output of the decryption algorithm has $\mathbb{Z}_2[X]/(X^n+1)$ as its range. The following theorem will describe the correctness of this system. The proof for this theorem follows the proof in [4], but with additional elaboration.

Theorem 1. *The cryptosystem Π is $1-\epsilon$ correct, where*

$$\epsilon = Pr\left(\|\boldsymbol{e}^T\boldsymbol{r} + \boldsymbol{k}_t^T\boldsymbol{r} + e_2 + k_v - \boldsymbol{s}^T\boldsymbol{e}_1 - \boldsymbol{s}^T\boldsymbol{k}_u\|_\infty \geq \left\lceil \frac{q}{4} \right\rfloor\right)$$

for $\boldsymbol{e}, \boldsymbol{e}_1, \boldsymbol{r}, \boldsymbol{s} \leftarrow B_\nu^k$, $e_2 \leftarrow B_\nu$, $\boldsymbol{k}_t \leftarrow \Theta_{d_t}^k$, $\boldsymbol{k}_u \leftarrow \Theta_{d_u}^k$ and $k_v \leftarrow \Theta_{d_v}$. The distribution Θ_d is the error distribution $x - Decompress_q(Compress_q(x,d),d)$ $mod^{\pm} q$, from compressing and decompressing an uniformly random $x \leftarrow R_q$.

Proof. We want to compare the output of the decryption algorithm (3) to the original message. In the decryption algorithm, v is decompressed from a compressed $v' = \mathbf{t}^T\mathbf{r} + e_2 + \lceil \frac{q}{2} \rfloor m$. Then $v = Decompress_q(Compress_q(v')) = \mathbf{t}^T\mathbf{r} + e_2 + \lceil \frac{q}{2} \rfloor m + k_v$ for a $k_v \in R_q$ from the distribution Θ_{d_v}. The same can be done for $\mathbf{u}$ and $\mathbf{t}$:

$$\begin{aligned}\mathbf{u} &= Decompress_q(Compress_q(\mathbf{A}^T\mathbf{r} + \mathbf{e}_1)) = \mathbf{A}^T\mathbf{r} + \mathbf{e}_1 + \mathbf{k}_u \\ \mathbf{t} &= Decompress_q(Compress_q(\mathbf{A}\mathbf{s} + \mathbf{e})) = \mathbf{A}\mathbf{s} + \mathbf{e} + \mathbf{k}_t\end{aligned}$$

for some $\mathbf{k}_u, \mathbf{k}_t \in R_q^k$ from the distributions $\Theta_{d_u}^k$ and $\Theta_{d_t}^k$ respectively. Then $v = (\mathbf{A}\mathbf{s})^T\mathbf{r} + \mathbf{e}^T\mathbf{r} + \mathbf{k}_t^T\mathbf{r} + e_2 + \lceil \frac{q}{2} \rfloor m + k_v$, and $v - \mathbf{s}^T\mathbf{u} = \mathbf{e}^T\mathbf{r} + \mathbf{k}_t^T\mathbf{r} + e_2 + \lceil \frac{q}{2} \rfloor m + k_v - \mathbf{s}^T\mathbf{e}_1 - \mathbf{s}^T\mathbf{k}_u$, as $(\mathbf{A}\mathbf{s})^T\mathbf{r} = \mathbf{s}^T(\mathbf{A}^T\mathbf{r})$. Let w denote $\mathbf{e}^T\mathbf{r} + \mathbf{k}_t^T\mathbf{r} + e_2 + k_v - \mathbf{s}^T\mathbf{e}_1 - \mathbf{s}^T\mathbf{k}_u$, so the expression $v - \mathbf{s}^T\mathbf{u}$ becomes $w + \lceil \frac{q}{2} \rfloor m$, and let $m' = Compress_q(v - \mathbf{s}^T\mathbf{u}, 1)$. Then

$$\begin{aligned}\|v - \mathbf{s}^T\mathbf{u} - Decompress_q(m', 1)\|_\infty &= \|v - \mathbf{s}^T\mathbf{u} - \left\lceil \frac{q}{2} \right\rfloor m'\|_\infty \\ &= \|w + \left\lceil \frac{q}{2} \right\rfloor m - \left\lceil \frac{q}{2} \right\rfloor m'\|_\infty\end{aligned}$$

which is less than $\lceil \frac{q}{4} \rfloor$ by Lemma 1. With this inequality, the difference of m and m' can be determined:

$$\begin{aligned}\left\| \left\lceil \frac{q}{2} \right\rfloor (m - m') \right\|_\infty &= \|w - w + \left\lceil \frac{q}{2} \right\rfloor (m - m')\|_\infty \\ &\leq \|w + \left\lceil \frac{q}{2} \right\rfloor (m - m')\|_\infty + \|-w\|_\infty \\ &= \|w + \left\lceil \frac{q}{2} \right\rfloor (m - m')\|_\infty + \|w\|_\infty < 2\left\lceil \frac{q}{4} \right\rfloor\end{aligned}$$

if $\|w\|_\infty < \lceil \frac{q}{4} \rfloor$. Then the inequality becomes $\|\lceil \frac{q}{2} \rfloor (m - m')\|_\infty < 2\lceil \frac{q}{4} \rfloor$.

Since q is an odd integer, it can either be expressed as $q = 4k+1$ or $q = 4k+3$, for an integer k. In the case that $q = 4k+1$, then $\lceil \frac{q}{2} \rfloor = \lceil 2k + \frac{1}{2} \rfloor = 2k+1$ and $2\lceil \frac{q}{4} \rfloor = 2\lceil k + \frac{1}{4} \rfloor = 2k$. Then the norm of $\lceil \frac{q}{2} \rfloor (m - m')$ becomes:

$$\begin{aligned}
\left\| \left\lceil \frac{q}{2} \right\rfloor (m - m') \right\|_\infty &= \max_i \left(\left\| \left\lceil \frac{q}{2} \right\rfloor (m_i - m'_i) \right\|_\infty \right) \\
&= \left| \left\lceil \frac{q}{2} \right\rfloor (m_j - m'_j) \bmod^{\pm} q \right| \\
&= |(2k+1)(m_j - m'_j) \bmod^{\pm} 4k+1| \\
&= |(-2k)(m_j - m'_j)| = 2k|m_j - m'_j|
\end{aligned}$$

which is strictly less than $2\lceil \frac{q}{4} \rfloor = 2k$. The only integer that satisfies this inequality is 0, meaning that $m_j = m'_j$.

For the case that $q = 4k+3$, $\lceil \frac{q}{2} \rfloor = \lceil 2k + 1 + \frac{1}{2} \rfloor = 2k+2$ and $\lceil \frac{q}{4} \rfloor = \lceil k + \frac{3}{4} \rfloor = k+1$. Then the inequality becomes $(2k+2)|m_j - m'_j| < 2k+2$, which is, again, only possible for $|m_j - m'_j| = 0$. Meaning that the biggest difference between m and m' is zero, that is $m = m'$ for all odd integers q. Thus, by letting $\epsilon = Pr(\|w\|_\infty \geq \lceil \frac{q}{4} \rfloor)$, the cryptosystem will be correct $(1 - \epsilon) \cdot 100\%$ of the time. □

4 Security

The security of the cryptosystem Π is based on the hardness of solving the Module Learning With Error Problem, MLWE. The MLWE-problem is a generalization of the LWE-problem from working over $\mathbb{R}$ to working over a R-module $M \subseteq R^n$, for a ring R.

4.1 MLWE-problem

The decision LWE-problem is about distinguishing between $(\mathbf{a}_i, b_i) \leftarrow \mathbb{Z}_q^n \times \mathbb{T}$, where $\mathbb{T} = ([0,1) \subseteq \mathbb{R}, +_{\text{mod } 1})$, the group of reals under addition modulo 1, and $(\mathbf{a}_i, \frac{1}{q}\mathbf{a}_i^T\mathbf{s} + e_i)$, where $\mathbf{a}_i, \mathbf{s} \leftarrow \mathbb{Z}_q^n$ and e_i is chosen accordingly to some probability density function over $\mathbb{T}$, for $i = 1, ..., m$. $\mathbf{s}$ remains the same for all i. This problem can also be interpreted as distinguishing between $(\mathbf{A}, \mathbf{b})$ and $(\mathbf{A}, \frac{1}{q}\mathbf{A}\mathbf{s}+\mathbf{e})$, where $\mathbf{A}$ is the matrix obtained from setting $\mathbf{a}_i^T = (a_{i1}, a_{i2}, ..., a_{in})$ as the i-th row, $\mathbf{s} \leftarrow \mathbb{Z}_q^n$ and $\mathbf{e} = (e_1, e_2, ..., e_m)^T$, where every e_i is chosen the in the same way as above. The lattice comes in the sense of $\frac{1}{q}\mathbf{A}\mathbf{s}$ being the lattice point, and $\frac{1}{q}\mathbf{A}\mathbf{s} + \mathbf{e}$ being interpreted as a small circle about this lattice point with radius $\|\mathbf{e}\|$. The problem is to determine if a $\mathbf{b}$ is uniformly random, or on the form $\mathbf{b} = \frac{1}{q}\mathbf{A}\mathbf{s} + \mathbf{e}$.

This problem can be expanded into the Ring LWE-problem, RLWE, where $a_1, a_2, ..., a_m$ are ring elements of R_q. The problem becomes to distinguishing between (a_i, b_i) and $(a_i, (a_i \cdot s) + e_i)$, where s and e_i are small in the sense their norm is small. $a_i \cdot s$ is the ring multiplication between a_i and s. Since the ring $R_q = \mathbb{Z}_q[X]/(X^n+1)$, the ring multiplication $a_i \cdot s$, can be written as the matrix multiplication

$$\begin{pmatrix} a_{i0} & -a_{i(n-1)} & \cdots & -a_{i1} \\ a_{i1} & a_{i0} & \cdots & -a_{i2} \\ \vdots & \vdots & \ddots & \vdots \\ a_{i(n-1)} & a_{i(n-2)} & \cdots & a_{i0} \end{pmatrix} \begin{pmatrix} s_0 \\ s_1 \\ \vdots \\ s_{n-1} \end{pmatrix} = \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_{n-1} \end{pmatrix} = \mathbf{c}$$

where the coefficients are given by $a_i = a_{i0} + a_{i1}x + \ldots + a_{i(n-1)}x^{n-1}$ and $s = s_0 + s_1 x + \ldots + s_{n-1}x^{n-1}$, then $a_i \cdot s = (1, x, x^2, \ldots, x^{n-1})\mathbf{c}$. By denoting the matrix generated from the polynomial a_i as $\mathrm{Rot}(a_i)$ and writing the matrix $\mathbf{A} = (\mathrm{Rot}(a_1), \mathrm{Rot}(a_2), \ldots, \mathrm{Rot}(a_m))$, the problem can be expressed as distinguishing between uniformly random chosen $\mathbf{b}$, and $\mathbf{b} = \mathbf{A}\mathbf{s} + \mathbf{e}$.

The RLWE-problem can further be generalized into the Module LWE-problem. The MLWE-problem is to distinguishing between uniformly random chosen $(\mathbf{a}_i, b_i)$ and $(\mathbf{a}_i, \langle \mathbf{a}_i, \mathbf{s} \rangle + e_i)$, where all $\mathbf{a}_i$'s are random elements from R_q^d, which is to be treated as an R-module, $\mathbf{s}$ is the same small polynomial from R_q^d for all $i = 1, \ldots, m$, and e_i are small error polynomials from R_q. This problem can then be written as $(\mathbf{A}, \mathbf{A}\mathbf{s} + \mathbf{e})$, where the i-th row of $\mathbf{A}$ is $\mathbf{a}_i^T = (a_{i1}, a_{i2}, \ldots, a_{id})^T$. In the same way as for RLWE the polynomial multiplication can be represented by matrix multiplication, so the matrix $\mathbf{A}$ can be expressed as the $m \times d$ block matrix:

$$\begin{pmatrix} \mathrm{Rot}(a_{11}) & \mathrm{Rot}(a_{12}) & \cdots & \mathrm{Rot}(a_{1d}) \\ \mathrm{Rot}(a_{21}) & \mathrm{Rot}(a_{22}) & \cdots & \mathrm{Rot}(a_{2d}) \\ \vdots & \vdots & \ddots & \vdots \\ \mathrm{Rot}(a_{m1}) & \mathrm{Rot}(a_{m2}) & \cdots & \mathrm{Rot}(a_{md}) \end{pmatrix}$$

MLWE's relation to other problems The following two reductions, alongside the converse directions, are described in [5]. Here, only the "easy" directions are included.

The MLWE-problem can be reduced into the Module Shortest Integer Solution (MSIS) problem. The MSIS-problem is for a given matrix $\mathbf{A} \in M_{n \times m}(R_q)$ and a parameter β, to find a vector $\mathbf{z} \in R^m$ such that $\mathbf{A}\mathbf{z} = 0 \mod q$ and $0 < \|\mathbf{z}\| \leq \beta$. If $\mathbf{z}$ is the MSIS-solution for the matrix $\mathbf{A}^T$, where $\mathbf{A}$ is obtained from the MLWE-problem, then the inner product $\langle \mathbf{A}\mathbf{s} + \mathbf{e}, \mathbf{z} \rangle = (\mathbf{A}\mathbf{s} + \mathbf{e})^T\mathbf{z} = \mathbf{s}^T\mathbf{A}^T\mathbf{z} + \mathbf{e}^T\mathbf{z} = \mathbf{e}^T\mathbf{z}$. Since the error term is small, this inner product will also be small. The inner product of $\mathbf{z}$ and a $\mathbf{b} \leftarrow R_q^d$ is not necessarily small, which means that $(\mathbf{A}, \mathbf{A}\mathbf{s} + \mathbf{e})$ can be distinguished from $(\mathbf{A}, \mathbf{b})$ with high probability, meaning that the MLWE-problem will be solved.

Furthermore, the MSIS-problem can be reduced into the Module Shortest Independent Vector Problem (Mod-SIVP). The Mod-SIVP problem is given a lattice basis $\mathcal{B}$, find $n = \dim(\mathcal{L}(\mathcal{B}))$, linear independent vectors $s_1, s_2, \cdots, s_n$ such that $\max_i \|s_i\| \leq \gamma \|\mathbf{u}\|$, for a $\gamma \geq 1$, where $\mathbf{u}$ is any of the shortest vectors in $\mathcal{L}(\mathcal{B})$. This is done by looking at the lattice $\mathcal{L}(\mathcal{B}) = \{\mathbf{x} \in R^m \mid \mathbf{A}^T\mathbf{x} = 0\}$. If the Mod-SIVP-problem is solved in this lattice for a given γ, then that answer is also the answer for the MSIS problem. Then there exists a reduction from MLWE to Mod-SIVP, there also exists a reduction in the other direction as described in [5]. Thus, MLWE is as hard as Mod-SIVP, which is assumed to be hard to solve.

4.2 IND-CPA security

To show the security of Π, we will first consider a modified cryptosystem, where the public-key is not compressed in algorithm 1: it will just return $((\mathbf{As}+\mathbf{e}, s_1), \mathbf{s})$. The public-key input in line 2 of algorithm 2 will not be decompressed. The decryption algorithm will remain the same. This cryptosystem will be denoted by $\Pi' = (Gen', Enc', Dec)$. The following proof follows the proof from [4], but with additional elaboration.

Theorem 2. *If the MLWE-problem is hard, then the cryptosystem Π' is IND-CPA secure.*

Proof. Define game G_0 as follows:

---

1. The challenger, $\mathcal{C}$, generates a public-key, using Gen', and sends it the the adversary, $\mathcal{A}$.
2. $\mathcal{A}$ may preform a polynomial amount of encryptions.
3. $\mathcal{C}$ chooses a bit, b, randomly between 0 and 1.
4. $\mathcal{A}$ sends two messages, m_0 and m_1, to $\mathcal{C}$.
5. $\mathcal{C}$ encrypts m_b, using $Enc'(pk, \cdot)$, and sends it back to $\mathcal{A}$.
6. $\mathcal{A}$ may perform a polynomial amount of additional encryptions.
7. $\mathcal{A}$ sends back a guess, $\hat{b}$, of b.

We can similarly define a game G_1, where all the steps are the same except from 1., where instead of the public-key being generated from Gen', it is randomly chosen from R_q^k. Then we can define the advantage of an adversary, $\mathcal{A}$, in G_0 as $|P(\hat{b} = b : G_0) - \frac{1}{2}|$, which is also the advantage of an adversary, $\mathcal{A}$, in Π'.

Considering the expression $|P(\hat{b} = b : G_0) - P(\hat{b} = b : G_1)|$. Then

$$\begin{aligned}
|P(\hat{b} = b : G_0) - P(\hat{b} = b : G_1)| &= |P(\hat{b} = b \cap b = 0 : G_0) + P(\hat{b} = b \cap b = 1 : G_0) \\
&\quad - P(\hat{b} = b \cap b = 0 : G_1) - P(\hat{b} = b \cap b = 1 : G_1)| \\
&= |P(\hat{b} = 0 \cap b = 0 : G_0) + P(\hat{b} = 1 \cap b = 1 : G_0) \\
&\quad - P(\hat{b} = 0 \cap b = 0 : G_1) - P(\hat{b} = 1 \cap b = 1 : G_1)|.
\end{aligned}$$

By using the fact that $Pr(B \cap A) = Pr(A \cap B) = Pr(B)Pr(A \mid B)$ and $Pr(b = 1 \mid \hat{b} = 1) = \frac{1}{2}$, we can deduce that

$$\begin{aligned}|Pr(\hat{b} = b : G_0) - Pr(\hat{b} = b : G_1)| &= \frac{1}{2}|(Pr(\hat{b} = 1 : G_0) - Pr(\hat{b} = 1 : G_1) \\ &\quad + (Pr(\hat{b} = 0 : G_0) - Pr(\hat{b} = 0 : G_1))| \\ &\leq \frac{1}{2}|Pr(\hat{b} = 1 : G_0) - Pr(\hat{b} = 1 : G_1)| \\ &\quad + \frac{1}{2}|Pr(\hat{b} = 0 : G_0) - Pr(\hat{b} = 0 : G_1)|.\end{aligned}$$

The first term, $\frac{1}{2}|P(\hat{b} = 1 : G_0) - P(\hat{b} = 1 : G_1)| = \frac{1}{2}Adv^{mlwe}_{k,k,\nu}(\mathcal{B})$, for an adversary $\mathcal{B}$. The second term, $\frac{1}{2}|P(\hat{b} = 0 : G_0) - P(\hat{b} = 0 : G_1)|$, is equal to the first term, since $|P(\hat{b} = 0 : G_0) - P(\hat{b} = 0 : G_1)| = |(1 - P(\hat{b} = 1 : G_0)) - (1 - P(\hat{b} = 1 : G_1))| = |P(\hat{b} = 1 : G_0) - P(\hat{b} = 1 : G_1)|$. Thus $|P(\hat{b} = b : G_0) - P(\hat{b} = b : G_1)| \leq Adv^{mlwe}_{k,k,\nu}(\mathcal{B}) \leq Adv^{mlwe}_{k+1,k,\nu}(\mathcal{C})$, for an adversary $\mathcal{C}$. An adversary can't be worse when given an extra sample; the adversary could simply ignore it to get the same advantage as for k samples.

Define a game G_2 that's the same as G_1 but in the encryption algorithm, the values for $\mathbf{u}$ and v are chosen uniformly random from R_q^k and R_q respectively. If the same probability calculation as above is done with respect to G_1 and G_2, it will result in $|Pr(\hat{b} = b : G_1) - Pr(\hat{b} = b : G_2)| \leq |Pr(\hat{b} = 1 : G_1) - Pr(\hat{b} = 1 : G_2)| \leq Adv^{mlwe}_{k+1,k,\nu}(\mathcal{A})$, for an adversary $\mathcal{A}$. The reason for getting $k + 1$ is that $(\mathbf{u}, v)$ is treated as an element from R_q^{k+1}. Furthermore, since $\mathbf{u}$ and v are random in G_2, then $Pr(\hat{b} = 1 : G_2) = \frac{1}{2}$ and $|Pr(\hat{b} = 1 : G_1) - \frac{1}{2}| \leq Adv^{mlwe}_{k+1,k,\nu}(\mathcal{A})$.

These terms can be expanded into:

$$\begin{aligned}Adv^{\Pi'}_{cpa}(\mathcal{A}) &= \left|P(\hat{b} = b : G_0) - \frac{1}{2}\right| \\ &= \left|P(\hat{b} = b : G_0) - P(\hat{b} = b : G_1) + P(\hat{b} = b : G_1) - \frac{1}{2}\right| \\ &\leq \left|P(\hat{b} = b : G_0) - P(\hat{b} = b : G_1)\right| + \left|P(\hat{b} = b : G_1) - \frac{1}{2}\right| \\ &\leq 2Adv^{mlwe}_{k+1,k,\nu}(\mathcal{C})\end{aligned}$$

for an adversary $\mathcal{C}$. Since the MLWE-problem is assumed to be hard to solve, the MLWE-advantage will be negligible, and thus Π' will be IND-CPA secure. □

The security of Π', does not imply that Π is also secure. In despite of this, it is still believed that Π keeps this security. The problem lays in the distribution of

$$\mathbf{t} = Decompress_q(Compress_q(\mathbf{As} + \mathbf{e}, d_t), d_t)$$

no longer being uniform in R_q^k. Then the MLWE-assumption is no longer valid, and so the distribution of the encryption $\mathbf{t}^T\mathbf{r}+\mathbf{e}_2+\lceil\frac{q}{2}\rfloor m$, is no longer guaranteed to be computationally indistinguishable from a uniformly random distribution. This can however be fixed by adding a small error, $\mathbf{e}'$, depending on the values of $\mathbf{t}$ and d_t, to the encryption algorithm so that $Decompress_q(Compress_q(\mathbf{t}, d_t), d_t)+\mathbf{e}'$ is uniformly random.

By adding more error to the scheme, it will become less correct. This is simply because the value of ϵ mentioned in Theorem 1 will be slightly greater, since by introducing more error will make the percentage of the error terms being greater than or equal to $\lceil\frac{q}{4}\rfloor$ greater. As it turns out, it might not be necessary to add this error. By choosing the right value of d_v the compress function will "lose" this error term anyway [4]. That is for some values of d_v, the following equation holds:

$$Compress_q\left(\mathbf{t}^T\mathbf{r}+\mathbf{e}_2+\left\lceil\frac{q}{2}\right\rfloor m, d_v\right) = Compress_q\left((\mathbf{t}+\mathbf{e}')^T\mathbf{r}+\mathbf{e}_2+\left\lceil\frac{q}{2}\right\rfloor m, d_v\right)$$

This is a result of the rounding in the compression function. If a small term, e, is to be added to x, then the function will round $\frac{2^d}{q}(x+e)$ and since $e \ll x$, the rounding of $\frac{2^d}{q}x+\frac{2^d}{q}e$ will be the same as the rounding of $\frac{2^d}{q}x$.

5 KEM

A Key Encapsulation Mechanism (KEM) is a mechanism that allows two parties $\mathcal{A}$ and $\mathcal{B}$ to share a common secret-key. Furthermore, $\mathcal{A}$ and $\mathcal{B}$ can only communicate through a public network, so every message they send to each other can be read by a third party $\mathcal{C}$. The two parties $\mathcal{A}$ and $\mathcal{B}$ may want to share a common key to use for symmetric encryption, which is much more efficient than using an asymmetric cryptosystem.

5.1 Algorithms

There is a Key Encapsulation Mechanism that can be constructed from the cryptosystem, $\Pi = (Gen, Enc, Dec)$, discussed in section 3. This KEM will also use two hash functions, $G : \{0,1\}^* \to \{0,1\}^{2\times 256}$ and $H : \{0,1\}^* \to \{0,1\}^{256}$.

Encapsulation algorithm (4)
Input: (public-key=pk)

1: $m \leftarrow \{0,1\}^{256}$
2: $(\hat{k}, r) = G(H(pk), m)$
3: $(\mathbf{u}, v) = Enc(pk, m; r)$
4: $k = H(\hat{k}, H(\mathbf{u}, v))$
5: return $((\mathbf{u}, v), k)$

Instead of writing the public-key as $(\mathbf{t}, s_1)$, it will be denoted as pk. The notation $Enc(pk, m; r)$ means that instead of generating a random r, as in the first line of the encryption algorithm 2, the algorithm will use the value of r from the hash-functions. By not generating a random element in the encryption algorithm, and using this predetermined r, the encryption algorithm will be deterministic.

Decapsulation algorithm (5)
Input: (secret-key=$\mathbf{s}$, ciphertext=$(\mathbf{u}, v)$)

1: $m' = Dec(\mathbf{s}, (\mathbf{u}, v))$
2: $(\hat{k}', r') = G(H(pk), m')$
3: $(\mathbf{u}', v') = Enc(pk, m'; r')$
4: if $(\mathbf{u}', v') = (\mathbf{u}, v)$:
5: return $k = H(\hat{k}, H(\mathbf{u}, v))$
6: else:
7: return $k = H(z, H(\mathbf{u}, v))$

Here we choose to not have the public-key as an input as it is public, and should be known, for all users on a network. In the case that re-encryption fails in line 3, the algorithm will output a key based on a random z. This Key Encapsulation Mechanism described in (4) and (5) is Kyber, it will be denoted as $\Psi = (Gen, Encap, Decap)$.

With algorithms (4) and (5), two parties $\mathcal{A}$ and $\mathcal{B}$ can share a common secret-key. This is done by $\mathcal{A}$ first using the key generation algorithm (1) to generate a public-key and a private-key. $\mathcal{A}$ sends the public-key to $\mathcal{B}$. Then $\mathcal{B}$ uses the the key encapsulation algorithm (4) to generate a key, k, and a ciphertext $(\mathbf{u}, v)$. $\mathcal{B}$ sends this ciphertext back to $\mathcal{A}$, and $\mathcal{A}$ can then use the decapsulation algorithm (5) with the secret-key and the ciphertext as inputs. The algorithm will then give $\mathcal{A}$ the same key, k, as $\mathcal{B}$ have. A third party $\mathcal{C}$ needs to have the secret-key in order to get the key, k. If $\mathcal{C}$ tries decapsulate the ciphertext using another key than the secret-key, the decapsulation algorithm (5) will output a key based on the ciphertext and a random element z.

5.2 IND-CCA security

The following theorem describes the correctness of Ψ. Here ϵ is defined as in Theorem 1.

Theorem 3. *If the cryptosystem Π is $1 - \epsilon$ correct, then the key encapsulation mechanism Ψ will be $1 - \epsilon$ correct.*

Proof. Let $pk, sk \leftarrow Gen()$. When using pk in $Encap$, the algorithm will compute the ciphertext $(\mathbf{u}, v) = Enc(pk, m; r)$ and the key $k = H(\hat{k}, H(\mathbf{u}, v))$, and outputs these two. When using $Decap$ with sk and $(\mathbf{u}, v)$ as input, the algorithm will first compute $m' = Dec(sk, (\mathbf{u}, v)) = Dec(sk, Enc(pk, m)$. From Theorem

1, m' is equal to m in $(1-\epsilon)100\%$ of the cases. Then $\hat{k}', \hat{r}'$ in (5) will be the same as $\hat{k}, r$ as in (4) and $(\mathbf{u}', v') = (\mathbf{u}, v)$. So (5) will return $k = H(\hat{k}, H(\mathbf{u}, v))$, which is the same key outputted from (4). □

On classical computers, the security of Ψ can be expressed as in the following theorem. Here, the hash functions, H and G, are modelled as random oracles. The bound for the CCA-advantage is provided from [6], where ϵ is defined as in Theorem 1.

Theorem 4. *If the cryptosystem Π is IND-CPA secure, then Ψ will be IND-CCA secure. More precisely, for any classical adversary $\mathcal{A}$, making at most q_{RO} queries to the random oracles H and G, there exists an adversary $\mathcal{B}$ such that:*

$$Adv^{\Psi}_{cca}(\mathcal{A}) \leq 3Adv^{\Pi}_{cpa}(\mathcal{B}) + q_{RO} \cdot \epsilon + \frac{3q_{RO}}{2^{256}}$$

When the random oracles H and G are modelled as quantum random oracles, there exists a bound described in [4] on Adv^{Ψ}_{cca} expressed as the square root of Adv^{Π}_{cpa} plus some q_{RO}-terms. This makes Ψ a quantum-resistant cryptosystem, but to achieve some desired bit-security, Π has to have double the bit-security, i.e., for Ψ to have 128-bit security, then Π has to be 256-bit secure. In practice, for Ψ to be used as a quantum resistant cryptosystem, the running times will be longer than if it is used for classical computers.

6 References

[1] P. W. Shor, "Algorithms for quantum computation: Discrete logarithms and factoring," in *35th Annual Symposium on Foundations of Computer Science*. Santa Fe, NM, USA: IEEE Computer Society Press, Nov. 20–22, 1994, pp. 124–134.

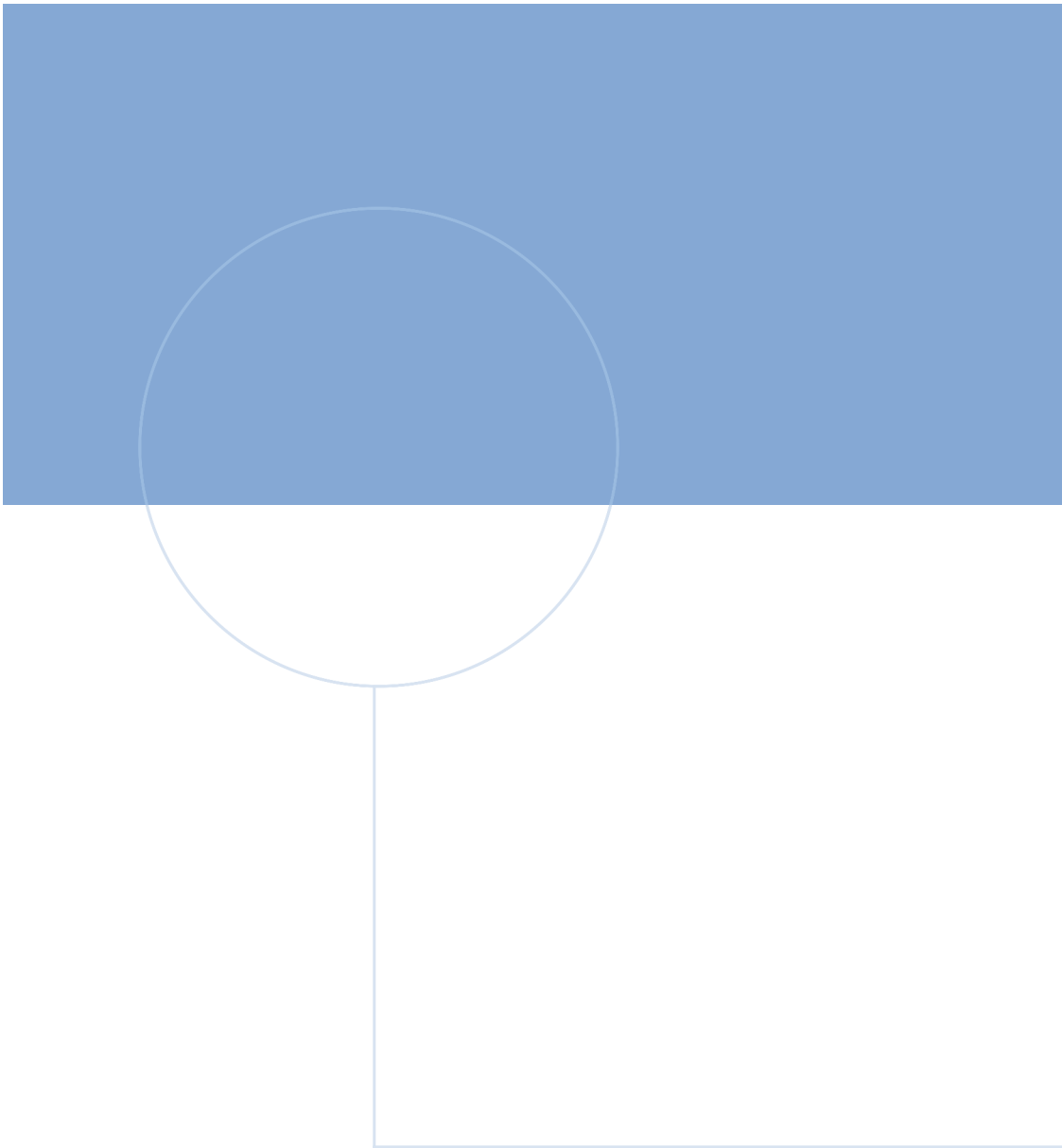
[2] NIST, "Post-quantum cryptography," 2017. [Online]. Available: https://csrc.nist.gov/Projects/Post-Quantum-Cryptography

[3] CRYSTALS, "Kyber," 2017. [Online]. Available: https://pq-crystals.org/kyber/

[4] J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, and D. Stehlé, "CRYSTALS – Kyber: a CCA-secure module-lattice-based KEM," Cryptology ePrint Archive, Report 2017/634, 2017, https://eprint.iacr.org/2017/634.

[5] A. Langlois and D. Stehlé, "Worst-case to average-case reductions for module lattices," Cryptology ePrint Archive, Report 2012/090, 2012, https://eprint.iacr.org/2012/090.

[6] D. Hofheinz, K. Hövelmanns, and E. Kiltz, "A modular analysis of the Fujisaki-Okamoto transformation," in *TCC 2017: 15th Theory of Cryptography Conference, Part I*, ser. Lecture Notes in Computer Science, Y. Kalai and L. Reyzin, Eds., vol. 10677. Baltimore, MD, USA: Springer, Heidelberg, Germany, Nov. 12–15, 2017, pp. 341–371.



NTNU

Norwegian University of
Science and Technology