Report on evaluation of KpqC Round-2 candidates

Daniel J. Bernstein, Jolijn Cottaar,
Emanuele Di Giandomenico, Kathrin Hövelmanns,
Andreas Hülsing, Mikhail Kudinov, Tanja Lange,
Mairon Mahzoun, Matthias Meijers, Alex Pellegrini,
Alberto Ravagnani, Silvia Ritsch, Sven Schäge,
Tianxin Tang, Monika Trimoska,
Marc Vorstermans, and Fiona Johanna Weber

30 September 2024
with some updates through 26 December 2024

Author list in alphabetical order, see: https://www.ams.org/profession/leaders/culture/CultureStatement04.pdf

Contents

Chapter 1

Introduction

This report is an update on the TU/e final report on Round 1 of KpqC. It covers our work on the 2nd-round candidates in the KpqC competition. We document the functioning and highlight changes in the submissions. In some cases we have found attacks on the specified systems, and in some cases we have found attacks exploiting bugs or timing variations in implementations. For all systems, we checked the security proofs to the extent that any were given and highlight gaps and issues. We revisit our security analysis of the generic attacks in case the designers changed the parameters or new attacks have been developed.

Each submission is treated in its own chapter, but given that four systems use lattices as the underlying structure, we treat lattices, lattice problems, and generic attacks in the following background chapter. We also cover transforms used in the constructions and their security proofs and comment on the software evaluation in this background chapter. After that we cover KEMs and then signature systems in alphabetical order.

Chapter 2

Background

Multiple schemes submitted to the Korean Post Quantum competition are based on some form of the Learning with Errors (LWE) problem or some of its variants, including Ring (RLWE), Module (MLWE), Learning with Rounding (LWR or M/RLWR) and NTRU. Of the round-2 candidates we have SMAUG-T (MLWE), HAETAE (MLWE), NCC-sign (RLWE), and NTRU+ (NTRU). The signature systems can also be attacked via variants of another problem called SIS: HAETAE relies on a "self-target" version of MSIS, and NCC-sign relies on a "self-target" version of RSIS.
In this chapter we first cover generic lattice attacks and then introduce the most common problems and tools in security proofs. For systems not based on lattices this is given as part of the chapter introducing the system and short introductions are given also for the lattice schemes to keep the chapters concise.
Finally, we comment on our methodology for performance evaluation.

2.1 Generic lattice attacks

Lattice-based schemes base their security on the hardness of finding short or close vectors in lattices. The schemes differ in how these lattices are defined and how they base the functionality of KEM or signature scheme on the lattice problem but they share the generic lattice attacks that apply. These are algorithms that find short non-zero vectors in the lattice. The runtime of the algorithms typically increases with the quality of output, ranging from polynomial for the LLL algorithm, which finds short vectors within an exponential approximation factor of the shortest length, to exponential for enumeration and sieving algorithms that can find vectors of minimal length. The BKZ lattice reduction algorithm [SE91] permits to interpolate between

these two extremes by means of the "block size" β. The BKZ algorithm repeatedly solves the exact Shortest Vector Problem (SVP) in a lattice of dimension β. These lattices are obtained as projections of the original lattice to a subspace. Taking a linear combination of vectors in that subspace, which give a shortest vector there, gives a relatively short vector in the full space. BKZ iterates taking a selection of β vectors, solving the exact SVP in the projection, moving back to the full space and considering the next set of β vectors to repeat. The special case of $\beta = 2$ is the LLL algorithm while taking β equal to the lattice dimension means solving the exact problem in the full lattice. LLL and BKZ come with stopping conditions on the length of the vectors which translate into approximation factors of how close to the shortest length these vectors get. In general, a larger β leads to shorter vectors at a longer runtime, however, often it is not necessary to compute the shortest vector in order to break the system, some relatively short vector, no longer than γ times the shortest length may be sufficient. Furthermore, a lattice might lend itself to shorter runtimes, e.g. for the integer lattice $\mathbf{Z}^n$ one immediately writes down n independent vectors of length 1, which is much shorter than would be expected from a random lattice, but generic lattices will not have such extreme cases.
Asymptotically, a lower bound of the attack cost is given by the time to solve SVP in dimension β and ignoring all lower-order terms, the iterations ("tours") in BKZ, and the costs of storage. We caution the reader that this need not be an actual lower bound as the lower-order terms might have negative signs and might have large coefficients. Other models try to account for more of the factors.
Albrecht, Player and Scott have created a lattice estimator used to estimate the costs of the above attack in [APS15]. Note that even though this eprint is not updated, their latest lattice-estimator software is at the moment (to the best of our knowledge) the state-of-the-art estimator available. This lattice-estimator software is different from the "LWE estimator" used in [ACD+18], in the round-1 report, and in many lattice proposals.
In order to use the lattice-estimator software, all the schemes, no matter the form of the problem, will be reduced to an LWE instantiation. On this LWE instantiation the estimator is then run. Note that this indeed gives only estimates, and should be seen as a indication, but very small numbers indicate problems. The estimates are included in the chapters on the respective schemes.
We use the following names for four different estimates that can be obtained from the lattice-estimator software:

- Primal Core-SVP. This is `usvp` from `estimate.rough` in lattice-estimator. This is the simplest estimate, and currently has the best comparability to the literature.

- Dual Core-SVP. This is `dual_hybrid` from `estimate.rough` in lattice-estimator. This estimates the cost of a different “dual” attack strategy. Occasionally Dual Core-SVP is noticeably below Primal Core-SVP, but there is ongoing controversy regarding this speedup (see, e.g., [DP23] and [PS24]).
- BDD. This is `bdd` (referring to bounded-distance decoding) from `estimate` (not `estimate.rough`) in lattice-estimator. This incorporates various effects ignored in Primal Core-SVP, and currently has the best comparability to the part of the literature looking “beyond Core-SVP”.
- Dual Hybrid. This is `dual_hybrid` from `estimate` in lattice-estimator. This incorporates various effects ignored in Dual Core-SVP. This is controversial for the same reasons as Dual Core-SVP.

The lattice-estimator software provides all four of these estimates for LWE-based schemes, but does not provide the dual estimates for NTRU-based schemes.

For HAETAE, the round-1 submission package included precompiled fragments of a script to estimate security levels for LWE and SIS. We applied a decompilation tool and linked in the estimator from CRYSTALS, but the script still did not finish successfully after more than two days of CPU time, so we terminated the run. The round-2 submission package instead includes a script that runs successfully. When we tried

```
python3 HAETAE_security_estimates.py \
--param="n=256 q=64513 k=2 l=4 eta=1 gamma=48.858 security=120 adapt=1"
```

we received output close to—but not exactly—the 120 column in Table 1 of the round-2 HAETAE documentation.

2.2 Security assumptions

Many LWE-like security assumptions have an interesting property of worst-case-to-average-case reductions: an algorithm to break the average case of the LWE-like assumption can be used, with at most a polynomial loss of efficiency, to break the worst case of a specified lattice problem.

The specified lattice problem is typically an approximate short vector problem for a class of lattices sharing key properties of the LWE-like assumption.

For example, a subexponential-time algorithm to break Ring-LWE in the average case implies a subexponential-time algorithm to break approximate short-vector problems in the worst case for *ideal* lattices. The approximation factor is also related to the Ring-LWE parameters.

2.2.1 The importance of studying the security level of assumptions

It is important to note that a worst-case-to-average-case reduction is not a guarantee of security. There are three basic reasons for this.
First, the polynomial loss of efficiency mentioned above is very large, so the theorems are consistent with the possibility that the LWE-like assumptions are much easier to break than the worst-case lattice problems. This has far-reaching consequences.
A concrete analysis and optimization from [Gär23] proves 2^{128} QROM IND-CCA2 security for a particular lattice system using dimension 79510 and a 37-bit modulus, assuming optimality of attacks against a particular worst-case lattice problem. This lattice dimension is much larger than lattice dimensions that have been deployed.
This means that for the currently used parameters, to the best of our knowledge, all lattice-based encryption systems do not reduce to some underlying classical worst-case assumption in a theoretically-sound way. Put differently, when applying the existing security reductions to the parameter sizes used in practice they would merely imply that a classical lattice problem can be solved that has very small parameters. However, such a scheme could be practically solved efficiently in any case, making the guarantees from the security proof vacuous.
Second, the cryptanalytic literature often breaks worst-case problems. Consider, for example, the discrete-logarithm problem for the multiplicative group $\mathbb{F}_{2^n}^*$. This problem has a worst-case-to-average-case reduction: an attack against the average case of the problem implies an attack against a worst-case problem (namely the worst case of the same problem). However, known attacks break this problem in polynomial time with a quantum computer, and in quasipolynomial time even without a quantum computer.
Third, even when all known attacks against a problem take exponential time, secure parameter selection requires understanding the attack cost in more detail. For example, in selecting a prime p to use for pre-quantum elliptic-curve cryptography over $\mathbb{F}_p$, it is important to understand that attacks use just $p^{1/2}$ operations rather than p operations.
The following sections look more closely at the most important attack prob-

lems in lattice-based cryptography. This is not a description of the known attack algorithms or the attack costs; it is an overview of problems that are *assumed* to be hard.

2.2.2 Assumptions used in cryptography for encryption systems

An LWE-like assumption states that the distributions $A, b = A \cdot s + e$ and A, b' are computationally indistinguishable where A is a suitable uniformly drawn (quadratic) matrix of dimension n (a lattice), s is a secret vector, e is a small error vector, and b' is a random vector. All values are integers. We observe that essentially $s \mapsto A \cdot s$ is a linear operation that is perturbed by some error term. Moreover, A spans a lattice. The closeness to purely linear operations gives these schemes their efficiency. In the literature, we can find several variants where the entries in A, s, e, b, b' range over different algebraic structures (and accordingly the operations $\cdot, +$ are defined differently). However, each of these assumptions requires that the mapping $s \mapsto A \cdot s + e$ is injective, giving important conditions on the size of the parameters.
In plain LWE, the vector and matrix entries are elements of some ring $\mathbb{Z}_q$ while $\cdot$ represents matrix multiplication modulo q. Ring LWE (RLWE) instead works with polynomials in the ring $\mathbf{Z}[x]/f(x)$ for some polynomial $f(x)$, typically $f(x) = x^n + 1$ and additionally reduces modulo some number q. The equation $A \cdot s$ turns into the multiplication of the *polynomials* a and s modulo $f(x)$. This can also be written in matrix form where rows of A are $a \cdot x^i$ followed by reduction modulo $f(x)$ and modulo q. Module LWE (MLWE) can be interpreted as a generalization to a spectrum that has LWE and RLWE as its endpoints, parameterizing the additional polynomial structure introduced [AD17]. In general these variants require more algebraic structure, with RLWE introducing stronger requirements on the algebraic structure than general MLWE.
Peikert and Pepin in [PP19] presented a general treatment of the Learning with Errors (LWE) assumption and its variants. Roughly, their framework gives a tight reduction from Ring-LWE (RLWE) to other algebraic LWE variants, including Module-LWE (MLWE), Order-LWE, and Middle-Product LWE. Their work shows that it is possible to use the hardness of Ring-LWE as a foundation for the hardness of all prior algebraic LWE problems. When focusing on LWE, MLWE, and RLWE, this is natural as RLWE places the strongest conditions on the additional structure. However, so far there are no attacks that can specifically exploit the additional structure induced by RLWE (or MLWE). A benefit is that the efficiency of the cryptosystem that

are based on LWE-variants can be higher than in plain LWE, both in terms of size as well as in speed of computation. The overall result when using RLWE and MLWE is that in Regev-like encryption systems, we can have higher efficiency with these variants. Roughly in Regev's original cryptosystem, an encryption of a single message bit would require n elements of $\mathbb{Z}_q$ in the ciphertext and public key. Using RLWE in Regev-like cryptosystems, decreases this to a single value. Whereas the definition of RLWE relies on a single polynomial, MLWE considers vectors of polynomials. When comparing this to RLWE, this can be used to balance the number of components of the vector with the length of its entries. Intuitively, at the same level of security, RLWE has to compensate for considering just one polynomial by having polynomials of larger degrees [AD17].

2.2.3 Computational vs. decisional LWE-variants

The computational version of plain LWE and its variants requires us to compute the secret s from a A, b with $b = A \cdot s + e$ instead of distinguishing it from random (A, b'). The computational and decisional version of LWE and its variants are polynomial-time equivalent. In the following we will usually be concerned with decisional variants since we deal with encryption systems that intrinsically capture that ciphertexts do not reveal a single bit to the attacker by requiring that ciphertexts are indistinguishable from encryptions of random values. We note that there is still a security loss when reducing the computational variant to the decisional [STHY23].

2.2.4 LWE vs. LWR

Whereas the LWE problem adds a small random error e to an otherwise linear equation, the learning with rounding (LWR) assumption introduces a deterministic error that intuitively cuts-off some of the least significant bits (LSBs) of the linear equation. LWE reveals $A, As+e$; LWR reveals A, $\lfloor As \rceil_{p,q}$ where $\lfloor x \rceil_{p,q}$ denotes $\lfloor x(p/q) \rceil$, the rounding of $x(p/q)$ to the neaarest integer. If $p < q$ then this maps values in $[0, q-1]$ to the smaller set $[0, p-1]$ and thus loses information. The resulting elements are smaller and improve bandwidth. Essentially, instead of blinding the least significant bits with an error e, LWR simply deletes the least significant bits.
Note that $(q/p)\lfloor As \rceil_{p,q}$ is close to As, and can thus be written as $As + e$ where e is small. However, LWR chooses this e deterministically from As, whereas LWE chooses e randomly.
An LWR attack can be used as an LWE attack as follows: given an LWE sample $A, As+e$, discard the bottom bits of $As+e$ to obtain an LWR sample,

and then apply the LWR attack. This reduction works with high probability if the amount of rounding is large enough; on the other hand, the reduction does not imply that LWR is as hard as LWE when the amount of rounding is similar to the size of e.

2.2.5 Partially-correct encryption system

Due to the introduction of errors (that can in rare cases accumulate quickly), most cryptosystems based on lattices feature non-perfect correctness. This means that in rare cases the decryption of a ciphertext may fail. The probability for this to happen is called decryption failure probability (DFP). For practical parameters, the DFP is usually very small, so that decryption failures will virtually not happen in most usage scenarios. However, an attacker that can find decryption failures learns valuable information on the secret key [DRV20]. Thus attackers might use strategies that specifically search for decryption failures. Recent improvements in these strategies improve the success probability to find more decryption failures once a single one has been found for a given key pair [DB22]. So the probabilities of a lattice-based cryptosystem need to be chosen such that finding any decryption failure is hard *in the first place*. However, recent analysis reveals that the DFP of most schemes used in practice are low enough when fixing the maximum number of decryption queries in total to some practical values [DRV20]. It is important to note that when proving security against quantum attackers as opposed to classical attackers, Grover's search algorithm can be utilized by any attacker and in particular attackers that aim at finding decryption failures.

2.2.6 Security assumptions

All schemes are essentially following the same template. For integers p, q with $2 < p \leq q$, if $w \in \mathbb{Z}_q$, let $\lfloor w \rceil_{p,q}$ denote $\lfloor w \cdot p/q \rceil$, where $\lfloor v \rceil$ represents rounding to the integer that is nearest to v. If $p = q$, no rounding is performed whatsoever. If $w \in R_q$ for some polynomial ring R_q with coefficients over $\mathbb{Z}_q$, $\lfloor w \rceil_{p,q}$ applies the rounding operation to each coefficient of w. Likewise, rounding a vector of elements will apply the rounding to each component of the vector. By convention we will understand that having an (error) distribution $\chi_x = \{0\}$ that always maps to zero implies that we will not draw an error at all.

Definition 2.2.1 (LWE). *Let n, q be positive integers and let χ_{LWE} be a probability distribution on $\mathbf{Z}_q^n$. Implicitly set $p = q$. Choosing a matrix $A \in \mathbf{Z}_q^{n \times n}$ uniformly at random and choosing $e \in \mathbf{Z}_q^n$ according to χ_{LWE}, define $A_{s,\chi_{\mathsf{LWE}}}$*

as the probability distribution on $\mathbf{Z}_q^{n\times n} \times \mathbf{Z}_q^n$ *that outputs* $(A, b = A \cdot s + e)$ *for given secret* $s \in \mathbf{Z}_q^n$*. Define* $U_{\mathsf{LWE}} = (A, w)$ *where* w *is uniform in* $\mathbf{Z}_q^n$*.*

Definition 2.2.2 (LWR)**.** *Let* n, q, p *be positive integers with* $p < q$*.Choosing a matrix* $A \in \mathbf{Z}_q^{n\times n}$ *uniformly at random, define* $\chi_{\mathsf{LWR}} = \{0\}^n$ *to be a distribution that always maps to zero and* $A_{\mathbf{s},\chi_{\mathsf{LWR}}}$ *as the distribution on* $\mathbf{Z}_q^{n\times n} \times \mathbf{Z}_q^n$ *that outputs* $(A, b = \lfloor A \cdot s + e \rceil_{p,q})$ *for given secret* $s \in \mathbf{Z}_q^n$*. Define* U_{LWR} *as the uniform distribution on* $\mathbf{Z}_q^n \times \mathbf{Z}_q$*. Define* $U_{\mathsf{LWR}} = (A, w)$ *where* w *is uniform in* $\mathbf{Z}_q^n$*.*

Definition 2.2.3 (RLWE)**.** *Let* n *be an integer, let* q *be a prime integer. Define* $R_q = \mathbf{Z}_q[x]/(x^n + 1)$ *and let* χ_{RLWE} *be a probability distribution on* R_q*. Implicitly set* $p = q$*. Choosing a polynomial* $A \in R_q$ *uniformly at random, drawing* e *at according to* χ_{RLWE}*, define* $A_{s,\chi_{\mathsf{RLWE}}}$ *as the probability distribution on* $R_q \times R_q$ *that outputs* $(A, b = A \cdot s + e)$ *for given secret* $s \in R_q$*. Define* $U_{\mathsf{RLWE}} = (A, w)$ *where* w *is uniform in* R_q*.*

Definition 2.2.4 (RLWR)**.** *Let* n *be an integer, let* q *be a prime integer, and let* p *be an integer with* $p < q$*. Define* $R_q = \mathbf{Z}_q[x]/(x^n + 1)$*. Define* $\chi_{\mathsf{RLWR}} = \{0\}^n$ *to be a distribution that always maps to zero. Choosing a polynomial* $A \in R_q$ *uniformly at random define* $\chi_{\mathsf{RLWR}} = p$ *and* $A_{s,\chi_{\mathsf{RLWR}}}$ *as the probability distribution on* $R_q \times R_q$ *that outputs* $(A, b = \lfloor (A \cdot s) \rceil_{p,q})$ *for given secret* $s \in R_q$*. Define* $U_{\mathsf{RLWR}} = (A, w)$ *where* w *is a uniform polynomial in* R_q*.*

Definition 2.2.5 (MLWE)**.** *Let* $k, m,$ *and* n *be integers, let* q *be a prime integer. Define* $R = \mathbf{Z}[x]/(x^n + 1)$ *and* $R_q = R/qR$ *and let* χ_{MLWE} *be a probability distribution on* R_q^k*. Implicitly set* $p = q$*. Choosing a matrix* $A \in R_q^{k\times m}$ *uniformly at random and an error* e *according to* χ_{MLWE}*, define the probability distribution* $A_{s,\chi_{\mathsf{MLWE}}}$ *on* $R_q^{k\times m} \times R_q^k$ *that outputs* $(A, b = A \cdot s + e)$ *for given secret* $s \in R_q^m$*. Define* $U_{\mathsf{MLWE}} = (A, w)$ *where* w *is uniform in* R_q^k*.*

Definition 2.2.6 (MLWR)**.** *Let* $k, m,$ *and* n *be integers, let* q *be a prime integer, and let* p *be an integer with* $p < q$*. Define* $R = \mathbf{Z}[x]/(x^n + 1)$ *and* $R_q = R/qR$*. Define* $\chi_{\mathsf{MLWR}} = \{0\}^{kn}$ *to be a distribution that always maps to zero. Choosing a matrix* $A \in R_q^{k\times m}$ *uniformly at random define* $A_{s,\chi_{\mathsf{MLWR}}}$ *as the probability distribution on* $R_q^{k\times m} \times R_q^k$ *that outputs* $(A, b = \lfloor (A \cdot s) \rceil_{p,q})$ *for given secret* $s \in R_q^m$*. Define* $U_{\mathsf{MLWR}} = (A, w)$ *where* w *is uniform in* R_q^k*.*

Definition 2.2.7 (Alternative Version)**.** *Consider the* x*-assumption for* $x \in \{\mathsf{LWE}, \mathsf{RLWE}, \mathsf{MLWE}, \mathsf{LWR}, \mathsf{RLWR}, \mathsf{MLWR}\}$*. Using the same parameters, we say that* x' *is the alternative version of* x *if in the computation of the output distribution we compute* $s^T \cdot A$ *instead of* $A \cdot s$*.*

We refer to LWE, MLWE, RLWE (and to any of the corresponding alternative versions) generally as LWE-like and more specifically to LWR, MLWR, RLWR (and to any of the corresponding alternative versions) as LWR-like.

Definition 2.2.8 (Computational LWE-like Problems)**.** *Assume we draw s according to some distribution $\chi_{x,s}$. Given an arbitrary number of independent samples from A_{s,χ_x}, with $x \in \{\mathsf{LWE}, \mathsf{RLWE}, \mathsf{MLWE}, \mathsf{LWR}, \mathsf{RLWR}, \mathsf{MLWR}\}$ (or a corresponding alternative version) the computational x problem asks to find s.*

Definition 2.2.9 ((Decisional) LWE-like Problems)**.** *Assume we draw s according to some distribution $\chi_{x,s}$. Given an arbitrary number of independent samples from A_{s,χ_x} or from the distribution U_x, with $x \in \{\mathsf{LWE}, \mathsf{RLWE}, \mathsf{MLWE}, \mathsf{LWR}, \mathsf{RLWR}, \mathsf{MLWR}\}$ (or a corresponding alternative version), the decisional x problem asks to distinguish between samples from A_{s,χ_x} and samples from U_x.*

When in the following we speak of any LWE-like assumption we specifically refer to its decisional variant. In classical formulations of the assumptions we typically have that $\chi_{x,s}$ is the uniform distribution. We say that LWE and LWR have the same algebraic structure if they have the same parameters n, q and they share A. Likewise, we say that RLWE and RLWR have the same algebraic structure if they share the same parameters n, q and they share A. Finally we say that MLWE and MLWR have the same algebraic structure if they share the same parameters k, m, n, q and they share random A.
The results in [ACPS09] show that for LWE-like schemes the secret keys can come from the same (Gaussian) distribution as the error. This accounts for virtually no security loss. This justifies using small secret keys in the first place.

2.2.7 The basic Regev cryptosystem

Description

We now describe a general form of the Regev system. The Regev system can be based on the $x \in \{\mathsf{LWE}, \mathsf{RLWE}, \mathsf{MLWE}, \mathsf{LWR}, \mathsf{RLWR}, \mathsf{MLWR}\}$ assumption for key generation and the y assumption for ciphertext generation where x and y have the same algebraic structure but possibly distinct p, p'. Observe that these assumptions implicitly define all ambient spaces.
Below we consider a message space $\mathcal{M} = \{0,1\}$ for $x \in \{\mathsf{LWE}, \mathsf{LWR}\}$ and $\mathcal{M} = \{0,1\}^n$ for $x \in \{\mathsf{RLWE}, \mathsf{MLWE}, \mathsf{RLWR}, \mathsf{MLWR}\}$. The PKE scheme consists of a collection of three algorithms $\mathsf{PKE} = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$:

- $\mathsf{KeyGen}(1^\kappa)$: The key generator chooses uniformly random A. Next it samples $b = \lfloor A{\cdot}s{+}e\rceil_{p,q}$ where s is chosen according to some distribution $\chi_{x,s}$ and $e \leftarrow \chi_x$. The public key is $\mathsf{pk} = (A, b)$ while the secret key is $\mathsf{sk} = s$.

- $\mathsf{Encrypt}(\mathsf{pk}, m)$: To encrypt message m, we compute an ephemeral key $b' = \lfloor s'^T \cdot A + e'\rceil_{p',q}$ for some s' that has been drawn from distribution $\chi_{y,s'}$ and $e' \leftarrow \chi_y$. Next we use the public key of the receiver $\mathsf{pk} = (A, b)$ to compute (c_1, c_2) where $c_1 = b'$ and $c_2 = \lfloor s'^T \cdot b + e''\rceil_{p',q} + \lfloor mp'/2\rceil$ where e'' is drawn according to χ_y.

- $\mathsf{Decrypt}(\mathsf{sk}, c)$: To decrypt ciphertext (c_1, c_2) using $\mathsf{sk} = s$, we compute $mq/2 \approx c_2(q/p') - c_1(q/p') \cdot s$ from which we can easily compute m.

Correctness

We say PKE has correctness δ if it holds that the probability $\Pr[m = \mathsf{Decrypt}(\mathsf{sk}, \mathsf{Encrypt}(\mathsf{pk}, m)) | (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\kappa)] = \delta$. We define the decryption failure probability DFP as $\mathsf{DFP} = 1 - \delta$.

Security

Consider the following game played between attacker A and challenger C.

- The challenger draws a random key pair using $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\kappa)$ and sends pk to the attacker.

- The attacker outputs a message m^*.

- The challenger draws random bit b and random message $m' \in \mathcal{M}$. It sets $m_0 := m^*$ and $m_1 := m'$. Next it computes $c^* \leftarrow \mathsf{Encrypt}(\mathsf{pk}, m_b)$ and sends it to A.

- The attacker outputs a bit b' indicating its guess for m^*.

The attacker wins if $b' = b$. The advantage of the attacker to win is defined as $\mathsf{Adv} = |\mathsf{Pr}[0 \leftarrow A(\mathsf{pk}, c^*)] - \mathsf{Pr}[1 \leftarrow A(\mathsf{pk}, c^*)]|$ where the probability is over the random choices of A and C.

Security proof of generic Regev scheme

The security proof for this scheme is very simple. In the first game hop, we exchange the public key element b with a truly random value b'. Any attacker that can observe this change can immediately be used to break the

x assumption. Next we change c_1 to random b''. Again, any attacker that can observe this change can be used to break the y assumption. Now we are in a game where the ciphertext is entirely independent of the public key and the message. Thus the attacker can only guess with probability 1/2.

Security loss

The security of the basic Regev scheme reduces tightly to that of the x or y assumption.

2.3 General design frameworks and provable security

2.3.1 The Fujisaki-Okamoto transform (with implicit rejection)

The Fujisaki-Okamoto (FO) transform turns a weakly secure PKE to an IND-CCA secure one [FO99, FO13]. The FO transform has been originally described for PKE. Dent transferred it to construction of KEMs [Den03]. However, the transforms only cover schemes with perfect correctness. The application to partially-correct schemes got much attention in the last years starting with [HHK17]. A good overview on the classical reductions is given in [FO13]. Some of the key insights are that:

- When using quantum reductions, there is a tight proof from deterministic PKEs with an additional strong property called disjoint simulatability to IND-CCA secure KEMs in the quantum random oracle model (QROM) [SXY18]. Disjoint simulatability is implied by sparse pseudo-randomness.

- When using classical reductions, there is a tight proof from a subclass of probabilistic PKEs to IND-CCA secure KEMs in the random oracle model [FO13].

- When using classical reductions, there is a tight proof from deterministic PKEs to IND-CCA secure KEMs in the random oracle model [FO13].

- When using quantum reductions, there is a non-tight proof [HHM22, BHH$^+$19] from deterministic and probabilistic PKEs to IND-CCA security in the quantum random oracle model (QROM). To avoid high security losses, it is useful to require implicit rejection, where the decryption oracle outputs random responses in case of a decryption failure. All black-box reductions have a square loss of security. Given current knowledge, it seems hard to avoid this non-tight security loss [JZM21] for measurement-based reductions (that measure the output of the QROM). Under certain assumptions on the reversibility of the attacker a non-black box reduction presented at EC'20 has a linear loss.

- Existing KEMs often include the receiver's public key to derive the shared key in the key-derivation function. The reason is usually that in this way they can achieve a form of domain separation that makes it harder to launch attacks exploiting decryption failures. However, there are also more formal arguments to apply this technique. It can be used to improve the tightness loss over the standard hybrid argument [BBM00] when analyzing in the more realistic multi-user setting. to derive the shared key. However, as [DHK+21] show, using the full pk is wasteful, a prefix of it suffices. Relying on just the prefix results in a considerable decrease of the running time of Kyber as opposed to using the full public key.

  We emphasize that the same reasoning applies if hashed public keys (via some other collision-resistant hash function) are used instead of the full public keys when deriving the shared key. With the same arguments as before one can use a truncated hash value to derive the shared key. However, the benefit of introducing this additional hash is generally unclear. We note that by explicitly considering the entropy of the truncated public keys, [DHK+21] provide alternative results that deviate from the existing variants of the Fujisaki-Okamoto transform. Essentially, the new transforms do not require the underlying PKE system to provide spreadness. Moreover, for certain parameters, the security losses reported in [DHK+21] improve on previous work.

We note that even though the results of [DHK+21] present a much more realistic setting it still does not provide corruption capabilities to the attacker. Strictly speaking, their result only holds if no party is ever corrupted. It is unclear if their improvements over the naive hybrid argument transfer to the setting with adaptive user corruptions as well.

2.3.2 Security loss

The security proof should have a low security loss that theoretically supports practical parameter choices. This however, highly depends on the security assumption that the security of the scheme is reduced to. As stated before, reductions to the worst-case hardness of classical lattice-based assumptions are very likely to not cover practical parameters since they have considerably high security losses. However, even if the reduction reduces to some decisional LWE-like assumption, the reduction is typically not tight, in particular, if it assumes quantum attackers in the so-called quantum random oracle model (QROM). This is mainly due to the application of the popular Fujisaki-Okamoto transform (and its variants) that generically turns an IND-CPA secure PKE into a IND-CCA2 secure KEM [HHM22, BHH+19]. Recent results show that under certain conditions this loss is unavoidable [JZM21] (for measurement-based, black-box reductions).

2.3.3 The Fiat-Shamir transform

The Fiat-Shamir transform turns an identification scheme into a signature scheme. A famous example that followed this design paradigm is the Picnic [CDG+17] signature scheme which made it into the third round of the NIST competition. Fiat-Shamir also has shown to be quite popular among submissions to the NIST signature on-ramp. Amongst the round-2 KpqC candidates, it is used by AIMer, HAETAE, and NCC-Sign.

Fiat-Shamir's core idea is to use the identification scheme in a non-interactive way to sign messages. To that end, the signer acts as the prover in the identification scheme, first choosing a commitment. The signer then uses the identification scheme in a non-interactive way: instead of having some verifier pick a challenge, the signer feeds the to-be-signed message (and all previous communications) through a hash function and takes the output as the challenge. The signer then computes a response and returns as its signature the commitment and the response.

The Fiat-Shamir with aborts paradigm additionally introduces rejection sampling. This is done to render the distribution of signatures sufficiently independent of sensitive information (the secret key), so that observing exchanged signatures will not help an attacker with forging a signature. Rejection sampling slightly complicates security proofs, in particular when concerning quantum attackers: two papers at CRYPTO 2023 [BBD+23, DFPS23] pointed out a flaw in the security proof of Dilithium. This flaw was directly tied to how the proof addressed rejection sampling. The papers also showed how to patch the proof by adapting the proof technique.

2.4 General notes on round-2 C software

Except for REDOG (see Section 5.7), all round-2 submissions provided C code. We modified the code to run within the SUPERCOP framework from `https://bench.cr.yp.to`. The level of modifications necessary for this varied from one submission to another.

2.4.1 Correctness

SUPERCOP's tests detected bugs in some of the software. Software developers are encouraged to use `gcc` or `clang` with the compiler options `-g -fsanitize=address -Wall -Wextra` and investigate all resulting error messages, although `-Wall` and `-Wextra` very often have false positives.

Even when software passes all of SUPERCOP's tests, the software could be computing something different from the specified cryptosystem. (As an analogy, it was announced in 2018 on NIST PQC Forum that the official Dilithium software was actually implementing a breakable signature system different from Dilithium; and it was announced in 2019 on the same forum that the official Falcon software was actually implementing a breakable signature system different from Falcon.) Submitters are encouraged to write a Python implementation from the specification, and to check that the Python implementation produces the same results as the C implementations.

2.4.2 Protection against timing attacks

SUPERCOP's TIMECOP tool detected data flow from secrets to timings in the software for all submissions.[1] Experience suggests that such data flow is often a security problem. To illustrate the danger, we developed and posted a fast key-recovery attack against the software for one submission; see Section 6.5. We also posted patches to illustrate ways to remove such data flow. We checked that the patches did not affect checksums produced by SUPERCOP.

A recurring source of such data flow within the KpqC submissions is Fisher–Yates sampling, which randomly permutes an array by swapping the first position with a random position, then swapping the second position with a random position after the first, etc. This becomes much slower when it is implemented in constant time. Better alternatives include [Ber17], [FRL24], and [Ber24a].

One case where data flow from secrets to timings is *not* a security problem is the following subroutine in RSA key generation: choose a secret integer

[1]This seems difficult to reconcile with [CKK+23, Tables 8 and 9, "Const-time Test"].

p uniformly at random from an interval, and then, if p is not prime, reject p and start over. Similar rejection-sampling loops often appear in post-quantum cryptography. TIMECOP offers a `crypto_declassify` function that software can use to mark a rejection condition as public.

2.4.3 Benchmarks

For five of the KpqC submissions, namely AIMer, HAETAE, MQ-Sign, NCC-Sign, and NTRU+, we submitted modified software to include in subsequent SUPERCOP releases for testing and benchmarking on many machines. We also submitted various updates to this software. Results from each machine appear on `https://bench.cr.yp.to` once the machine has completed its SUPERCOP run. The graphs in Chapters 7 and 12 are from SUPERCOP version 20240909; some of these submissions to SUPERCOP had already appeared in SUPERCOP versions 20240625, 20240716, or 20240808.

By the benchmarking cutoff date for this report (September 2024), two KpqC submissions with C code, namely PALOMA and SMAUG-T, had not yet been submitted to SUPERCOP. In both cases, the latest available code had timing variations, and avoiding those timing variations required slowdowns or cryptosystem modifications. Also, in the case of PALOMA, it was clear that large speedups are possible compared to the latest available code. See Sections 4.6 and 6.5. The PALOMA and SMAUG-T teams indicated that updates were in progress (and some updates did appear subsequently).

An important warning regarding benchmarks is that differences in cycle counts are often outweighed by differences in communication costs. For example, on Intel's Golden Cove microarchitecture, pre-quantum X25519 DH uses about 2^{16} cycles (see `https://lib25519.cr.yp.to/speed.html`), while lattice systems often use far fewer cycles; but communicating a byte costs roughly 2^{11} times as much as a CPU cycle (see [Ber23a]), so sending a 1KB lattice key instead of a 32-byte ECC key costs the equivalent of roughly 2^{21} cycles. For the same reason, comparisons of the efficiency of post-quantum systems should account not just for differences in cycle counts but also for size differences.

2.4.4 Future speeds

Another important warning is that current speeds will often be superseded by future speeds (although obviously this is less important in cases where the current speeds are already satisfactory for applications).

Applying an "optimizing" compiler to reference software produces speeds that are often much slower than the speeds that users will see. Faster software is typically CPU-specific, labor-intensive to write, and often not finished within the limited time available for a competition. All of the benchmarks shown in this report are for faster software using AVX2 vector instructions, running on Intel/AMD CPUs that support those instructions; but further speedups should be expected even on those CPUs, with variations from one KpqC submission to another.

For each of the five submissions with software released in SUPERCOP, this report includes a table showing, for the AVX2 software for the smallest parameter set, the number of instructions used for key generation. There are also similar tables for encapsulation and decapsulation in the KEM case, and for signing and verification in the signature case.

Within each table, if an instruction is in function F, where F is called by function G, then the instruction is tallied in the table row for F, not the table row for G; for example, instructions used inside a Keccak subroutine will be tallied in the table row for the Keccak subroutine, not for the higher-level function calling that subroutine. The table rows add up to 100% of the instructions.

A CPU that runs at most (e.g.) 4 instructions per cycle, such as a CPU with Intel's Skylake microarchitecture, will have cycle counts at least 1/4 of the instruction count. Sometimes the cycle count is actually (e.g.) 1/3 of the instruction count because some instructions are waiting for others to complete, or because the CPU can run only 3 instructions per cycle for the type of instructions used in the software. Sometimes it is possible to rearrange instructions to avoid the first type of slowdown, or to select different instructions to avoid the second type of slowdown.

Some instructions are consumed by overhead rather than essential arithmetic. Often it is possible to modify software to reduce overhead. Sometimes it is possible to reduce the number of instructions used for essential arithmetic. Some cases where such speedups appear to be possible are noted later in this report. However, there are also many cases where current subroutines appear to be close to optimal (for AVX2), for example in Keccak computations.

Future readers are encouraged to check https://bench.cr.yp.to to see whether there are updates demonstrating higher performance.

Part I

Public-Key Encryption and Key-Establishment Algorithms

Chapter 3

NTRU+: Compact Construction of NTRU Using Simple Encoding Method

NTRU+ [KP22] is a lattice-based submission to the KpqC competition. It builds on the NTRU system introduced by Hoffstein, Pipher and Silverman in 1998 [HPS98] and many of the improvements since, in particular the NTTRU system introduced by Lyubashevsky and Seiler in 2019 [LS19]. We have analyzed this system in detail in the first round, for a more in detail analysis see the first report [CHH+23] and the bachelor thesis of Luc Steenbakkers [Ste23]. The main novelties to previous works are that the authors propose new transformations ACWC_2 and $\overline{\mathsf{FO}}^{\perp}$. The transformation ACWC_2 is used to guarantee that the worst-case decryption probability remains bounded. In contrast to previous works like [DHK+23] the transformation has a tight security reduction. It relies on injectivity, and message and randomness recoverability of the underlying PKE scheme. The $\overline{\mathsf{FO}}^{\perp}$ transform is a variant of the Fujisaki-Okamoto transform to turn an IND-CPA secure PKE into a IND-CCA2 secure KEM. This transformation uses a novel approach that promises a tight security reduction and faster running times since it does not rely on re-encryption as previous works. This is a unique feature. The authors additionally present a direct transformation from an IND-CPA secure PKE to IND-CCA2 secure PKE that likewise does not use re-encryption. The new FO variants rely on rigidity and injectivity of the underlying IND-CPA secure PKE.
This chapter starts with a short summary of changes made after round 1. Following this is the system description, an in-depth look at their security proofs and an analysis of their software.

3.1 Changes from round 1

There are several changes from the version of round 1:
Importantly, there have been changes in how the SOTP transformation is defined, addressing an attack described by Lee [Lee23a]. Apart from this, the authors have added an IND-CCA version of a PKE (NTRU+PKE) in Section 5 based on NTRU+ in addition to the IND-CCA secure KEM version (NTRU+KEM) in Section 4. There are some differences between these versions. We assume the Section 5 version is the version implemented (and thus should be considered), since they contain steps pertaining directly to efficient implementation. Importantly, from a provable security point of view, the definition of injectivity has been changed to a computational requirement. Likewise, the definition of rigidity has been changed to now not quantify over all variables anymore. To address multi-target attacks, the authors have claimed to also introduce a new slightly different key derivation. However, this is neither given in the actual descriptions of the schemes (Figure 10, Figure 11, or Figure 15) nor when the authors apply their basic PKE scheme to the more classical FO transform (Figure 10).

3.2 System description

We will first describe the more mathematical part of the key exchange, this is copied from the round 1 report [CHH+23], and ends with a short list of interesting choices made in the specification of the NTRU+KEM in Section 8.
NTRU+ works with three rings

$$\begin{aligned} R &= \mathbf{Z}[x]/(x^n - x^{n/2} + 1) \\ R_q &= (\mathbf{Z}/q\mathbf{Z})[x]/(x^n - x^{n/2} + 1) \\ R_3 &= (\mathbf{Z}/3\mathbf{Z})[x]/(x^n - x^{n/2} + 1), \end{aligned}$$

where n and q are integers with $\gcd(q, 3) = 1$ and $n = 2^i 3^j$, with $i, j > 0$ to ensure n is even. This is a large deviation from the original NTRU, and follows the NTTRU paper by choosing a ring and values n and q such that the Number Theoretic Transform (NTT) can be computed efficiently.
The key generation works as follows. The sparse polynomials f' and g are generated using the Centered Binomial Distribution, which means the probability for a coefficient to be -1 or 1 is $1/4$ for both, and the probability for a 0 is $1/2$. Using this distribution approximates a narrow discrete Gaussian distribution.

The secret key is then created as $f = 3f' + 1$ and g. This shape of f goes back to a paper by Hoffstein and Silverman [HS01] and is also used in NTTRU. It gives the benefit of saving one division by f mod 3 in the decryption/decapsulation process. This does mean q needs to be somewhat larger to avoid decryption failures since the coefficients of f are larger.
The public key then is computed as $h = 3g/(3f' + 1) \bmod q$.
In encapsulation, a random message $m^+ \in \{0, 1\}^n$ is sampled. The shared symmetric key K and the randomness r are then generated by hashing m^+. Then m is created as an element of $\{-1, 0, 1\}^n$ using the Semi-generalized One-Time pad (SOTP) introduced by the authors. So $m = \mathrm{SOTP}(m^+, \mathrm{hash}(r))$, which basically splits up the hash of r into $u_1||u_2$ and then finds $m = (m^+ \oplus u_1) - u_2$. Afterwards the encryption of the message m is similar as in NTRU:

$$c = r \cdot h + m \in R_q.$$

Decapsulation starts with the following calculation:

$$m' = (f \cdot c \bmod {}^{\pm}q) \bmod {}^{\pm}3,$$

where mod$^{\pm}$ indicates that the set of representatives of the residue classes is taken centered around 0, e.g., mod $^{\pm}3$ means a result in $\{-1, 0, 1\}$. Note that the mixing of moduli can lead to decryption errors here. Then r' is obtained by $r' = (c - m') \cdot h^{-1}$. If $m = m'$ and $r = r'$, then using m' and r' as input to the inverse function of the SOTP, named Inv, will return the message m^+ (by splitting up the hash of r' again into u_1' and u_2' and computing $m'^+ = (m' + u_2') \oplus u_1'$). Given m'^+ the user can compute $\mathrm{hash}(m'^+)$ and compare this to r'. If encapsulation was done correctly and no decryption error appeared, $m'^+ = m^+$ and thus $r' = \mathrm{hash}(m'^+)$. If this matches, the user can compute K.
Interesting aspects of the Section 8 NTRU+KEM specification, found in Algorithms 15–17, are:

- Polynomials are when possible mapped using a mapping $NTT()$ to the specific product ring, explanation on this can be found in Appendix B of the submission.

- The public polynomial h is multiplied often with 2^{16} to account for the Montgomery reduction. This is not explained further, and could use more detail.

- Anything that is being sent in this protocol is encoded (and then later decoded) to a certain length to ensure efficiency when implemented with the AVX2 instruction set.

- Note that the decryption error check is not done until after the message was hashed. This looks like a method to hide the exact reason for a decryption error since there is now only a single point where the error symbol is output. However, usage of implicit rejection, where random values are returned in case of an error symbol, could improve this situation further since if an error occurs at all would additionally be hidden from certain attackers. It is unclear why implicit rejection has not been considered over explicit rejection.

3.3 Security considerations

In round 1 we have analyzed this system to check applicability of known attacks. For a more extensive summary we refer you to the report of the first round [CHH+23].
As seen in the first round the known key-search attacks (like meet-in-the-middle) do not detract from the security of NTRU+. Also the structural attacks considered (evaluate-at-1 attack and general lattice attacks) did not result in any viable attacks. For the general lattice attacks we did use another lattice estimator than the one used by the authors, but the differences are negligble. This fits with the state-of-the-art lattice estimators at the moment. The new lattice estimations we have done can be found in Table 3.1. These results are only at most a few bits off from the reported security levels.

Version	Primal Core-SVP	BDD
NTRU+576	116	137
NTRU+768	166	184
NTRU+864	191	208
NTRU+1152	268	281

Table 3.1: Estimated security levels for NTRU+ based on the lattice estimator [APS15]

3.4 Provable security

The authors of NTRU+ [KP22] chose to use a new two-stage IND-CCA2 transformation instead of deploying existing adaptations [DHK+21, SXY18, HHK17] of the Fujisaki–Okamoto transform [FO99]. A first step checks that the plaintext is in $\{0,1\}^*$ and then a (relatively standard) transformation is used.

The attack sketched by Lee [Lee23a], at least in the reference implementation, leads to a practical attack. Following an initial misunderstanding the NTRU+ submitters have now acknowledged the attack and that it also applies to the specification.

3.4.1 General remarks

The existence of Lee's attack points to a flaw in the security proof of IND-CCA security. Our understanding is that the attack is related to the fact that in the specification (and implementation) NTRU+'s algorithms do not always test membership of the received values in the specified sets. In the cryptographic literature this is not untypical. Often this omission is made to concentrate on core design features. But it is a very common source of errors and misleading claims. We highly suggest that the authors will employ a much more rigorous way of indicating which sets are efficiently recognizable and for which values the algorithms will actually call such an algorithm. We stress that proper membership tests may increase the runtime of all algorithms.
Another general issue is that in the definitions, it is sometimes unclear over what probability space the probabilities are taken exactly. In particular, when $b \leftarrow \mathsf{Alg}(i)$ is a probabilistic algorithm that takes input i to produce output bit b the probability $p = \Pr[b = 1]$ needs specification over what exactly the probability is taken. Usually this is over the random coins used by Alg. If now we use the equivalent $\mathsf{Alg}(i; r)$ expression with explicit random coins r, we need to specify how r is drawn then. In particular, Alg is now a deterministic algorithm and thus it is unclear what p should mean if the probability is specified over all r (so there is no random experiment at all).
In Algorithm 16 of [KP22] (the encapsulation algorithm), we observe that in line 2 the session key and the random coin are outputted hashing the hashed public key together with the message. However, in the pseudocode described in Figure 27 (encapsulation algorithm for NTRU+KEM), we observe that in line 2 the same output just needs the hashing of the message. Such a difference between the cryptographic description and implementation should be clarified.

3.4.2 FO transform without re-encryption

An interesting idea that is introduced by the authors of NTRU+ is a FO transformation that does not use re-encryption. Not re-encrypting improves the overall performance of NTRU+ compared to exsiting techniques that rely on re-encrypting the message [HHK17]. See Figures 11 and 12 in [KP22] for an overview of both FO transforms (used in the decapsulation algorithms). The authors of NTRU+ provide an argument, see Lemma 4.3 in [KP22], on why both decapsulation algorithms have the same output distribution except with negligible probability. The proof of Lemma 4.3 seems to be incomplete. Let us sketch the issue.

To complete the proof of Lemma 4.3, the authors have to compare the equations (i) $c \overset{?}{=} \mathsf{Enc}(\mathsf{pk}, m, r)$ (with re-encryption) and (ii) $r' \overset{?}{=} r$ (without re-encryption). For (ii), there is only a single r that fulfills the equation. However, for (i), there could be many pairs (m, r) that fulfill the equation. To make the output distributions equal, except with negligible probability, it seems that (i) should only hold for a single pair (m, r). To argue for this, the authors try to exploit the collision resistance property, which intuitively says that it should be hard for an attacker to find (m, r) and (m', r') such that both map to the same ciphertext c. In contrast to previous versions of the NTRU+ document, this now correctly takes care of the adaptive freedom the attacker obtains in the IND-CCA2 security game, where it can create arbitrary ciphertexts that are sent for decapsulation. Also, the authors argue that this form of collision-resistance is guaranteed under the assumption that the underlying generic NTRU system is IND-PCA secure. So indeed according to the authors, collision-resistance holds only computationally. However, the proof of Lemma 4.3 does not reflect this right now and needs to be revised. The stated bounds rather indicate that collision-resistance holds statistically (as it was the case in a previous version). The proof needs to be fixed in the ROM and the QROM. To fix the proof the authors have to show that any attacker against the IND-CCA2 security of NTRU+ can be used to break collision-resistance, while successfully simulating the decryption oracle. In particular, they need to show that any such IND-CCA2 attacker can be used to ultimately *extract* a collision.

3.4.3 Using Public Keys to derive shared secrets

Although the cryptographic description does not make this explicit at all times, NTRU+ uses the hashed public key of the receiver as an input to derive the shared key. We question this approach and suggest to consider using just the prefix of the public key instead as described in [DHK+21]. This saves computational resources, specifically for encapsulation since the entire public key now does not be hashed at all. When additionally introducing implicit rejections (that further protect against timing attacks), NTRU+ could rely on the theorems in [DHK+21]. As a result, the proof does not have to rely on the spreadness of the underlying IND-CPA secure scheme.

	doc Intel enc	S Intel enc	clean AMD enc	S AMD enc	doc Intel dec	S Intel dec	clean AMD dec	S AMD dec
ntruplus576	23000	23760	55179	8846	14000	16973	19814	9391
kyber90s512		30343		12369		21351		11875
ntruplus768	30000	30700	37193	12789	19000	22458	17330	13280
ntruplus864	32000	33333	51431	13553	21000	24449	19302	14391
ntruplus1152	39000	41890	38796	17137	25000	30507	24911	17883
kyber90s768		40816		19463		29347		18639
kyber90s1024		54227		28892		40491		27957
kyber512		35851		34980		28097		28322
kyber768		53692		53457		42172		44178
kyber1024		73998		76584		60651		64786

Table 3.2: Cycle counts for NTRU+ and Kyber collected by SUPERCOP ("S") on Intel Skylake and AMD Zen 2, compared to cycle counts in the NTRU+ documentation ("doc") for Intel Coffee Lake and NTRU+ cycle counts from KpqClean ("clean") for AMD Zen 2. Table is sorted by the last column.

3.5 Round-2 C software

The round-2 software provides a KEM and a PKE. The PKE appears to take only a limited-length message as input (like the PKEs inside various other KEMs, although PKEs vary in whether they aim for CCA security). Our investigations of the NTRU+ software have focused on the KEM.
The sizes of keys etc. in the round-2 software match the documentation.
Modifying the software for this submission to pass TIMECOP was easier than for any of the other submissions. Specifically, there is a rejection-sampling loop in key generation, and simply using `crypto_declassify` to mark this loop makes the software pass TIMECOP.
SUPERCOP's tests identified a correctness issue specifically for the `avx2` implementation of `ntruplus864` under some compilers, reporting that "`crypto_kem_dec returns nonzero`". The same implementation worked with other compilers.
We submitted this software for inclusion in the 25 June 2024 release of SUPERCOP. The NTRU+ team announced a software update on 25 July 2024 (compatible with previous NTRU+ software), and we submitted a corresponding update for the 8 August 2024 release of SUPERCOP.
Table 3.2 displays some of the resulting cycle counts. For comparison, the table also shows Kyber cycle counts. NTRU+ provides better tradeoffs than

Kyber between speed and claimed security level. Note, however, that the main costs for both `ntruplus` and `kyber` come from communicating lattice keys and lattice ciphertexts.
The two microarchitectures selected for SUPERCOP columns in the table are designed for comparability to two previous benchmark reports for NTRU+. First, the table copies cycle counts from the NTRU+ documentation for an Intel Core i7-8700K (Coffee Lake, which is practically identical to Skylake). Compared to SUPERCOP, the documentation reports cycle counts that are 5–10% smaller for enc and close to 20% smaller for dec. Some expected effects that could help explain this are as follows:

- The NTRU+ documentation does not comment on Turbo Boost, so presumably Turbo Boost was enabled. An Intel Core i7-8700K can boost by 27% (from 3.7GHz to 4.7GHz) depending on what operations are being run.

- SUPERCOP uses a particularly efficient RNG, which should save time for enc.

Second, the table copies AVX2 benchmarks from an AMD Zen 2 in `https://github.com/kpqc-cryptocraft/KpqClean_ver2` ("Environment2" is a Ryzen 7 4800H). Those benchmarks are larger than the SUPERCOP numbers for dec and much larger for enc. It turns out that those benchmarks used a slow reference implementation of SHA-256, rather than optimized SHA-256 software.
Tables 3.3, 3.4, and 3.5 show the number of instructions used for key generation, encapsulation, and decapsulation respectively inside AVX2 software for `ntruplus576`. In each case, more than 50% of the instructions are used inside SHA-256 computations. The NTRU+ software calls SHA-256 functions from OpenSSL, and OpenSSL automatically uses SHA-256 instructions on CPUs supporting those instructions (Intel Ice Lake and newer; AMD Zen); these tables are from AMD Zen 2.

25514	53.27%	`sha256-x86_64.s:sha256_block_data_order_avx2`
390	0.81%	`wmmintrin.h:aes256ctr_prf`
183	0.38%	`chacha.S:crypto_stream_chacha20_moon_avx2_64_constbranchindex_blocks_avx2`
182	0.38%	`x86_64cpuid.s:OPENSSL_cleanse`
145	0.30%	`lhash.c:ossl_lh_strcasehash`
145	0.30%	`memmove-vec-unaligned-erms.S:__memcpy_avx_unaligned_erms`
138	0.29%	`emmintrin.h:aes256ctr_prf`
128	0.27%	`malloc.c:_int_free`
125	0.26%	`kem.c:crypto_kem_keypair`
115	0.24%	`memset-vec-unaligned-erms.S:__memset_avx2_unaligned_erms`
98	0.20%	`ctype.c:ossl_tolower`
95	0.20%	`evp_fetch.c:evp_generic_fetch`
94	0.20%	`lhash.c:getrn`
88	0.18%	`md32_common.h:SHA256_Final`
88	0.18%	`o_str.c:OPENSSL_strcasecmp`
82	0.17%	`malloc.c:malloc`
77	0.16%	`digest.c:evp_md_init_internal`
73	0.15%	`knownrandombytes.c:randombytes`
72	0.15%	`aes256ctr.c:aes256ctr_prf`
58	0.12%	`emmintrin.h:aes256ctr_init`
56	0.12%	`xmmintrin.h:aes256ctr_init`
55	0.11%	`md32_common.h:SHA256_Update`
53	0.11%	`property.c:ossl_method_store_cache_get`
19838	41.42%	others
47893	100%	

Table 3.3: Instructions used for AVX2 software for `ntruplus576` key generation. Each table row tallies the number of instructions used directly by one function. If function F calls function G then instructions inside G are tallied for G, not for F.

51028	71.12%	`sha256-x86_64.s:sha256_block_data_order_avx2`
2868	4.00%	`sha512-x86_64.s:sha512_block_data_order_avx2`
562	0.78%	`x86_64cpuid.s:OPENSSL_cleanse`
435	0.61%	`lhash.c:ossl_lh_strcasehash`
384	0.54%	`malloc.c:_int_free`
336	0.47%	`wmmintrin.h:aes256ctr_prf`
306	0.43%	`lhash.c:getrn`
297	0.41%	`chacha.S:crypto_stream_chacha20_moon_avx2_64_constbranchindex_blocks_avx2`
294	0.41%	`ctype.c:ossl_tolower`
285	0.40%	`evp_fetch.c:evp_generic_fetch`
277	0.39%	`memmove-vec-unaligned-erms.S:__memcpy_avx_unaligned_erms`
264	0.37%	`o_str.c:OPENSSL_strcasecmp`
259	0.36%	`memset-vec-unaligned-erms.S:__memset_avx2_unaligned_erms`
246	0.34%	`malloc.c:malloc`
231	0.32%	`digest.c:evp_md_init_internal`
176	0.25%	`md32_common.h:SHA256_Final`
159	0.22%	`property.c:ossl_method_store_cache_get`
159	0.22%	`digest.c:EVP_Digest`
153	0.21%	`sparse_array.c:ossl_sa_get`
150	0.21%	`emmintrin.h:aes256ctr_prf`
150	0.21%	`pthread_rwlock_common.c:pthread_rwlock_unlock@@GLIBC_2.34`
146	0.20%	`sha512.c:SHA512_Final`
141	0.20%	`digest.c:EVP_DigestFinal_ex`
12445	17.34%	others
71750	100%	

Table 3.4: Instructions used for AVX2 software for `ntruplus576` encapsulation. Each table row tallies the number of instructions used directly by one function. If function F calls function G then instructions inside G are tallied for G, not for F.

25514	52.39%	`sha256-x86_64.s:sha256_block_data_order_avx2`
2868	5.89%	`sha512-x86_64.s:sha512_block_data_order_avx2`
1109	2.28%	`poly.c:poly_crepmod3`
380	0.78%	`x86_64cpuid.s:OPENSSL_cleanse`
336	0.69%	`wmmintrin.h:aes256ctr_prf`
290	0.60%	`lhash.c:ossl_lh_strcasehash`
256	0.53%	`malloc.c:_int_free`
212	0.44%	`lhash.c:getrn`
205	0.42%	`memset-vec-unaligned-erms.S:__memset_avx2_unaligned_erms`
196	0.40%	`ctype.c:ossl_tolower`
195	0.40%	`verify.c:verify`
190	0.39%	`evp_fetch.c:evp_generic_fetch`
176	0.36%	`o_str.c:OPENSSL_strcasecmp`
164	0.34%	`malloc.c:malloc`
154	0.32%	`digest.c:evp_md_init_internal`
150	0.31%	`emmintrin.h:aes256ctr_prf`
146	0.30%	`sha512.c:SHA512_Final`
128	0.26%	`kem.c:crypto_kem_dec`
123	0.25%	`memmove-vec-unaligned-erms.S:__memcpy_avx_unaligned_erms`
106	0.22%	`property.c:ossl_method_store_cache_get`
106	0.22%	`digest.c:EVP_Digest`
102	0.21%	`sparse_array.c:ossl_sa_get`
100	0.21%	`pthread_rwlock_common.c:pthread_rwlock_unlock@@GLIBC_2.34`
15497	31.82%	others
48703	100%	

Table 3.5: Instructions used for AVX2 software for `ntruplus576` decapsulation. Each table row tallies the number of instructions used directly by one function. If function F calls function G then instructions inside G are tallied for G, not for F.

Chapter 4

PALOMA: Binary Separable Goppa-based KEM

PALOMA [KJKK22] is a code-based KEM, based on binary Goppa codes. While PALOMA is close to the NIST submission Classic McEliece [ABC+22] and references it frequently, there are several differences.
The Goppa polynomial g is chosen to split completely over $\mathbb{F}_{2^m}$ while it is chosen to be irreducible in Classic McEliece. This means that the support and the t roots of g need to share $\mathbb{F}_{2^m}$ and the parameters are chosen so that $n + t < 2^m$ and for the given parameters this is a strict inequality. There are also some other differences in how the system achieves CCA security.
Below, we first discuss cryptanalysis results regarding PALOMA. Afterwards, we discuss the provable security claims made in the PALOMA specification and implementation considerations.

4.1 System description

Let $q = 2^m$. A binary Goppa code is defined by

- a list $L = (\alpha_1, \ldots, \alpha_n)$ of n distinct elements in $\mathbb{F}_q$, called the *support*.
- a square-free polynomial $g(x) \in \mathbb{F}_q[x]$ of degree t such that $g(\alpha_i) \neq 0$ for all $1 \leq i \leq n$. This $g(x)$ is called the *Goppa polynomial*.

The corresponding binary Goppa code $\Gamma(L, g)$ is

$$\left\{ \mathbf{c} \in \mathbb{F}_2^n \,\middle|\, S(\mathbf{c}) = \frac{c_1}{x - \alpha_1} + \frac{c_2}{x - \alpha_2} + \cdots + \frac{c_n}{x - \alpha_n} \equiv 0 \bmod g(x) \right\}$$

- This code $\Gamma(L, g)$ has length n, dimension $k \geq n - mt$ and minimum distance $d \geq 2t + 1$.

A parity-check matrix of this code is

$$H' = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ \alpha_1 & \alpha_2 & \alpha_3 & \cdots & \alpha_n \\ \alpha_1^2 & \alpha_2^2 & \alpha_3^2 & \cdots & \alpha_n^2 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \alpha_1^{t-1} & \alpha_2^{t-1} & \alpha_3^{t-1} & \cdots & \alpha_n^{t-1} \end{pmatrix} \cdot \begin{pmatrix} \frac{1}{g(\alpha_1)} & 0 & 0 & \dots & 0 \\ 0 & \frac{1}{g(\alpha_2)} & 0 & \dots & 0 \\ 0 & 0 & \frac{1}{g(\alpha_3)} & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & \frac{1}{g(\alpha_n)} \end{pmatrix}$$

and AH' for any invertible matrix A defines the same code. The parity-check matrix defined by having the coefficients of $(x - \alpha_i)^{-1} \bmod g$ in the i-th column is AH', where

$$A = \begin{pmatrix} g_1 & g_2 & \cdots & g_t \\ g_2 & g_3 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ g_t & 0 & \cdots & 0 \end{pmatrix}$$

and PALOMA uses this matrix $H = AH'$ as the secret parity-check matrix. PALOMA chooses $g(x) = \prod_{\alpha \in T}(x - \alpha)$ for $T \subseteq \mathbb{F}_q \setminus \{\alpha_1, \alpha_2, \dots, \alpha_n\}$ with $|T| = t$. Hence, $g(x)$ splits completely over $\mathbb{F}_q$. Classic McEliece chooses g irreducible over $\mathbb{F}_q$. This difference impacts what decoding algorithms are readily available and the submission puts significant effort into extending Patterson's decoding algorithm to deal with reducible g, using an approach from [BN19]. It is worth highlighting that [BN19] targeted random separable polynomials; the choice of g and handling of how the support and the roots of g share $\mathbb{F}_{2^m}$ are new to PALOMA.
The second-round submission is more careful in defining the secret key and how seeds turn into parts of the key. It also includes an additional seed for handing implicit rejection.
The separable Goppa code is generated from some seed r_C, this code is then scrambled by right multiplying with a permutation matrix P and then attempting to bring it to systematic form. The permutation matrix is generated from a seed $r_{\tilde{C}}$ and if the resulting HP cannot be brought into systematic form only the second stage repeats with a new choice of $r_{\tilde{C}}$ until this succeeds. The initial seed r_C is not stored and instead the support L and Goppa polynomial g are part of the secret key; $r_{\tilde{C}}$ is part of the secret key as is the matrix S^{-1}, where S satisfies that SHP is in systematic form. Finally, another random seed r is sampled which is included in the secret key to be used for implicit rejection.

1. Sample $r_C \in \{0,1\}^{256}$.
2. $(\alpha_1, \alpha_2, \ldots, \alpha_q) = \text{SHUFFLE}_{r_C}(\mathbb{F}_q)$
3. $L = (\alpha_1, \alpha_2, \ldots, \alpha_n),\ T = (\alpha_{n+1}, \alpha_{n+2}, \ldots, \alpha_{n+t})$.
4. Compute $g = \prod_{\alpha \in T}(x - \alpha)$ and corresponding parity-check matrix H.
5. Sample $r_{\tilde{C}} \in \{0,1\}^{256}$
6. Compute $n \times n$ permutation matrix P using seed $r_{\tilde{C}}$.
7. Try to bring HP to systematic form, GOTO 5 if this fails.
8. Let $\hat{H} = SHP$ in systematic form.
9. Sample $r \in \{0,1\}^{256}$.
10. Output $\hat{H}$ as public key and $(L, g, S^{-1}, r_{\tilde{C}}, r)$ as secret key.

HP can be put in systematic form with 29% probability (no change compared to Classic McEliece). For PALOMA, any choice of T leads to valid g; Classic McEliece has a procedure to construct an irreducible g that succeeds almost always.
Binary Goppa codes are alternant codes and can be decoded efficiently using the Patterson decoder [Pat75], however, this decoder requires computing the inverse of the syndrome polynomial modulo g and there is no reason a-priori that this polynomial should be co-prime to g in the case considered in PALOMA. The submission thus presents a version of Patterson decoding that is suitable for this choice of g. We first describe the regular version and then point out where changes were needed.
To decode the received vector $\mathbf{x} = \mathbf{c} + \mathbf{e}$ to $\mathbf{c} \in \Gamma(L, g)$, first compute the syndrome polynomial

$$s(x) = \sum_{i=1}^{n}(c_i + e_i)/(x - \alpha_i) \equiv \left(\sum_{i=1}^{n} e_i \prod_{j \neq i}(x - \alpha_j)\right) / \prod_{i=1}^{n}(x - \alpha_i) \bmod g(x).$$

If $\mathbf{e} \neq 0$ this polynomial $s(x) \neq 0$ by definition of the code.
Put $f(x) = \prod_{i=1}^{n}(x - \alpha_i)^{e_i}$ with $e_i \in \{0,1\}$. Then, using the chain and product rules on derivatives, $f'(x) = \sum_{i=1}^{n} e_i \prod_{j \neq i}(x - \alpha_j)^{e_j}$. Thus $s(x) \equiv f'(x)/f(x) \bmod g(x)$.
Split $f(x)$ into odd and even terms: $f(x) = A^2(x) + xB^2(x)$ and observe that over binary fields $f'(x) = B^2(x)$ as all even powers of x have derivative 0.

Thus $B^2(x) \equiv f(x)s(x) \equiv (A^2(x) + xB^2(x))s(x) \bmod g(x)$ which Patterson normally transforms by dividing by $s(x) \bmod g(x)$. However, as mentioned above, $s(x)$ need not be co-prime to $g(x)$ if g is reducible. If $g(x)$ is irreducible it is possible to compute $B^2(x)(x + 1/s(x)) \equiv A^2(x) \bmod g(x)$ and to recover A and B from a half-gcd computation on the polynomials $v(x) \equiv \sqrt{x + 1/s(x)}$ and $g(x)$. Note that this computation is the first half of inverting v modulo g, and co-primality is ensured for irreducible g as well. At every step in the computation of the extended Euclidean algorithm, $A(x) = B(x)v(x) + h(x)g(x)$ and the half-gcd computation stops when the degrees of A and B are balanced: $\deg(A) \le \lfloor t/2 \rfloor, \deg(B) \le \lfloor (t-1)/2 \rfloor$.
PALOMA uses the generalized version of Patterson's decoder for reducible g from [BN19], dealing with non-constant $\gcd(g, s)$.
Let $\tilde{s} = 1 + xs$ and $g_1 = \gcd(g, s), g_2 = \gcd(g, \tilde{s})$. By construction, s and $\tilde{s}$ are co-prime. Define g_{12} via $g = g_1 g_2 g_{12}$, and observe that $A = ag_2$ and $B = bg_1$ for some polynomials a, b. One then has $b^2 u \equiv a^2 \bmod g_{12}$, where $u = (g_1/g_2)^2 \tilde{s}/s \bmod g_{12}$. Finally, they compute a partial gcd of $\sqrt{u}$ and g_{12} (accounting for bounds on the degrees of a and b) and recombine the result to $f(x) = a^2 g_2^2 + b^2 g_1^2 x$.
To hide timing information on secret g they would need to use g_1, g_2, g_{12} of maximum possible degree t and thus require more work and extra effort to hide the actual degrees. So far, this protection against timing attacks is not implemented but decoding is already very slow (see Section 4.4).
Classic McEliece uses a Berlekamp decoder instead of the Patterson decoder. This decoder does not require g to be irreducible and could thus be used for the g in PALOMA.
The main observation for using this decoder is that $\mathbf{c} \in \Gamma(L, g)$ implies $\mathbf{c} \in \Gamma(L, g^2)$. This fact is normally shown when proving that the minimum distance is at least $2t + 1$.
Let $\mathbf{c} \in \Gamma(L, g)$, then

$$s(x) = \sum_{i=1}^{n} c_i/(x - \alpha_i) = \left(\sum_{i=1}^{n} c_i \prod_{j \ne i} (x - \alpha_j) \right) / \prod_{i=1}^{n} (x - \alpha_i) \equiv 0 \bmod g(x).$$

over $\mathbb{F}_{2^m}$:

$$(f_{2i+1} x^{2i+1})' = f_{2i+1} x^{2i}, \quad (f_{2i} x^{2i})' = 0 \cdot f_{2i} x^{2i-1} = 0,$$

thus

$$f'(x) = \sum_{i=0}^{(w-1)/2} f_{2i+1} x^{2i} = \left(\sum_{i=0}^{(w-1)/2} \sqrt{f_{2i+1}} x^i \right)^2 = F^2(x).$$

Having $s(x) \equiv F^2(x)/f(x) \equiv 0 \bmod g(x)$ for squarefree g means $g|F$, thus $g^2|F^2$. Note that $f(x)$ factors into linear terms and has as roots the support elements α_i in the positions i where the error occurred. Given that $g(\alpha_i) \neq 0$ for all i, f is invertible modulo g.
Let $\mathbf{c} \in \Gamma(L, g^2)$, then $s \equiv 0 \bmod g^2 \Rightarrow s \equiv 0 \bmod g$ holds obviously.

The Berlekamp decoder can be used for any generalized Reed-Solomon code and is used here for $\Gamma(L, g^2)$. This algorithm goes back to computing the feedback polynomial for an LFSR given twice as many output bits as the state length.

1. Let $\mathbf{v} = \mathbf{c} + \mathbf{e}$. Then
$$B(x) = \sum_{i=1}^{n} \frac{v_i}{g^2(\alpha_i)} \prod_{j \neq i} (x - \alpha_j).$$
In particular, $B(\alpha_i) = v_i \left(\prod_{j \neq i} (\alpha_i - \alpha_j) \right) / g^2(\alpha_i)$.

2. Put $A(x) = \prod_i (x - \alpha_i)$.

3. Use Berlekamp–Massey to compute approximant b/a to B/A such that $\gcd(a, b) = 1, \deg(a) \leq t, \deg(b) < t$, and $\deg(aB - bA) < \deg(A) - t$.

4. If a divides A, compute $f = B - bA/a$ and $\mathbf{v} = (B(\alpha_1) - f(\alpha_1), B(\alpha_2) - f(\alpha_2), \ldots, B(\alpha_n) - f(\alpha_n))$

Classic McEliece chooses the Berlekamp decoder for ease of safe implementation (correctness and constant timeness). The same approach could work for PALOMA and would probably be faster than their adaptation of Patterson. See [Ber24b] for a full explanation of the Berlekamp decoder for binary Goppa codes.
PALOMA highlights that it needs to use the full matrix $H = AH'$ for Patterson decoding and it includes S^{-1} in the secret key and the computation of $S^{-1}s$ before decoding. Classic McEliece uses instead that the public key defines the same code as H' and as H. The syndrome does indeed depend on the use of the exact parity check matrix, but the set of code words does not. Given that $\hat{H}$ is in systematic form, meaning that the first $n - k$ columns are an identity matrix, the syndrome $\mathbf{s}$ can be extended to a corresponding received word $\mathbf{v}$ by appending k zeros to $\mathbf{s}$. It is then easy to compute the corresponding syndrome $H'\mathbf{v}$ and use this in decoding. Note that due to the very simple structure of H' this does not require storing H' as a matrix, so the secret key need not include any matrices. See, e.g., [Ber24b, Section 8].

The use of P in PALOMA seems redundant with the earlier use of shuffle, which already gives a random permutation on $\mathbb{F}_q$. It is possible that the authors mean to save time by having to do the first 4 steps of KeyGen only once and having the loop for handling non-invertible matrices only apply to steps 5–7. However, these are the more expensive steps, and the split at step 5 means that the resulting codes are not uniformly random in the set of codes with completely-split g. The failure in step 7 depends not only on L, which effectively gets overwritten by the permuted version of it, but also on g, which remains part of the secret key. Hence, it needs to be considered whether this rejection sampling leaks information on g. We still do not see a valid attack but consider this a bit concerning.

4.2 Security considerations

For most purposes, the choice of g between PALOMA and Classic McEliece does not matter. While limiting g to polynomials that split completely over $\mathbb{F}_{2^m}$ limits the key space, this number is so large that key search is not even close to the fastest known attacks to recover messages. Within the keyspace covered by PALOMA there are several equivalent codes and an ongoing research project is to see whether there are proportionally larger equivalence classes. This work is currently under development by Lorenz Panny, a former TU/e PhD student.
A concern expressed in [For18] in comparing choices for NTS-KEM and Classic McEliece is that reducible choices of g can bring structural or algebraic attacks into reach. For algebraic attacks we checked [FOPT10, COT14, EM22, CMT23] and several more. Note, again, that the public key is a hidden parity-check matrix $\hat{H} = SAH'P$ of $\Gamma(L, g)$ for

$$H' = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ \alpha_1 & \alpha_2 & \alpha_3 & \cdots & \alpha_n \\ \alpha_1^2 & \alpha_2^2 & \alpha_3^2 & \cdots & \alpha_n^2 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \alpha_1^{t-1} & \alpha_2^{t-1} & \alpha_3^{t-1} & \cdots & \alpha_n^{t-1} \end{pmatrix} \cdot \begin{pmatrix} \frac{1}{g(\alpha_1)} & 0 & 0 & \dots & 0 \\ 0 & \frac{1}{g(\alpha_2)} & 0 & \dots & 0 \\ 0 & 0 & \frac{1}{g(\alpha_3)} & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & \frac{1}{g(\alpha_n)} \end{pmatrix}$$

Let $M = (m_{ij})$ be the generator matrix of the same code, then $\hat{H}M^T = 0$, hence

$$\sum_{j=1}^{n} m_{ij}\alpha_j^u / g(\alpha_j') = 0,\ 1 \le i \le k,\ 0 \le u \le t-1.$$

	doc	isd0	isd1	isd2
PALOMA-128	166.21	162.84	159.76	153.74
PALOMA-192	267.77	245.67	241.94	229.63
PALOMA-256	289.66	277.20	272.80	255.45

Table 4.1: PALOMA attack costs in $\log_2$(bit operations). The "isd0" column is the 0-level ISD (Prange + Lee–Brickell + Leon) cost predicted by CryptAttackTester. The "isd1" column is the 1-level ISD (Stern + Dumer) cost predicted by CryptAttackTester. The "isd2" column is the 2-level ISD (MMT + BJMM) cost predicted by CryptAttackTester. CryptAttackTester does not account for the costs of long-distance communication; the real-world speedup from isd1 to isd2 is smaller than the bit-operation speedup in the table. Computing this table took about a day on a dual AMD EPYC 7742: e.g., CryptAttackTester used 2800 core-hours searching many attack parameters for PALOMA-256. For comparison, the "doc" column is BJMM-ISD bit operations estimated on page 48 of the round-2 PALOMA documentation.

Use variables X_j for the unknown $\alpha'_j \in \mathbb{F}_{2^m}$ and Y_j for the unknown $1/g(\alpha'_j) \in \mathbb{F}_{2^m}$ for $1 \leq j \leq n$. Here we use α'_j to denote the j-th element after permutation using P. Given that that the KEM shared secret is defined with respect to this permuted order, it is the relevant target of the attacker. Using Gröbner basis computations to solve the system $\sum_{j=1}^{n} m_{ij} X_j^u Y_j = 0$ requires high rate (dimension divided by length of the code) and does not make use of g (Y is general). This means, that the difference of PALOMA and Classic McEliece is ignored in the modeling. None of the attacks we considered gave an improvement over message-recovery attacks and we have not yet found a modeling that makes use of the structure of g.
The state-of-the-art attack thus appears to be information-set decoding. See Table 4.1 for our estimates of the security levels of PALOMA-128, PALOMA-192, and PALOMA-256.
Note that these estimates differ from those presented in the submission. The full version of the CryptAttackTester paper [BC24] contains the full details for PALOMA and Classic McEliece.

4.3 Implementation considerations

PALOMA shares with Classic McEliece the large public keys and small ciphertexts. The PALOMA private key includes a matrix even through decoding algorithms for binary Goppa codes do not require this.
Requiring the roots of g and $\alpha_1, \ldots, \alpha_n$ to fit into $\mathbb{F}_{2^m}$ means that one cannot

take $n = 2^m$. This is not a large issue: previous efforts to obtain the best tradeoffs between security level and key size for the McEliece cryptosystem usually take n smaller than 2^m.

A benefit of PALOMA's key generation is that g can be sampled by sampling t random elements in $\mathbb{F}_{2^m}$ and defining them as roots of g. Classic McEliece instead defines g as the minimal polynomial over $\mathbb{F}_{2^m}$ of a random element of $\mathbb{F}_{2^{mt}}$, trying again on occasions when this polynomial does not have degree t.

To analyze the rejection probability for the minimal-polynomial procedure, note that the minimal polynomial over $\mathbb{F}_{2^m}$ of any element $\zeta \in \mathbb{F}_{2^{mt}}$ has degree d for some divisor d of t. If $d \neq t$ then $d \leq t/2$. There are 2^{md} monic polynomials of degree d, so there are at most 2^{md} monic irreducible polynomials of degree d, so there are at most $2^{md}d$ roots ζ of such polynomials in $\mathbb{F}_{2^{mt}}$. The actual number of such roots is closer to 2^{md}, but the upper bound $2^{md}d$ is good enough to see that rejections are vanishingly unlikely. For example, for $t = 64$, there are most 2^{32m+5} elements $\zeta \in \mathbb{F}_{2^{64m}}$ having $d = 32$, at most 2^{16m+4} elements $\zeta \in \mathbb{F}_{2^{64m}}$ having $d = 16$, etc., for an overall rejection probability at most $2^{-(32m-5)}+2^{-(48m-4)}+\cdots$. All proposals starting with [McE78] have $m \geq 10$.

The speed difference between the two methods of generating g is more obvious. A textbook computation of the minimal polynomial of ζ applies linear algebra to a $(t+1)\times t$ matrix over $\mathbb{F}_{2^m}$ obtained from $t+1$ powers of ζ, using about t^3 operations in $\mathbb{F}_{2^m}$, whereas a textbook computation of the product of $x-\alpha$ for t values of α takes only about t^2 operations in $\mathbb{F}_{2^m}$. The exponent 3 here can be improved, but the exponent 2 can also be improved; one cannot expect a degree-t minimal-polynomial computation to be competitive with multiplying t linear polynomials.

On the other hand, the main bottleneck in key generation is a different linear-algebra step to produce the public key, an $mt \times (n - mt)$ matrix over $\mathbb{F}_2$. A textbook computation uses on the order of $n^3 \approx m^3t^3$ operations in $\mathbb{F}_2$, whereas the t^3 operations in $\mathbb{F}_{2^m}$ use only about m^2t^3 operations in $\mathbb{F}_2$. The constants are also favorable to minimal-polynomial computation; for example, minimal-polynomial computation uses about 4% of the cycles for current key-generation software for `mceliece348864f`.

4.4 Round-1 benchmarks

In round 1, PALOMA advertised better speeds than Classic McEliece on an Apple M1. Our benchmarks found, however, that the Classic McEliece software is much faster than the PALOMA software; see Table 4.2. We

already included this in our final report for Round 1.
For round 2, PALOMA instead compares to Classic McEliece on an Apple M3. We do not have an M3 available for testing, but presumably the M3 speeds for Classic McEliece are at least as good as the M1 speeds shown in Table 4.2.
The implementation of decapsulation in PALOMA chose to use a new adaptation of Patterson's algorithm which seems slower than a direct reuse of the Berlekamp implementation in Classic McEliece. That system chose not to use Patterson: it is not clear that a constant-time implementation of Patterson's algorithm can outperform Berlekamp, even without the extra complications of adapting Patterson's algorithm to the case of reducible g. Adapting the Berlekamp software from Classic McEliece to PALOMA should produce large decoding speedups for PALOMA; See Section 4.6 for further discussion.
The implementation is additionally slower than necessary in the form of plaintext confirmation chosen to involve generating a matrix from a seed and doing a matrix multiplication. A simpler hash function call should suffice.
While we understand the rationale for choosing split g and have not been able to show any security degradation from this choice, we do not think the other choices (Patterson, use of S, e, e') are beneficial for speed (they look detrimental to it) nor do they achieve better security compared to alternatives.
There are some other quirks in PALOMA that produce slowdowns:

- KeyGen shuffles the columns of H even though the order in L was random already.
- The secret key includes S^{-1} with $\hat{H} = SHP$ in systematic form. They comment that key size could be saved as S depends on r_C and $R_{\tilde{C}}$, but miss that S is not necessary at all in decoding. Note that syndrome $\mathbf{s} \in \mathbb{F}_2^{n-tm}$ expands to noisy codeword $\mathbf{v} = \mathbf{s}00\ldots0$ for $\hat{H}$ in systematic form.

It would thus be easy to decrease the size of the private key by skipping S^{-1}.
We understand that, just like for Classic McEliece, PALOMA does not use only the seed as private key but keeps some parts expanded. This makes KeyGen and Decap a lot faster than if the full expansion was needed.
Further slowdowns come from the use of extra permutations in encapsulation and decapsulation.
We also observe that the PALOMA security analysis appears to assume (on page 51) that decapsulation checks that the recovered error vector has weight exactly t and that re-encrypting the recovered e produces the ciphertext. These checks do not appear in the PALOMA specification of decapsulation

	cycles	ms	doc
PALOMA-128 init	10342016	5.011	
PALOMA-128 keypair	154491948	74.851	64.00
libmceliece 348864f keypair	71569974	34.675	
PALOMA-128 enc	406522	0.197	0.03
libmceliece 348864f enc	19178	0.009	
PALOMA-128 dec	19770110	9.579	9.00
libmceliece 348864f dec	235124	0.114	
PALOMA-192 init	10329976	5.005	
PALOMA-192 keypair	646889506	313.415	261.00
libmceliece 460896f keypair	218800512	106.008	
PALOMA-192 enc	819752	0.397	0.04
libmceliece 460896f enc	40764	0.020	
PALOMA-192 dec	122390384	59.298	59.00
libmceliece 460896f dec	651966	0.316	
PALOMA-256 init	10331352	5.005	
PALOMA-256 keypair	630017080	305.241	323.00
libmceliece 6960119f keypair	368109154	178.347	
PALOMA-256 enc	1000180	0.485	0.04
libmceliece 6960119f enc	76454	0.037	
PALOMA-256 dec	123652778	59.909	60.00
libmceliece 6960119f dec	682152	0.331	

Table 4.2: Measurements of PALOMA vs. libmceliece speed, Apple M1, Icestorm core, gcc 13.2.0. The "cycles" column is the median of 31 measurements of the round-1 PALOMA software and of libmceliece 20230612. The "ms" column is milliseconds calculated from cycles and 2.064GHz clock speed. For comparison, the "doc" column is milliseconds reported on page 30 of the PALOMA documentation. The PALOMA documentation does not specify 3.2GHz Firestorm cores vs. 2.064GHz Icestorm cores, but the Icestorm dec measurements are a good match. The PALOMA documentation also reports milliseconds for an Intel Core i5; libmceliece uses 256-bit vectors on Intel and uses portable code on the M1, so the speed gap will be larger on Intel.

or in the PALOMA software. These checks are important for the applicability of the security proofs (see Section 4.5) and are important for security (see Section 4.7).

4.5 Provable security claims

The PALOMA specification provides a provable security result: to construct an IND-CCA secure Key Encapsulation Mechanism, PALOMA uses its own modified version of the Fujisaki-Okamoto (FO) design paradigm [FO99, HHK17], the paradigm that was also used by most of the NIST proposals (including the winner Kyber).
The core idea of the FO paradigm is to use a given public-key encryption scheme to encrypt a randomly chosen message, from which a key can be derived by feeding the message (and sometimes some auxiliary information) into a hash function. Intuitively, this makes the key unpredictable unless the attacker can break the encryption algorithm. To make chosen-ciphertext attacks unfeasible, the encryption algorithm is modified in a certain way. This modification prevents that attackers can build dishonest ciphertexts that will be accepted by the decapsulation algorithm.

4.5.1 Security proof of PALOMA in round 1

In our round-1 report we highlighted the following issues:
At a first glance, PALOMA-KEM is constructed from PALOMA-PKE by applying a variant of the FO design paradigm. To argue IND-CCA security of PALOMA-KEM, the specification relatively briefly recalls the security reasoning for FO that was given in [HHK17].
There are, however, several gaps in the proof: PALOMA-KEM deviates from the established FO paradigm in several different ways (detailed below). The submission does not address at all that/why it introduces these modifications and if/why they do not decrease security.
While it is not straightforward to assess whether/how these deviations lead to an attack, each one on its own makes it impossible to apply the established security proof.
Undesirable dependencies in the decapsulation algorithm. To deal with chosen-ciphertext attacks, FO-KEMs react to dishonest ciphertexts by returning a pseudorandom value. This pseudorandom value is derived from a secret random seed which is part of the secret key (besides the secret key needed for decrypting). The FO paradigm picks this seed independently from the secret key used for decrypting ciphertexts, unlike PALOMA-KEM,

which reuses a seed that was already used to generate the secret decryption key. This leads to undesirable leakage on the secret decryption key during decapsulations of dishonest ciphertexts, and leaves a gap in the security proof.

Plaintext permuting not covered by FO paradigm. Following the FO paradigm, one would expect that security of PALOMA-KEM is based on security of the encryption algorithm introduced as algorithm 6. Instead, it is based on a modification of algorithm 6, called algorithm 18, which introduces a permutation step. This leaves a gap in the security proof – to close this gap, it would have been necessary to show that security of algorithm 18 can be based on security of algorithm 6.

One possible explanation for this modification might be that the full FO design can only be applied to probabilistic schemes: it could be that the random sampling of the permutation matrix was introduced to make the encryption algorithm probabilistic. In this case, the submission could consider switching to FO-alternatives for deterministic schemes (e.g., [BHH+19]) instead.

Treatment of dishonest ciphertexts not covered by FO paradigm. As described above, dishonest ciphertexts are treated by FO-KEMs in a specific way to mitigate chosen-ciphertext attacks. To identify such dishonest ciphertexts, PALOMA-KEM deviates from the standard check imposed by the FO paradigm. This leaves a gap in the security proof – to close this gap, it would have been necessary to show that the alternative check performed by PALOMA-KEM is equivalent to the one that is imposed by the FO paradigm.

Additional gap - sampling of messages not covered by FO paradigm. FO-KEMs sample messages uniformly at random. This is needed to be able to base security of the KEM on security of the involved encryption algorithm – the security definition for the encryption algorithm assumes uniform messages. PALOMA-KEM instead samples messages using algorithm 13. This leaves a gap in the security proof – to close this gap, it would be necessary to analyze whether algorithm 13 yields the required uniform distribution. (See section 5.5.4 of [ABC+22] as an example for such a discussion.)

4.5.2 Security proof of PALOMA before the update in August 2024

The updated security reasoning for round 2 addressed some of the concerns brought up in round 1.

The updated specification resolved the undesirable dependency in the decapsulation algorithm (pseudo-random values leaking on the secret key) as PALOMA started using independent randomness r. Instead of using r itself as the seed for the pseudo-random values, r first gets expanded to an error

vector $\tilde{e}$. Using $\tilde{e}$ unifies how the two branches (valid/invalid ct) compute their keys with respect to the length/format of their hash inputs. Proposition 5.1 of the PALOMA security analysis indicates that there is no entropy loss.
The following subtlety arose in addressing the second issue:

- The IND-CCA transform used for PALOMA permutes the plaintexts. This is a deviation from 'standard FO' transforms. The motivation for this deviation is not stated explicitly. The deviation makes it necessary to give a dedicated security proof that closes the resulting gap in the security reasoning. The specification now included such a proof.

The following issues brought up in Round 1 so far were unaddressed:

- The identification of dishonest ciphertexts still deviated from the FO paradigm, and hence vulnerabilities against chosen-ciphertext attacks could not generically be ruled out. In more detail, it was not discussed or proven that the alternative ciphertext validity check does not introduce chosen-ciphertext vulnerabilities.

- It is was furthermore left unaddressed whether the message sampling is actually captured by the FO paradigm. (I.e., if the algorithm yields the required uniform distribution.)

4.5.3 Security proof of PALOMA after the update in August 2024

The updated security reasoning gives a new modular proof. Upon inspection, it is not clear why this is necessary. (I.e., why a security result cannot be obtained by simply applying [HHK17] and follow-ups.) At a first glance, it might furthermore be surprising that the section shows equivalence between OW-PCA and OW-CPA security, given that the KEM's security statement does not base security on OW-PCA security. We suspect a mismatch between what the theorem states and what its proof actually requires.
The ciphertext validity check that is used in place of the re-encryption check was modified by introducing an additional weight check, and the proof section now argues equivalence of the two checks, implicitly assuming that the decoder behaves perfectly. This assumption is not further studied.
The following concerns remain unaddressed:

- The message distribution that might be potentially problematic, in case messages (and thus the resulting session keys) can be predicted too easily.

- Giving security proofs against quantum attackers.

4.5.4 Interpretation of provable security results

For round 1

The formal security arguments discussed in subsection 4.5.1 do not seem sound, and have an additional significant shortcoming: the arguments only consider classical adversaries. This also means that the bounds do not apply against quantum adversaries that at least can gain a polynomial advantage using Grover.
A result might be obtained by switching to the design paradigm underpinning McEliece, and then using recent work on this design paradigm [BP18, BHH+19], but this would require a redesign of PALOMA.

For round 2

Our initial conclusion was as follows: "While we did not find proof that the introduced deviations create a security break, we cannot fully rule out that they might." We then found a chosen-ciphertext attack (see Section 4.7) exploiting one of the proof gaps. The August 2024 PALOMA update stopped that attack, but, because of the remaining deviations, we cannot rule out the possibility of further attacks. In any case, it seems that the deviations result in PALOMA requiring more running time than necessary, without explicitly highlighting the rationale/resulting benefits.

4.6 Round-2 C software

The software provides `w_openssl` and `wo_openssl` options, evidently referring to using or not using the OpenSSL library. Our investigations have focused on the `w_openssl` option.
The sizes of keys etc. in the round-2 software match the documentation.
The software has uninitialized hash output buffers in `utility.c`. We modified this in a way that seems to produce consistent output, but more work would be required to check that this matches the specification.
We noticed that the software exits when an internal vector has weight 0. We developed an attack that exploits this to recover the shared secret from a given PALOMA ciphertext by observing the receiver's reactions to tweaked versions of the ciphertext. See our KpqC-bulletin email from 13 Apr 2024 14:05:13 -0700. This security problem comes from a deviation between the

software and the specification, whereas our July 2024 IND-CCA attack (see Section 4.7) applies both to the software and the specification.

TIMECOP identifies many issues in the PALOMA software. Presumably these issues can be exploited to recover secret keys through timing, as in Section 6.5.

We highlight three sources of timing variations for PALOMA. First, permutations are carried out with Fisher–Yates sampling. Two ways to change the PALOMA specification to avoid this issue would be (1) replacing Fisher–Yates sampling with one of the alternatives mentioned in Section 2.4 or (2) eliminating the permutations as part of a broader redesign.

Second, the PALOMA decoding algorithm works with polynomials of variable degree (starting with the greatest common divisors on line 3 of Algorithm 3 in the specification). The easiest way to make the software run in constant time would be to carry out each operation to the maximum possible degree. This is feasible, but will take some effort and will produce some slowdown in decapsulation.

Third, multiplication of elements of $\mathbb{F}_{2^{13}}$ is carried out with lookups in some precomputed tables (e.g., a table of size 2^{14} containing all 7-bit-by-7-bit products). We provided a replacement multiplication function (KpqC-bulletin email from 26 Apr 2024 22:49:08 +0200) that avoids table lookups. This produced a 3× slowdown in decapsulation, but it is clear that more work on the software would produce speedups:

- About half of the operations in the replacement function are converting back and forth between packed 13-bit field elements and an unpacked format. If more of the code were written in the unpacked format then most of the conversions would disappear.

- Further speedups are possible from "bitslicing" arithmetic operations across CPU words, for example carrying out 64 bit operations in parallel with a single 64-bit operation. An initial analysis indicates that [Cho17] is applicable to PALOMA and would produce better speeds than the original PALOMA software while running in constant time.

- Many CPUs have instructions for binary-polynomial multiplication, and then a multiplication is just a few instructions. Typically these can again be carried out on more than one input at once.

There are some CPUs with variable-time multipliers; a further advantage to the bitslicing approach is that it will run in constant time on those CPUs.

We highlight that the speeds presented in Table 4.2 use the original round-1 software, not our constant-time-multiplication patch.

PALOMA is the only submission for which our analyses conclude that, with no changes to the specification, large software speedups are possible compared to the current software. This relies on extrapolation from the Classic McEliece software, where the major subroutines include optimizations that are applicable to the corresponding subroutines in PALOMA. In particular, for decapsulation, the Berlekamp decoder applies to any squarefree Goppa polynomial g, so it does not need any adjustments for the split polynomials used in PALOMA; and, as noted above, the techniques for fast arithmetic from [Cho17] should work for PALOMA. This does not mean that PALOMA will end up with identical speeds to Classic McEliece: some aspects of PALOMA are different from Classic McEliece, such as PALOMA's use of permutations. See generally Section 4.4.
The PALOMA team has indicated that a software update is in progress.

4.7 A chosen-ciphertext attack

As noted in Section 4.5, after identifying various proof gaps, we found a fast chosen-ciphertext attack exploiting one of those gaps. We announced the attack in the July 2024 KpqC workshop and in accompanying kpqc-bulletin email.
The attack recovers the session key (shared secret) given a ciphertext, a public key, and a decapsulation oracle for other ciphertexts. The attack takes 1 query to RO_G and $O(n)$ queries to the decapsulation oracle and to RO_H. The decapsulation oracle is used only for comparison to guessed session keys, so this attack can also be used as a reaction attack using observations of whether a server successfully responds to data that the attacker encrypted under those guessed keys.
We also implemented this attack, calling subroutines from the reference software. Our experiments worked for all 100 KATs for PALOMA-128, taking around 30 seconds for each session-key recovery on one core of a 3GHz Intel Skylake. We posted a demo script to reproduce the experiments. Tweaking the script to attack PALOMA-256 takes under 7 minutes per session-key recovery.
We had previously found and implemented a reaction attack against the software in the original submission package, exploiting a deviation of the software from the specification (see Section 4.6). That attack did not work against the specified version of PALOMA. The software deviation was fixed by an April 2024 software patch from the PALOMA team. The CCA attack demo works against the patched PALOMA software.
For the specified PALOMA KEM, we also announced in July 2024 a partial-

key-recovery attack which in one call to RO_G and at most n calls to the decapsulation oracle and to RO_H can determine whether 0 is one of the support elements in L and, if so, which column it belongs to in the public matrix. We implemented this and found that this causes the reference software to enter an infinite loop, which is another deviation from the specified KEM. The infinite loop allows an easy timing attack obtaining the same information. These attacks are possible for two reasons:

- The extended Patterson decoder has predictable outputs under certain inputs and in particular often outputs the 0 vector. (This was also exploited in the previous reaction attack.)
- The use of r does not sufficiently limit what is a valid ciphertext. (This is a new result exploited in the new attacks.)

The first part manifests itself in two different forms. First of all, the Extended Patterson decoder inherently fails to decode a weight-1 error if that error is at the support position $\alpha = 0$. In that case the syndrome polynomial $s(x)$ corresponds to $1/x$ modulo g so that $\tilde{s}(x) = 1 + (x/s) \bmod g$ gives 0. In that case $g_1 = 1$, $g_2 = g$, $g_{12} = 1$ and the output of SolveKeyEqn is $(1, 0)$ leading to $\sigma = g^2$, which by construction does not have any roots in L and thus SolveKeyEqn returns the 0 vector.
This issue is specific to the Extended Patterson decoder and g reducible and does not appear in the regular Patterson decoder. It can be caught as a special case by checking that the input to SolveKeyEqn has $(0, 1)$ as the first two components and returning $(0, 1)$ in that case, but care must be taken not to cause side-channel attacks. This might also not be necessary as $t = 1$ is not an option for any of the parameter sets. In any case, the implementation needs to be fixed to avoid infinite loops. (The implementation is computing g_{12} by calling `gf_poly_mul`, which multiplies g_1 by g_2 modulo g, obtaining 0 instead of the correct g. The implementation is then dividing g by 0, which loops forever.)
Secondly, random inputs to the Extended Patterson decoder often return the 0 vector. This is because they find a random polynomial of degree less than t which will often not have roots in L. A simple model says that this occurs with probability $(1 - 1/q)^n$, which is 62%, 51%, 45% for PALOMA-128, 192, 256. Our experiments are close to this model.
For both of these failure cases, the attacks use the following observations: The 0 vector is mapped to itself by any permutation of $e^* = \hat{e}$ and we can run RO_G in the forward direction on $e^* = 0$ to obtain the matching $\hat{r} = \mathrm{RO}_G(00\ldots0)$. Hence, we can produce a ciphertext which is often valid by using this $\hat{r}$ with any syndrome vector s of length $n - k$ and we can

compute the KEM key $\kappa = \mathrm{RO}_H((00....0), \hat{r}, s)$ that this would produce if the decapsulation indeed returns the 0 vector. This means, we compare the output of the decapsulation oracle with this κ to see if the decoding step indeed returned 0. This machinery can then be used in the two attacks.

Identifying if $0 \in L$ and finding its position in the public matrix: This attack iterates s over the n different columns of the public-key matrix $\hat{H}$. The first part of the ciphertext is $\hat{r}$ obtained from RO_G on input the 0 vector, the second part is the column h_i.
If the output of the decapsulation oracle matches $\mathrm{RO}_H((00\ldots0), \hat{r}, h_i)$ we have found the location of $\alpha = 0$. If we do not encounter this for any of the n columns then we know that 0 is not in L.

OW-CCA2 and IND-CCA2 attack: Given a challenge ciphertext (r, s) this attack prepares related ciphertexts and candidate KEM keys as follows. Iterating over all columns h_i we compute challenge ciphertext $(\hat{r}, s + h_i)$, where $\hat{r}$ is as above, and compare the output of the decapsulation oracle to $\mathrm{RO}_H((00\ldots0), \hat{r}, s + h_i)$. If this matches we learn that the decoding step has returned the 0 vector. Apart from the issue exploited in identifying if $0 \in L$, the decoder works correctly on syndromes coming from error vectors of weight up to t. Since s is a valid syndrome, thus corresponding to a weight-t error, having the decoding algorithm output the 0 vector means that $s + h_i$ corresponds to an error of weight larger than t. Hence, the position i in $\hat{e}$ was not set.
If the outputs do not match we do not a priori know if position i was set and had been flipped to 0 by the addition $s + h_i$, thus creating a syndrome matching an error vector of weight $t - 1$, which decodes correctly, of if the random error locator polynomial happened to have a root in L.
We can get clarity about which of the two cases we are in by taking combinations of two positions. If position j is known to be 0 in $\hat{e}$, then performing the above with $s + h_j + h_i$ is guaranteed to give non-zero output if position i was set, so that the two bit flips lead to a weight-t error while it has a significant probability of producing the 0 vector otherwise. Varying over different known positions j quickly gives clarity. Furthermore, we know that $\hat{e}$ that led to s has weight t, so we can stop once we have identified $n - t$ positions as 0 positions.
This way of modifying s to obtain $\hat{e}$ matches our earlier attack against the software (see Section 4.6), which used crashes in the code as an oracle. The July 2024 attack relies on the CCA2 decapsulation oracle, which adds the complication that the new attack needs to provide otherwise valid cipher-

texts, achieved as explained above.
To finish the OW-CCA2 or IND-CCA2 attack after recovering $\hat{e}$ we apply PermInv, for which we use r from the original ciphertext, to get e^*, and obtain $\kappa = \mathrm{RO}_H(e^*, r, s)$. This concludes the OW-CCA2 attack, which is what we implemented. For an IND-CCA2 attack, simply compare this κ to the candidate k^* provided in the challenge.
The attacks as described are stopped by checking that $\hat{e}$ as returned by Decrypt$(sk; \hat{s})$ has weight t. This check is mentioned in PKE$_1$ which is used in the security proof for the IND-CCA security in chapter 5, but it is not present in the PKE or KEM specified in chapter 3. It is also not present in the implementation provided. It is not clear if that alone suffices to achieve CCA2 security. At least a proof that if $\hat{e}$ returned by the extended Patterson decoder has weight t then $\hat{s}$ must be valid seems necessary here; note that this has to be specific to the decoder used.
For the decoder used in Classic McEliece, there is a proof of this rigidity, but the software still performs re-encryption to make sure that nothing goes wrong. See [Ber24b, Theorem 5.1.3, Theorem 7.1, and Section 8].
Inside the PALOMA security analysis (chapter 5), Alg 21 and 22 perform the weight check and re-encryption. However, these two checks do not appear in the KEM specification in chapter 3 and are not implemented in the C code provided with the submission package. A system using these two checks should be able to use the standard FO transform and achieve CCA2 security if the RSD problem is hard. Including these checks will slow down the decapsulation procedure and may require including the public key in the secret key. For ways to avoid this inclusion, see Classic McEliece.

4.8 Further updates

Following the attacks presented in July the PALMOA designers informed us that they were working on an update to their implementation that would add an extra check for the weight of e and would also remove timing variations. In August they sent us and then posted an updated version of the specification docoument, which we considered above in the section on proofs, and the updates should stop the attacks, however we are still awaiting the software update.

Chapter 5

REDOG

This chapter analyzes the security of the REinforced modified Dual-Ouroboros based on Gabidulin codes, REDOG [KHL+22], a public-key encryption system submitted to KpqC. REDOG is a code-based cryptosystem using rank-metric codes, aiming at providing a rank-metric alternative to Hamming-metric code-based cryptosystems.
Rank-metric codes were introduced by Delsarte [Del78] and independently rediscovered by Gabidulin [Gab85] in 1985, who focused on those that are linear over a field extension. Gabidulin, Paramonov, and Tretjakov [GPT91] proposed their use for cryptography in 1991. The GPT system was attacked by Overbeck [Ove05, Ove08] who showed *structural* attacks, permitting recovery of the private key from the public key.
During the mid 2010s new cryptosystems using rank-metric codes were developed such as Ouroboros [DGZ17] and the first round of the NIST competition on post-quantum cryptography saw 5 systems based on rank-metric codes: LAKE [ABD+17a], LOCKER [ABD+17b], McNie [GKK+17], Ouroboros-R [AAB+17a], and RQC [AAB+17b]. For all these systems see NIST's Round-1 Submissions page. Gaborit announced an attack weakening McNie and the McNie authors adjusted their parameters. A further attack was published in [LT18] and NIST did not advance McNie into the second round of the competition.
ROLLO, a merger of LAKE, LOCKER and Ouroboros-R, and RQC made it into the the second round but got broken near the end of it by significant advances in the cryptanalysis of rank-metric codes and the MinRank problem in general, see [BBB+20] and [BBC+20a]. In their report at the end of round 2 [AASA+20], NIST wrote an encouraging note on rank-metric codes: "Despite the development of algebraic attacks, NIST believes rank-based cryptography should continue to be researched. The rank metric cryptosystems offer a nice alternative to traditional hamming metric codes with

comparable bandwidth." (capitalization as in the original).
Kim, Kim, Galvez, and Kim [KKGK21] proposed a rank-metric system in 2021 which was then analyzed by Lau, Tan, and Prabowo in [LTP21] who also proposed some modifications to the issues they found. REDOG resembles the system in [LTP21].
We found several attacks against the REDOG submission to the first round, see [LPR23] for the full paper, but also showed that these were not fundamentally breaking REDOG. The following describes the system and the changes to the 2nd-Round submission.

5.1 Preliminaries and background notions

This section gives the necessary background on rank-metric codes for the rest of the chapter.
Let $\{\alpha_1, \dots, \alpha_m\}$ be a basis of $\mathbb{F}_{q^m}$ over $\mathbb{F}_q$. Write $x \in \mathbb{F}_{q^m}$ uniquely as $x = \sum_{i=1}^m X_i \alpha_i$, $X_i \in \mathbb{F}_q$ for all i. So x can be represented as $(X_1, \dots, X_m) \in \mathbb{F}_q^m$. We will call this the *vector representation* of x. Extend this process to $\mathbf{v} = (v_1, \dots, v_n) \in \mathbb{F}_{q^m}^n$ defining a map $\mathsf{Mat} : \mathbb{F}_{q^m}^n \to \mathbb{F}_q^{m \times n}$ by:

$$\mathbf{v} \mapsto \begin{bmatrix} V_{11} & V_{21} & \dots & V_{n1} \\ V_{12} & V_{22} & \dots & V_{n2} \\ \vdots & \vdots & \ddots & \vdots \\ V_{1m} & V_{2m} & \dots & V_{nm} \end{bmatrix}.$$

Definition 5.1.1. *The rank weight of* $\mathbf{v} \in \mathbb{F}_{q^m}^n$ *is defined as* $\mathsf{wt}_R(\mathbf{v}) := \mathsf{rk}_q(\mathsf{Mat}(\mathbf{v}))$ *and the rank distance between* $\mathbf{v}, \mathbf{w} \in \mathbb{F}_{q^m}^n$ *is* $d_R(\mathbf{v}, \mathbf{w}) := \mathsf{wt}_R(\mathbf{v} - \mathbf{w})$.

Remark 5.1.2. *It can be shown that the rank distance does not depend on the choice of the basis of* $\mathbb{F}_{q^m}$ *over* $\mathbb{F}_q$. *In particular, the choice of the basis is irrelevant for the results in this document.*

When talking about the space spanned by $\mathbf{v} \in \mathbb{F}_{q^m}^n$, denoted as $\langle \mathbf{v} \rangle$, we mean the $\mathbb{F}_q$-subspace of $\mathbb{F}_q^m$ spanned by the columns of $\mathsf{Mat}(\mathbf{v})$.
For completeness, we introduce the Hamming weight and the Hamming distance. These notions will be used in our message recovery attack against REDOG's implementation.
The *Hamming weight* of a vector $\mathbf{v} \in \mathbb{F}_{q^m}^n$ is defined as $\mathsf{wt}_H(\mathbf{v}) := \#\{i \in \{1, \dots, n\} \mid v_i \neq 0\}$ and the Hamming distance between vectors $\mathbf{v}, \mathbf{w} \in \mathbb{F}_{q^m}^n$ is defined as $d_H(\mathbf{v}, \mathbf{w}) := \mathsf{wt}_H(\mathbf{v} - \mathbf{w})$.

Let $D = d_R$ or $D = d_H$. Then an $[n, k, d]$-code C with respect to D over $\mathbb{F}_{q^m}$ is a k-dimensional $\mathbb{F}_{q^m}$-linear subspace of $\mathbb{F}_{q^m}^n$ with *minimum distance*

$$d := \min_{\mathbf{a},\mathbf{b}\in C,\, \mathbf{a}\neq\mathbf{b}} D(\mathbf{a}, \mathbf{b})$$

and *correction capability* $\lfloor (d-1)/2 \rfloor$. If $D = d_R$ (resp. $D = d_H$) then the code C is also called a *rank-metric* (resp. *Hamming-metric*) code. All codes in this document are linear over the field extension $\mathbb{F}_{q^m}$.
We say that G is a *generator matrix* of C if its rows span C. We say that H is a *parity check matrix* of C if C is the right-kernel of H.
A very well-known family of rank metric codes are *Gabidulin codes* [Gab85], which have $d = n - k + 1$.
In this analysis we can mostly use these codes as a black box, knowing that there is an efficient decoding algorithm using the parity-check matrix of the code and decoding vectors with errors of rank up to $\lfloor (d-1)/2 \rfloor$.
Another definition needed for the specification of REDOG is that of circulant matrix.

Definition 5.1.3. *A circulant matrix M over $\mathbb{F}_{q^m}$ is a $k \times n$ matrix of the form*

$$M = \begin{pmatrix} m_0 & m_1 & m_2 & \cdots & m_{n-1} \\ m_{n-1} & m_0 & m_1 & \cdots & m_{n-2} \\ m_{n-2} & m_{n-1} & m_0 & \cdots & m_{n-3} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ m_{n-k-1} & m_{n-k} & m_{n-k+1} & \cdots & m_{n-k-2} \end{pmatrix}$$

where $m_0, m_1, \ldots, m_{n-1} \in \mathbb{F}_{q^m}$.
In other words, each row of the matrix is a cyclic permutation of the previous row.

A final definition necessary to understand REDOG is that of isometries.

Definition 5.1.4. *Consider vectors in $\mathbb{F}_{q^m}^n$. An* isometry with respect to the rank metric *is a matrix $P \in \mathsf{GL}_n(\mathbb{F}_{q^m})$ satisfying that $\mathsf{wt}_R(\mathbf{v}P) = \mathsf{wt}_R(\mathbf{v})$ for any $\mathbf{v} \in \mathbb{F}_{q^m}^n$.*

Obviously matrices $P \in \mathsf{GL}_n(\mathbb{F}_q)$ are isometries as $\mathbb{F}_q$-linear combinations of the coordinates of $\mathbf{v}$ do not increase the rank and the rank does not decrease as P is invertible. The rank does also not change under scalar multiplication by some $\alpha \in \mathbb{F}_{q^m}^*$: $\mathsf{wt}_R(\alpha\mathbf{v}) = \mathsf{wt}_R(\mathbf{v})$. Note that the latter corresponds to multiplication by $P = \alpha I_n$.
Berger [Ber03] showed that any isometry is obtained by composing these two options.

Theorem 5.1.5. *[Ber03, Theorem 1] The isometry group of $\mathbb{F}_{q^m}^n$ for the rank metric is generated by scalar multiplications by elements in $\mathbb{F}_{q^m}^*$ and elements of $\mathsf{GL}_n(\mathbb{F}_q)$. This group is isomorphic to the product group $\left(\mathbb{F}_{q^m}^*/\mathbb{F}_q^*\right) \times \mathsf{GL}_n(\mathbb{F}_q)$.*

5.2 System specification

This section introduces the specification of REDOG. We follow the notation of [LTP21], with minor changes, adjusted for the round-2 submission.
The system parameters are positive integers $(n, k, \ell, q, m, r, \lambda, t_1, t_2)$, with $\ell < n$ and $t_1 + \lambda t_2 \leq r \leq \lfloor (n-k)/2 \rfloor$, as well as a hash function $\mathsf{hash} : \mathbb{F}_{q^m}^{2n-k} \to \mathbb{F}_{q^m}^{\ell}$.

KeyGen:

1. Select $H = (H_1 \mid H_2)$, $H_2 \in \mathsf{GL}_{n-k}(\mathbb{F}_{q^m})$, a parity check matrix of a $[2n-k, n]$ Gabidulin code, with syndrome decoder Φ correcting r errors.
2. Select a complete rank matrix $M \in \mathbb{F}_{q^m}^{\ell \times n}$ and isometry $P \in \mathbb{F}_{q^m}^{n \times n}$ (with respect to the rank metric).
3. Select a λ-dimensional $\mathbb{F}_q$-subspace $\Lambda \subset \mathbb{F}_{q^m}$ containing 1 and select a random circulant matrix $S^{-1} \in \mathsf{GL}_{n-k}(\mathbb{F}_{q^m})$ having entries only in Λ;
4. Compute $F = MP^{-1}H_1^T \left(H_2^T\right)^{-1} S$ and publish the public key $\mathsf{pk} = (M, F)$. Store the secret key $\mathsf{sk} = (P, H, S, \Phi)$.

Encrypt $(\mathbf{m} \in \mathbb{F}_{q^m}^{\ell}, \mathsf{pk})$

1. Generate uniformly random $\mathbf{e} = (\mathbf{e}_1, \mathbf{e}_2) \in \mathbb{F}_{q^m}^{2n-k}$ with $\mathbf{e}_1 \in \mathbb{F}_{q^m}^n$ having $\mathsf{wt}_R(\mathbf{e}_1) = t_1$ and $\mathbf{e}_2 \in \mathbb{F}_{q^m}^{n-k}$ having $\mathsf{wt}_R(\mathbf{e}_2) = t_2$;
2. Compute $\mathbf{m} = \mathsf{msg} + \mathsf{hash}(\mathbf{e})$.
3. Compute $\mathbf{c}_1 = \mathbf{m}M + \mathbf{e}_1$ and $\mathbf{c}_2 = \mathbf{m}F + \mathbf{e}_2$ and send $(\mathbf{c}_1, \mathbf{c}_2)$.

Decrypt $((\mathbf{c}_1, \mathbf{c}_2), \mathsf{sk})$

1. Compute $\mathbf{c}' = \mathbf{c}_1 P^{-1} H_1^T - \mathbf{c}_2 S^{-1} H_2^T = \mathbf{e}' H^T$ where the vector $\mathbf{e}' := (\mathbf{e}_1 P^{-1}, -\mathbf{e}_2 S^{-1})$.
2. Decode $\mathbf{c}'$ using Φ to obtain $\mathbf{e}'$, recover $\mathbf{e} = (\mathbf{e}_1, \mathbf{e}_2)$ using P and S.
3. Solve $\mathbf{m}M = \mathbf{c}_1 - \mathbf{e}_1$. Output $\mathsf{msg} = \mathbf{m} - \mathsf{hash}(\mathbf{e})$.

5.2.1 Suggested parameters

We list the suggested parameters of REDOG for 128,192 and 256 bits of security submitted to KPQC.

Security parameter	$(n, k, \ell, q, m, r, \lambda, t_1, t_2)$
128	$(30, 6, 25, 2, 59, 12, 3, 6, 2)$
192	$(44, 8, 37, 2, 83, 18, 3, 12, 2)$
256	$(58, 10, 49, 2, 109, 24, 3, 15, 3)$

Table 5.1: Suggested parameters.

5.3 The Pad Thai Attack

In this section, we describe our attack on REDOG which succeeds in recovering the messages corresponding to REDOG's ciphertexts. Let us first give an overview of the attack.

5.3.1 Overview

We break down the description of the Pad Thai attack into two steps. We aim to construct a system of linear equations which can be solved uniquely for $\mathbf{m}$, and subsequently for $\mathbf{e}_1$ and $\mathbf{e}_2$. Finally, we recover the message by computing $\mathsf{msg} = \mathbf{m} - \mathsf{hash}(\mathbf{e}_1 \mid \mathbf{e}_2)$.

First Step.

The goal of the first step is to construct a system of linear equations starting from the relation $\mathbf{c}_2 = \mathbf{mF} + \mathbf{e}_2$. To construct this system, we combine columns of $\mathbf{F}$ and the corresponding entries of $\mathbf{c}$ in order to obtain a system of equations $\mathbf{c}_2' = \mathbf{mF}' + \mathbf{e}_2'$ where $\mathbf{e}_2'$ has only t_2 nonzero entries whose positions are known.
Assume that we know the mentioned system. Observe that, for all security levels of REDOG, we have $t_2 = 2$ or $t_2 = 3$, which means that $n-k-t_2$ entries in $\mathbf{c}_2'$ are error-free. Let $i_1, \ldots, i_{t_2} \in \{1, \ldots, n-k\}$ be such that $e_{2,i_j} \neq 0$ for $j \in \{1, \ldots, t_2\}$. Take $\mathbf{F}'' \in \mathbb{F}_{2^m}^{\ell \times (n-k-t_2)}$ as the submatrix of $\mathbf{F}'$ consisting of the columns F_i' for $i \neq i_1, \ldots, i_{t_2}$. Similarly, compute $\mathbf{c}_2'' \in \mathbb{F}_{2^m}^{n-k-t_2}$ by taking the entries c_i' where $i \neq i_1, \ldots, i_{t_2}$. Then, the message $\mathbf{m}$ satisfies

$$\mathbf{c}_2'' = \mathbf{mF}'', \tag{5.1}$$

which is an underdetermined system as we have ℓ unknowns of $\mathbf{m}$ and $n-k-t_2$ equations, where $n-k-t_2 < \ell$ for every security level. In reality, $\ell - n + k = t_2 + 1$ for every security level, which means that we are t_2+1 equations short.

Second Step.

In order to uniquely compute $\mathbf{m}$ we need to pad the system in (5.1) with $t_2 + 1$ extra error-free equations by combining some of the equations from $\mathbf{c}_1 = \mathbf{m}\mathbf{M} + \mathbf{e}_1$. Let $c'_{1,i} = \mathbf{m}M'_i$ for $i = 1, \dots, t_2 + 1$ be such error-free equations. We can add these equations to (5.1) and obtain a new system

$$(\mathbf{c}''_2 \mid c_{1,1} \mid \cdots \mid c_{1,t_2+1}) = \mathbf{m}\left(\mathbf{F}'' \mid M'_1 \mid \cdots \mid M'_{t_2+1}\right). \tag{5.2}$$

In REDOG's specification, $\mathbf{M}$ is chosen uniformly at random among the full-rank matrices in $\mathbb{F}_{2^m}^{\ell\times n}$. Moreover, $\mathbf{F}$ is assumed to be another random matrix by [KHL+24, Problem 2] so we can safely assume that $\left(\mathbf{F}'' \mid M'_1 \mid \cdots \mid M'_{t_2+1}\right) \in \mathbb{F}_{2^m}^{\ell\times\ell}$ is a random matrix, thus having full rank with high probability. We can now compute $\mathbf{m}$ by inverting the system (5.2) and recover msg.

5.3.2 First Step

We describe a method to produce a system of equations $\mathbf{c}'_2 = \mathbf{m}\mathbf{F}' + \mathbf{e}'_2$ where the Hamming weight $\mathsf{wt}_H(\mathbf{e}'_2) = t_2$ and the positions of non-zero entries of $\mathbf{e}'_2$ are known. This can be done because of the following observation.

Remark 5.3.1. *Let F_i denote the i-th column of $\mathbf{F}$, then $c_{2,i} = \mathbf{m}F_i + e_{2,i}$. Assume that $e_{2,i} = e_{2,j}$ for some i, j. Then*

$$c_{2,i} + c_{2,j} = \mathbf{m}(F_i + F_j) + e_{2,i} + e_{2,j} = \mathbf{m}(F_i + F_j).$$

Let $\alpha_1, \dots, \alpha_{t_2} \in \mathbb{F}_{2^m}^*$ be such that $\langle \mathbf{e}_2 \rangle_{\mathbb{F}_2} = \langle \alpha_1, \dots, \alpha_{t_2} \rangle_{\mathbb{F}_2}$. So each entry $e_{2,i}$ of $\mathbf{e}_2$ can assume a value in an $\mathbb{F}_2$-vector subspace of $\mathbb{F}_{2^m}$ containing 2^{t_2} elements. This suggests that REDOG's encryption algorithm chooses $\mathbf{e}_2$ among $2^{(n-k)t_2}$ possibilities (actually, less than $2^{(n-k)t_2}$ as the rank weight constraint $\mathsf{wt}_R(\mathbf{e}_2) = t_2$ must also hold). Label the unknown values of $\langle \mathbf{e}_2 \rangle_{\mathbb{F}_2}$ as $\{0, \alpha_1, \alpha_2, \dots, \alpha_{2^{t_2}-1}\}$, where

$$\alpha_j = \sum_{h=1}^{t_2} z_{j,h}\alpha_h \tag{5.3}$$

for some $z_{j,h} \in \mathbb{F}_2$ for every $j = t_2 + 1, \dots, 2^{t_2} - 1$. Most of the time we will handle 0 separately, but for convenience, we define $\alpha_0 = 0$.

In the following, let $2^{\{1,\dots,n\}}$ denote the set of subsets of $\{1,\dots,n\}$, and thus $\left(2^{\{1,\dots,n\}}\right)^t$ a vector of t such subsets.

Definition 5.3.2 (Set of arrangements of subsets)**.** *Let $t \in \mathbb{N}$ be a positive integer. We define the set of arrangements of t disjoint subsets over n elements as the set of ordered tuples in $\left(2^{\{1,\dots,n\}}\right)^t$ defined as*

$$A_{t,n} := \left\{ \mathbf{a} \in \left(2^{\{1,\dots,n\}}\right)^t \mid \bigcup_{i=1}^{t} a_i = \{1,\dots,n\}, a_i \cap a_j = \emptyset \ \forall i \neq j \right\}.$$

Proposition 1. *Let $\alpha = \{\alpha_1,\dots,\alpha_{t_2}\} \subset \mathbb{F}_{2^m}^*$ be a set of $\mathbb{F}_2$-linearly independent elements. There exists a one-to-one correspondence between the set $E_{\alpha,n-k} := \{\mathbf{e} \in \mathbb{F}_{2^m}^{n-k} \mid \langle \mathbf{e} \rangle_{\mathbb{F}_2} \subseteq \langle \alpha_1,\dots,\alpha_{t_2} \rangle_{\mathbb{F}_2}\}$ and $A_{2^{|\alpha|},n-k}$.*

Proof. For $\mathbf{e} \in E_{\alpha,n-k}$, denote by

$$\mathbf{e}^0 := \{i \in \{1,\dots,n-k\} \mid e_i = 0\}$$

and by

$$\mathbf{e}^{\alpha_j} := \{i \in \{1,\dots,n-k\} \mid e_i = \alpha_j\}$$

the positions where $\mathbf{e}_2$ is 0 and α_j for all $j = 1,\dots,2^{t_2}-1$, respectively. We prove that the map

$$\begin{aligned} \varphi_{\alpha,n-k} : E_{\alpha,n-k} &\to A_{2^{|\alpha|},n-k} \\ \mathbf{e} &\mapsto (\mathbf{e}^0, \mathbf{e}^{\alpha_1}, \dots, \mathbf{e}^{\alpha_{2^{t_2}-1}}) \end{aligned}$$

is a bijection by showing that it is both injective and surjective. Let $\mathbf{e}, \mathbf{f} \in E_{\alpha,n-k}$ be such that $\mathbf{e} \neq \mathbf{f}$. Then there exists $i \in \{1,\dots,n-k\}$ such that $e_i \neq f_i$. Write $e_i = \alpha_{j_1}$ and $f_i = \alpha_{j_2}$ for some $j_1, j_2 \in \{0,\dots,2^{t_2}-1\}$ with $j_1 \neq j_2$, then $\mathbf{e}^{\alpha_{j_1}} \neq \mathbf{f}^{\alpha_{j_1}}$. It follows that $\varphi_{\alpha,n-k}(\mathbf{e}) \neq \varphi_{\alpha,n-k}(\mathbf{f})$.
On the other hand, let $\mathbf{a} \in A_{2^{|\alpha|},n-k}$ and let $\mathbf{e} \in \mathbb{F}_{2^m}^{n-k}$ be such that $e_j = 0$ for every $j \in a_1$ and $e_j = \alpha_{i-1}$ for every $j \in a_{i-1}$ and every $i = 2,\dots,2^{t_2}$. Clearly, $\langle \mathbf{e} \rangle_{\mathbb{F}_2} \subseteq \langle \alpha_1,\dots,\alpha_{t_2} \rangle_{\mathbb{F}_2}$ and $\varphi_{\alpha,n-k}(\mathbf{e}) = \mathbf{a}$.

□

Definition 5.3.3 (Good basis of a vector)**.** *Let $\mathbf{v} \in \mathbb{F}_{2^m}^n$ be such that $\mathsf{wt}_R(\mathbf{v}) = t$. A set of elements $\{\alpha_1,\dots,\alpha_t\} \subset \mathbb{F}_{2^m}^t$ such that $\langle \mathbf{v} \rangle_{\mathbb{F}_2} = \langle \alpha_1,\dots,\alpha_t \rangle_{\mathbb{F}_2}$, is called a good basis for $\mathbf{v}$ if for every $j \in \{1,\dots,t\}$ there exists an $i \in \{1,\dots,n\}$ such that $v_i = \alpha_j$.*

Remark 5.3.4. *It is clear that a good basis exists for every $\mathbf{e} \in \mathbb{F}_{2^m}$. To see that, let $t = \mathsf{wt}_R(\mathbf{e})$ and define the basis $\{\alpha_1,\dots,\alpha_t\}$ of $\langle \mathbf{e} \rangle_{\mathbb{F}_2}$ by taking the leftmost t entries in $\mathbf{e}$ which are linearly independent over $\mathbb{F}_2$. Then $\{\alpha_1,\dots,\alpha_t\}$ is a good basis for $\mathbf{e}$.*

Denote by $A'_{2^{t_2},n-k}$ the subset of $A_{2^{t_2},n-k}$ so that $a_i \neq \emptyset$ for $i = 2, \dots, t_2 + 1$.

Definition 5.3.5 (Arrangement of a vector)**.** *Let* $\mathbf{e} \in \mathbb{F}_{2^m}^{n-k}$ *with* $\mathsf{wt}_R(\mathbf{e}) = t_2$ *and* $\alpha = \{\alpha_1, \dots, \alpha_{t_2}\} \subset \mathbb{F}_{2^m}^{t_2}$ *be a good basis for* $\mathbf{e}$*. Then we call* $\varphi_{\alpha,n-k}(\mathbf{e})$ *the arrangement of* $\mathbf{e}$ *with respect to* α*.*

Observe that, given a good basis α for $\mathbf{e}$, the arrangement of $\mathbf{e}$ w.r.t. α is in $A'_{2^{t_2},n-k}$.

Algorithm 1 RearrangeSystem

Input: An arrangement $\mathbf{a} \in A'_{2^{t_2},n-k}$, a REDOG's partial ciphertext $\mathbf{c}_2 \in \mathbb{F}_{2^m}^{n-k}$ corresponding to a message $\mathbf{m} \in \mathbb{F}_{2^m}^{\ell}$ under the partial public key $\mathbf{F} \in \mathbb{F}_{2^m}^{\ell \times (n-k)}$.

Output: A vector $\mathbf{c}''_2 \in \mathbb{F}_{2^m}^{n-k-t_2}$ and a matrix $\mathbf{F}'' \in \mathbb{F}_{2^m}^{\ell \times (n-k-t_2)}$.

```
1. Fix elements $x_i \in a_i$ for every $i = 2, \dots, t_2 + 1$;
2. Construct $\mathbf{F}'' \in \mathbb{F}_{2^m}^{\ell \times (n-k-t_2)}$ and $\mathbf{c}''_2$ by computing the following columns and values:
    (a) $F''_j = F_j$ and $c''_{2,j} = c_{2,j}$ for every $j \in a_1$;
    (b) $F''_j = F_j + F_{x_i}$ and $c''_{2,j} = c_{2,j} + c_{2,x_i}$ for all $j \in a_i \setminus \{x_i\}$ and $i = 2, \dots, t_2 + 1$;
    (c) $F''_j = F_j + \sum_{h=1}^{t_2} z_{i-1,h} F_{x_h+1}$ and $c''_{2,j} = c_{2,j} + \sum_{h=1}^{t_2} z_{i-1,h} c_{2,x_h+1}$ for all $j \in a_i$ and $i = t_2 + 2, \dots, 2^{t_2}$.
3. Return $\mathbf{F}''$ and $\mathbf{c}''_2$, the matrix $\mathbf{F}''$ and vector $\mathbf{c}''$ punctured at $x_i, i = 2, \dots, t_2 + 1$.
```

Proposition 2. *Let* $\mathbf{c}_2 = \mathbf{m}\mathbf{F} + \mathbf{e}_2$ *be a REDOG's partial ciphertext and* $\varphi_{\alpha,n-k}(\mathbf{e}_2)$ *be the arrangement of* $\mathbf{e}_2$ *w.r.t a good basis* α*. Then Algorithm 1 returns* $\mathbf{c}''_2 \in \mathbb{F}_2^{n-k-t_2}$ *and* $\mathbf{F}'' \in \mathbb{F}_{2^m}^{\ell \times (n-k-t_2)}$ *such that*

$$\mathbf{c}''_2 = \mathbf{m}\mathbf{F}''.$$

Proof. The algorithm repeatedly applies the observation in Remark 5.3.1. Each $j \in a_1$ has that $c_{2,j}$ is error free. Each $j \in a_i, i = 2, \dots, t_2 + 1$ has $c_{2,j} = \mathbf{m}F_j + \alpha_{i-1}$. The algorithm selects one such index as x_i and then applies Remark 5.3.1 to cancel the α_{i-1} in all other $c_{2,j}$ for $j \in a_i \setminus \{x_i\}$. Eventually, $\mathbf{c}''_2$ is punctured at x_i so that only those entries without error remain.

Similarly, all $c_{2,j}$ with $j \in a_i$ for $i = t_2+2, \ldots, 2^{t_2}$ have error α_{i-1} added and (5.3) states the coefficients $z_{i-1,h}$ representing α_{i-1} in the basis. Again using that c_{x_h} contributes α_{h-1} shows that the third case produces an error-free $c''_{2,j}$ for $j \in a_i$.
In total, the matrix and vector are punctured at the t_2 positions of the x_i, thus producing $n-k-t_2$ error free equations $c''_{2,j} = \mathbf{m}F''_j$. $\square$

Proposition 2 together with Algorithm 1 provides a method that transforms the system of equations $\mathbf{c}_2 = \mathbf{mF} + \mathbf{e}_2$ into a smaller system $\mathbf{c}''_2 = \mathbf{mF}''$ that does not involve any noise. As the system is underdetermined, we present an algorithm in the next section that exploits REDOG's partial ciphertext $\mathbf{c}_1$ to obtain the necessary remaining equations for the system.

Remark 5.3.6. *Note that Proposition 2 assumes knowledge of the arrangement of $\mathbf{e}_2$. We want to stress that, since the basis α is unknown, knowing the arrangement of $\mathbf{e}_2$ w.r.t. α does not necessarily mean knowing $\mathbf{e}_2$. This assumption will be satisfied as we iterate over all possible arrangements of $\mathbf{e}_2$.*

5.3.3 Second Step

In this subsection, we investigate how to pad the system equations $\mathbf{c}''_2 = \mathbf{mF}''$ with t_2+1 additional equations to uniquely determine $\mathbf{m}$. The idea is to construct these extra equations, combining equations from $\mathbf{c}_1 = \mathbf{mM} + \mathbf{e}_1$. Observe that since $\mathsf{wt}_R(\mathbf{e}_1) = t_1$, then any set $\{e_{1,i_1}, \ldots, e_{1,i_{t_1+1}}\}$ of t_1+1 entries of $\mathbf{e}_1$ is linearly dependent, i.e. there exist $z_1, \ldots, z_{t_1+1} \in \mathbb{F}_2$ not all zero such that

$$\sum_{j=1}^{t_1+1} z_j e_{1,i_j} = 0.$$

This suggests that given a set of t_1+1 equations of $\mathbf{c}_1 = \mathbf{mM} + \mathbf{e}_1$ one can search the space of $\mathbb{F}_2$-linear combinations for t_1+1 non-zero combinations of the equations, which cancels the error factor.

Remark 5.3.7. *Observe that we need to make sure that the columns M_{i_j} for $j = 1, \ldots, t_1+1$ are linearly independent, as otherwise we might run into*

$$\sum_{j=1}^{t_1+1} z_j M_{i_j} = 0$$

which is a useless equation. However, the probability for this to happen is negligible for each parameter set.

Since we need $t_2 + 1$ extra equations to pad the system, we need to find $t_2 + 1$ equations simultaneously with this method. In total, we obtain a linear system of $\ell = n - k + 1$ equations that can be solved to recover the message.

5.3.4 The Full Attack

For each system rearrangement that we perform in the first step, we need to test all paddings in the second step. Testing the solution of each system we construct implies computing a candidate message $\mathbf{m}' \in \mathbb{F}_{2^m}^{\ell}$ and candidate errors $\mathbf{e}_1' \in \mathbb{F}_{2^m}^{n}$ and $\mathbf{e}_2' \in \mathbb{F}_{2^m}^{n-k}$ and checking whether the rank weights of $\mathbf{e}_1'$ and $\mathbf{e}_2'$ match t_1 and t_2, respectively. Therefore, combining the two steps described in this section, we obtain the following algorithm.

Algorithm 2 PadThaiAttack

```
Input: A REDOG's ciphertext $\mathbf{c} = (\mathbf{c}_1, \mathbf{c}_2) \in \mathbb{F}_{2^m}^{2n-k}$ corresponding to a message $\mathsf{msg} \in \mathbb{F}_{2^m}^{\ell}$ under the public key $\mathsf{pk} = (\mathbf{M}, \mathbf{F}) \in \mathbb{F}_{2^m}^{\ell \times (2n-k)}$.
Output: The message $\mathsf{msg}$.

For each arrangement $\mathbf{a} \in A'_{2^{t_2}, n-k}$ do:
    1. Let $\mathbf{F}'', \mathbf{c}'' = \mathrm{RearrangeSystem}(\mathbf{a}, \mathbf{c}_2, \mathbf{F})$;
    2. Pick random sets $J_1, \dots, J_{t_2+1} \subset \{1, \dots, n\}$ with $|J_i| = t_1 + 1$;
    3. Let $\mathbf{M}_{J_i}$ be the matrix consisting of columns of $\mathbf{M}$ indexed by $J_i$;
    4. If $\mathsf{rk}(\mathbf{M}_{J_i}) < t_1 + 1$ for some $i \in \{1, \dots, t_2 + 1\}$ then go to Step 2.
    5. For every $(\mathbf{v}_1, \dots, \mathbf{v}_{t_2+1}) \in \left(\mathbb{F}_2^{t_1+1}\right)^{t_2+1}$ do:
        (a) Compute $M'_i = \mathbf{M}_{J_i} \mathbf{v}^\top$ for each $i = 1, \dots, t_2 + 1$;
        (b) Let $\mathbf{c}_{1,J_i}$ be the vector consisting of the entries of $\mathbf{c}_1$ indexed by $J_i$;
        (c) Compute $c'_{1,i} = \mathbf{c}_{1,J_i} \mathbf{v}_i^\top$ for each $i = 1, \dots, t_2 + 1$;
        (d) Let $G := (\mathbf{F}'' \mid M'_1, \dots, M'_{t_2+1})$ and $\mathbf{y} := (\mathbf{c}'' \mid c'_{1,2}, \dots, c'_{1,t_2+1})$;
        (e) Compute $\mathbf{m}' = \mathbf{y}\mathbf{G}^{-1}$;
        (f) Compute $\mathbf{e}'_1 = \mathbf{c}_1 - \mathbf{m}'\mathbf{M}$ and $\mathbf{e}'_2 = \mathbf{c}_2 - \mathbf{m}'\mathbf{F}$;
        (g) If $\mathsf{wt}_R(\mathbf{e}'_1) = t_1$ and $\mathsf{wt}_R(\mathbf{e}'_2) = t_2$ then return $\mathsf{msg} = \mathbf{m}' - \mathsf{hash}(\mathbf{e}'_1 \mid \mathbf{e}'_2)$.
```

Let us provide an argument for the correctness of the Pad Thai attack.

Proposition 3. *Algorithm 2, under the assumption that matrix* $\mathbf{G}$ *in Step 5.(d) is invertible, recovers a valid message* msg *corresponding to a REDOG's ciphertext* $\mathbf{c}$ *under public key* $\mathsf{pk} = (\mathbf{M}, \mathbf{F})$*.*

The proof of Proposition 3 follows directly from Proposition 2 and the arguments given in Section 5.3.3. We thus omit the full proof. In the next section, we give the complexity analysis of our attack and point out some areas of improvement.

Remark 5.3.8. *The matrix* $\mathbf{G}$ *in Step 5.(d) of Algorithm 2 needs to be invertible for the attack to be successful. This is a direct consequence of*

Lemma 5.4.2, which can be applied to show that $\mathbf{G}$ *is invertible with probability* ~ 1. *A more elaborate argument is presented in Section 5.4.1.*

5.4 Analysis of the Pad Thai Attack

In this section, we describe the complexity of our attack on REDOG described in Algorithm 2. Let us start with the following easy lemma.

Lemma 5.4.1. *The cardinality of* $A_{2^{t_2},n-k}$ *is* $2^{t_2(n-k)}$.

Proof. By Proposition 1 there is a bijection between $A_{2^{t_2},n-k}$ and $E_{\alpha,n-k}$ for a fixed set $\alpha = \{\alpha_1, \dots, \alpha_{t_2}\} \subset \mathbb{F}_{2^m}^{t_2}$. The number of elements in $E_{\alpha,n-k}$ is clearly $2^{t_2(n-k)}$.

$\square$

A first assessment of the complexity of the Pad Thai attack is given in the following proposition.

Proposition 4. *The Algorithm 2 recovers the message* msg *corresponding to a REDOG ciphertext* $\mathbf{c}$ *under public key* $\mathsf{pk} = (\mathbf{M}, \mathbf{F})$ *in*

$$\mathcal{O}(2^{(t_1+1)(t_2+1)+t_2(n-k)}\ell^{\omega}m^2) \tag{5.4}$$

field operations, where $2 \leq \omega \leq 3$ *is the matrix multiplication exponent.*

Proof. The algorithm consists of two nested cycles. The first cycle iterates over all arrangements $A'_{2^{t_2},n-k}$, which is a subset of $A_{2^{t_2},n-k}$ whose cardinality is reported in Lemma 5.4.1.
The most expensive steps of each cycle of Algorithm 2 are Steps 2.(c) in Algorithm 1 and 5.(e). The former computes $2^{t_2} - t_2 - 1$ sums $F_j + \sum_{h=1}^{t_2} z_{i-1,h}F_{x_{h+1}}$, i.e. the sum of t_2 elements of $\mathbb{F}_{2^m}^{\ell}$ for a total number of operations in $\mathbb{F}_2$ in $\mathcal{O}(2^{t_2}mt_2\ell)$. The latter happens in the nested cycle and inverts a matrix $\mathbf{G} \in \mathbb{F}_{2^m}^{\ell\times\ell}$. Its cost, using schoolbook multiplications in finite fields, is in $\mathcal{O}(\ell^{\omega}m^2)$ and is performed $2^{(t_1+1)(t_2+1)}$ times for each outer cycle. For each parameter set in Table 5.1 we have that

$$2^{t_2}mt_2\ell < 2^{(t_1+1)(t_2+1)}\ell^{\omega}m^2.$$

Combining with the number of outer cycles we obtain the claimed complexity.

$\square$

The following table reports the updated security provided by REDOG based on our attack.

Security parameter	$(n, k, \ell, q, m, r, \lambda, t_1, t_2)$	Pad Thai attack
128	$(30, 6, 25, 2, 59, 12, 3, 6, 2)$	**93.8**
192	$(44, 8, 37, 2, 83, 18, 3, 12, 2)$	**138.37**
256	$(58, 10, 49, 2, 109, 24, 3, 15, 3)$	**237.3**

Table 5.2: $\log_2$ of the complexity of the Pad Thai attack for each security level of REDOG according to equation (5.4) with $\omega = 2.807$.

Table 5.2 suggests that the combination of parameters of REDOG security level 256 has a smaller loss of security under the Pad Thai attack compared to security levels 128 and 192. This can be explained by the choices for parameter t_2.

5.4.1 Success Probability

The matrix $\mathbf{F} \in \mathbb{F}_{2^m}^{\ell \times n-k}$ is assumed to be indistinguishable from random as per [KHL+24, Problem 2], hence we can consider every output matrix $\mathbf{F}'' \in \mathbb{F}_{2^m}^{\ell \times n-k-t_2}$ of Algorithm 1 as random too. Furthermore, given that each pad consists of a combination of random columns of an actually random matrix $\mathbf{M} \in \mathbb{F}_{2^m}^{\ell \times n}$, we conclude that the entire matrix $\mathbf{G} \in \mathbb{F}_{2^m}^{\ell \times \ell}$ can be considered as a matrix chosen uniformly at random. We only need to estimate the probability that $\mathbf{G}$ is invertible in order for the attack to succeed when the right system has been set up. To this end we can use the following result of [LPR23].

Lemma 5.4.2 ([LPR23, Lemma 4.2])**.** *Let V be a t-dimensional subspace $V \subseteq \mathbb{F}_2^m$ and let $S \in V^s$ be a uniformly random s-tuple of elements of V. The probability $p(q, s, t)$ that $\langle S_i \mid i \in \{1, \ldots, s\}\rangle = V$ is*

$$p(q,s,t) = \begin{cases} 0 & \text{if} \quad 0 \le s < t; \\ \sum_{i=0}^{t} \begin{bmatrix} t \\ i \end{bmatrix}_q (-1)^{t-i} q^{s(i-t)+\binom{t-i}{2}} & \text{otherwise}, \end{cases} \tag{5.5}$$

where $\begin{bmatrix} t \\ i \end{bmatrix}_q$ is the q-binomial coefficient, counting the number of subspaces of dimension i of $\mathbb{F}_2^t$, and $\binom{a}{b} = 0$ for $a < b$. In particular, this probability does not depend on m or on the choice of V, but only on its dimension.

By setting $V = \mathbb{F}_{2^m}^{\ell}$ and $s = t = \ell$ in the above lemma, we obtain that the set of columns of $\mathbf{G}$ spans the entire space V. In other words, $\mathbf{G}$ is invertible with probability ~ 1 for every cycle and every security level. As a result, we are assured that the attack succeeds at recovering the secret msg with probability ~ 1.

5.4.2 Attack Improvements

In this subsection, we point out an interesting behavior of Algorithm 1 of the first step of our attack. We observe that Algorithm 1 rearranges the system $\mathbf{c}_2 = \mathbf{mF} + \mathbf{e}_2$ depending only on the arrangement of $\mathbf{e}_2$. Here is an example for $t_2 = 2$.

Example 5.4.3. *Let* $\mathbf{e}, \mathbf{f} \in \mathbb{F}_{2^m}^{n-k}$ *be the vectors*

$$\mathbf{e} = (0, 0, \alpha, \alpha, \beta, \alpha + \beta)$$

and

$$\mathbf{f} = (0, 0, \beta, \beta, \alpha + \beta, \alpha).$$

These two vectors have the same arrangement w.r.t. the bases $\{\alpha, \beta\}$ *and* $\{\beta, \alpha + \beta\}$*, respectively. Now, let* $\mathbf{c_e} = \mathbf{mF} + \mathbf{e}$ *and* $\mathbf{c_f} = \mathbf{mF} + \mathbf{f}$ *and let* $x_1 = 3$ *and* $x_2 = 5$*. Let also* $\mathbf{a}$ *be the arrangement of* $\mathbf{e}$ *(equivalently, of* $\mathbf{f}$*). Then, on inputs* $(\mathbf{a}, \mathbf{c_e})$ *and* $(\mathbf{a}, \mathbf{c_f})$*, Algorithm 1 produces the same output.*

This means that we can run the first cycle on a subset of the arrangements $A'_{2^{t_2},n-k}$ as Algorithm 1 does not distinguish between errors having the same arrangement. Indeed, for each arrangement there are $r = \prod_{i=0}^{t_2-1}(2^{t_2} - 2^i)$ other arrangements for which Algorithm 1 produces the same output. The updated complexity becomes therefore

$$\mathcal{O}(2^{(t_1+1)(t_2+1)+t_2(n-k)}\ell^\omega m^2 r^{-1}). \tag{5.6}$$

The updated values are as follows.

Security parameter	$(n, k, \ell, q, m, r, \lambda, t_1, t_2)$	Pad Thai attack
128	$(30, 6, 25, 2, 59, 12, 3, 6, 2)$	**91.22**
192	$(44, 8, 37, 2, 83, 18, 3, 12, 2)$	**135.78**
256	$(58, 10, 49, 2, 109, 24, 3, 15, 3)$	**229.9**

Table 5.3: $\log_2$ of the complexity of the Pad Thai attack for each security level of REDOG according to equation (5.6) with $\omega = 2.807$.

Another improvement comes from noting that the complexity estimates consider all arrangements in $A_{2^{t_2},n-k}$ including those that correspond to error vectors $\mathbf{e}_2$ that have rank weight $\mathsf{wt}_R(\mathbf{e}_2) \leq t_2$. Considering only arrangements for error vectors with rank weight exactly t_2 slightly reduces the number of arrangements that we need to iterate over in the first step.

Remark 5.4.4. *We want to stress that the values in Table 5.2 and Table 5.3 are overestimates due to the fact that Algorithm 2 iterates over $A'_{2^{t_2},n-k}$, which is a subset of $A_{2^{t_2},n-k}$ and that we assume schoolbook arithmetic in $\mathbb{F}_{2^m}$.*

5.5 Other observations and potential directions

In this section we measure key sizes in bytes or kilobytes (KB). We believe that the key sizes reported in [KHL+24, Table 1] have been computed erroneously. Regarding the public key size of REDOG, the authors seem to use the formula

$$\text{size}_{\mathsf{pk}} = m\ell(n-k)/8$$

which forgets about the size of the matrix M. Plugging the values of the parameters in the above formula we also get slightly different values from those reported in Table 2. In [LPR23] we had recommended defining M via a seed; which would reduce the M part of the public key to just the size of the seed, typically 256 bits, but means a slowdown in the computations for computing M from the seed, this affects KeyGen, encryption/encapsulation, and decapsulation. An alternative option is that M is considered a system parameter and shared by all users. This is also not consistent with the current description, as M is listed as part of the public key, and would require some security analysis.
For the currently described version using a random M, we believe that the correct formula is

$$\text{size}_{\mathsf{pk}} = m\ell(2n-k)/8.$$

Moreover, we believe that the formula computing the sizes of private keys is incorrect as it seems to forget that P is an isometry w.r.t. the rank metric, meaning that there is an $\mathbb{F}_{q^m}$-scalar to take into account as per Theorem 5.1.5. The correct formula is then

$$\text{size}_{\mathsf{sk}} = (m(3n-2k+1)+n^2)/8$$

Using these formulas, the correct key sizes are

Security parameter	$\text{size}_{\mathsf{pk}}$	$\text{size}_{\mathsf{sk}}$
128	9.72	0.68
192	29.99	1.42
256	69.11	2.47

Table 5.4: Actual REDOG key sizes in KB.

Finally, the submission document describes only a PKE and not a KEM. As discussed in Section 2.2, the FO transform can be used to turn a OW-CPA secure PKE into an IND-CCA secure KEM. While such considerations are entirely missing, see below, the switch to the KEM increases the secret key size by requiring the inclusion of the public key for the reencryption step, and of course leading to decapsulation speeds that are slower than the decryption speeds.

5.5.1 Parameter selection

It is not clear to us how the complexities of rank decoding attacks for each security level have been computed. In [LPR23] we pointed to some code that performs this computation, and running this on the suggested parameters we note that the complexities reported in [KHL+24] are computed using Winograd exponent for matrix multiplication instead of Strassen's. This is a safe choice, but for practical attacks estimates using Strassen's exponent are more reliable. Also we believe that at least for security 128 and 192 the best complexity is given by brute force attack instead of BBB+. In their document, the authors talk about a fix in the code we provided. It would be helpful if the authors could share their version of such code so that we can make clear how these values are computed.

5.5.2 Potential directions

We became aware of attacks on rank metric code based schemes using Loidreau's idea of selecting invertible matrices having entries restricted to a small space Λ of dimension λ, see [CC20] for $\lambda = 2$ and [Gha22] for $\lambda = 3$. REDOG uses $\lambda = 3$. We believe that something can be said when combining the mentioned analysis strategies with that of the Frobenius-weak attack, described for example in [HTMR16].

5.5.3 Implementation

We could not run the C-code coming with the implementation package of REDOG. We assume that the REDOG team is still working on the code and suggest to add a README to the package with the instruction for compiling and running the code. We think that providing a working implementation package is crucial for us to perform a complete analysis.
Nevertheless, we produced a SageMath implementation of the cryptosystem following the specification faithfully, except for the fixes we point out in Section 5.5.4. We paste a copy of the implementation here.

```
from itertools import chain

# params for security 128
q,m,n,k,l,t1,t2,lamb = 2,59,30,6,25,6,2,3
F = GF(q^m)
baseF = GF(q)

# I need this C just to use methods like rank_weight_of_vector()....
C = codes.LinearRankMetricCode(random_matrix(F,25,30),baseF)

def rand_circulant(size):
  LAMB = vector(F,[1]+[F.random_element() for i in range(lamb-1)])
  while C.rank_weight_of_vector(LAMB) != lamb:
    LAMB = vector(F,[1]+[F.random_element() for i in range(lamb-1)])
  mat = random_matrix(baseF,lamb,n-k)
  row = list(LAMB*mat)
  S = [row]
  for i in range(size-1):
    row = [row[-1]] + row[0:-1]
    S += [row]
  return matrix(S), LAMB


def generror(length, weight):
  tmpe = vector(F,[F.random_element() for i in range(weight)])
  mat = random_matrix(baseF,weight,length)
  e = tmpe*mat
  while C.rank_weight_of_vector(e) != weight:
    tmpe = vector(F,[F.random_element() for i in range(weight)])
    mat = random_matrix(baseF,weight,length)
    e = tmpe*mat
  return e

def keygen():
  S,LAMB = rand_circulant(n-k)
  while not S.is_invertible():
    S,LAMB = rand_circulant(n-k)

  C = codes.GabidulinCode(F,2*n-k,n)
  H = C.parity_check_matrix()
  H2 = H[:,n:]
  while not H2.is_invertible():
    C = codes.GabidulinCode(F,2*n-k,n)
    H = C.parity_check_matrix()
    H2 = H[:,n:]
  H1 = H[:,:n]
  gamma = F.random_element()
  while gamma in baseF:
    gamma = F.random_element()

  Q = random_matrix(baseF,n,n)
  while not Q.is_invertible():
    Q = random_matrix(baseF,n,n)
  P = gamma*Q

  M = random_matrix(F,l,n)
  while not M.rank() == l:
    M = random_matrix(F,l,n)

  pubkeyF = M*P.inverse()*H1.transpose()*(H2.transpose()).inverse()*(S.inverse())
  return (M,pubkeyF),(P,S,H,H1,H2,LAMB,C)

def encrypt(m,pk):
  e1 = generror(n,t1)
  e2 = generror(n-k,t2)

  #e_sent = vector(chain(e1,e2))

  y1 = m*pk[0] + e1
  y2 = m*pk[1] + e2

  y = vector(chain(y1,y2))
  return y


def decrypt(y,sk,pk):
  y1 = y[:n]
  y2 = y[n:]
  s = y1*sk[0].inverse()*(sk[3].transpose()) - y2*sk[1]*(sk[4].transpose())
  X = sk[2].solve_right(s)
  c = C.decode_to_code(X)

  e_rec = X - c
  e_rec1 = e_rec[:n]*sk[0]
  e_rec2 = e_rec[n:]*sk[1].inverse()
  e_rec = vector(chain(e_rec1,e_rec2))
  m_rec = pk[0].solve_left(y1-e_rec1)
  return e_rec,m_rec


pk,sk = keygen()
C = sk[6]
# Encrypt
m = random_vector(F,l)
y = encrypt(m,pk)

# Decrypt (returns error vector and message)

e_rec,m_rec = decrypt(y,sk,pk)

print("m == m_rec : ", m_rec == m)
```

5.5.4 Editorial observations

There are some inconsistencies in the document [KHL+24] that we point out in the following list (which is not ordered by importance):

- In the specification, we recommend clearly stating that the matrix $P \in \mathbb{F}_{q^m}^{n \times n}$ represents an isometry of the space with respect to the rank metric. Otherwise decryption fails, as pointed out in [LPR23];
- If the intent is for S^{-1} to be chosen as a circulant matrix, we recommend clearly stating this in the specification of the system;
- The Gabidulin decoder is denoted by both Φ and Φ_H. We recommend choosing one notation and sticking to it (or if there is any difference between regular decoder and syndrome decoder, which should not be the case, clearly stating that).
- The statement "We adopt the approach of selecting matrices S and S^{-1} and addressing..." does not make sense.
- Tables 2, 3, and 4 report key sizes of other PQ candidates using bytes, bits and KB as measures. We recommend choosing one and sticking to it throughout all the tables, this improves readability.
- In section 4.3 first paragraph the citation [LT19] should be replaced with [LTP21].
- Similarly to previous item, right after Definition 7, the citation [LT19] should be replaced with [LTP21].

5.6 Provable Security of REDOG

In this section, we discuss the security proof of REDOG. No changes seem to be made regarding the security proof of REDOG compared to the submission in the first round.

5.6.1 IND-CPA Security of REDOG

We would like to address the following improvements of the security proof to the authors.

1. *Arguments regarding problem 2.* There does not seem to be a formal argument on why Problem 2 is a hard problem to solve for the given

$F = MH_1^T[H_2^T]^{-1}S$. We are aware that so far, no attacks have been found that efficiently distinguish matrix F from an arbitrary matrix R. The authors remark that, due to the choices for matrices M and S^{-1}, Overbeck's attack [Ove08] can be circumvented. However, this does not imply that this is a hard problem in general. Next to this, we wonder why problem two is equivalent to a matrix factorization problem, as a formal argument for this is lacking.

2. *Game transition REDOG to mod DO.Gab-PKE missing.* The security proof of Modified DO.Gab-PKE [KKGK21] takes S^{-1} to be an invertible matrix, whereas $S^{-1} \in \mathsf{GL}_{n-k}(\Lambda)$ for some finite-dimensional Λ. A formal argument for this transition is missing. One may reduce the IND-CPA security of REDOG to the IND-CPA security of the Modified DO.Gab-PKE using a game transition. However, we would recommend to make appropriate changes to the proof of [KKGK21] so that game $\mathcal{G}_0$ corresponds to an honest run of the REDOG system.

3. *Hash function* `hash`. In the proof of Theorem 1 of [KKGK21], it is mentioned in game $\mathcal{G}_2$ that m is completely random because of the usage of $\mathcal{H}$. This requires an extra game transition (between games $\mathcal{G}_1$ and $\mathcal{G}_2$) where the hash function is replaced by a value taken uniformly random. In this way, there is no dependence between error $\mathbf{e}$ and $\mathbf{m}'$ anymore in $\mathcal{G}_2$. Note that this introduces the advantage of breaking hash function `hash` in the security proof.

5.6.2 IND-CCA2 secure KEM using FO Transformation

The authors do not mention how REDOG, given that it is IND-CPA secure, can be transformed into a IND-CCA2 secure KEM using an appropriate FO Transformation. It is recommended to specify which FO transformation will be used and discuss how this influences the performance of the system. This should be considered as future work for the authors.

5.7 Round-2 C software

As noted above, the round-2 C software from the REDOG team appears to be work in progress. For example, the `rbc_97_vec_clear` function in the software varies in the number of arguments (and is not a `varargs` function).

Chapter 6

SMAUG-T: The Key Exchange Algorithm based on Module-LWE and Module-LWR

SMAUG [CCH$^+$22] and TiGER [PJP$^+$22] are lattice-based key encapsulation algorithms (KEMs). According to the recommendations after the first round, SMAUG and TiGER have been merged. The result of this merge is called SMAUG-T. SMAUG-T is essentially SMAUG (the parameter sets are now called SMAUG-T128, SMAUG-T192, and SMAUG-T256) together with a new parameter set called TiMER that, like TiGER, uses the D2 encoding to achieve smaller parameter sizes. The new parameter set achieves secret key size of 136 bytes while the ciphertexts are 608 bytes long. Together with a public key size of 672 bytes, this is the most efficient variant of SMAUG-T. In TiMER, the intermediate randomness that is used to derive encapsulated keys has reduced entropy compared to SMAUG-T128 (128 bits instead of 256). With respect to ciphertext size, this makes TiMER more efficient than TiGER and highly efficient state-of-the-art schemes like LightSaber. The authors of SMAUG-T point out that TiMER is better suited for lightweight devices as used in IoT applications. At the same time they caution that the use of the D2 encoding as well as the reduced entropy "may affect security in the future". As a compromise they recommend to use TiMER while restricting the number of operations the KEM system will perform. However, they do not provide concrete numbers for this.
The first report provides details on SMAUG and TiGER. For more information on relating SMAUG to Kyber [SAB$^+$22] see the thesis [Mer23] by Jorge Correa-Merlino. SMAUG-T is based on the hardness of the MLWE (Module

Learning with Errors) and the MLWR (Module Learning with Rounding) assumption.
In the following, we will provide a brief description and discussion of key features of SMAUG-T. It is instructive to compare it with the winner of NIST's PQC competition, Kyber. Naturally, the improvements employed in SMAUG-T can be based on a more mature field of knowledge, incorporating more recent research results as compared to Kyber and thus we expect to see improvements.

6.1 System description

Conceptually and on a high level, SMAUG-T closely follows the well-known template for building lattice-based KEMs as initiated by Regev [Reg05]. Section 2.2 provides more details on the Regev template.
Regev-like cryptosystems mimic the well-known ElGamal cryptosystem, where each ciphertext consists of an ephemeral public key and another part that is a function of the message. Decryption of the message is possible with a shared secret which is derived from the long-term key of the receiver $(\mathsf{pk}, \mathsf{sk})$ and the fresh ephemeral key $(\mathsf{epk}, \mathsf{esk})$ of the sender (so either from $(\mathsf{pk}, \mathsf{esk})$ by the sender or from $(\mathsf{epk}, \mathsf{sk})$ by the receiver). All Regev-like encryption systems have comparatively simple security proofs linking them to some LWE-like assumption. These underlying security assumptions mainly differ in i) the type of underlying lattice structure they are defined in, ii) the distribution that the secret key is chosen from, and iii) the error distribution that is used to bias (destroy) otherwise linear dependencies between public and secret variables. Applying these assumptions in constructions is simple. They guarantee that the long-term public key and the ephemeral public key of the ciphertext are indistinguishable from random values to passive attackers. For random public keys, the decrypted message is essentially also a random value to the attacker. In the security proof, this is enough to show that the scheme guarantees the relatively weak notion of IND-CPA security (indistinguishability under chosen plaintext attacks). Thus, even though there is a wide variety of LWE-like assumptions, each Regev-like cryptosystem chooses its algebraic setting and security assumption while following the same template.
In terms of provable security, the concrete choices of security assumptions influence the concrete correctness error and the achieved security level. Employing additional mechanisms often results in new tradeoffs. In TiMER, error correcting codes (ECCs) are applied that allow to decrease the size of ciphertexts and secret keys. At the same time ECCs are in this context often

regarded as a source of implementation problems that increase the risk of side-channel attacks.

6.1.1 Concrete parameters

When contrasted with SMAUG, SMAUG-T simply introduces a new set of parameters called TiMER while keeping the parameters of the SMAUG submission from the first round (after the update). Here we comment on the most recent choices while comparing with Kyber.
In general, the parameters should make the scheme i) have low bandwidth and high throughput, ii) have high concrete security supported by a proof to some security assumption, and iii) have adequate DFP.
Kyber uses MLWE with base ring $R_q = \mathbf{Z}_q[x]/(X^d + 1)$. The values of q and d are fixed to $q = 3329$ and $d = 256$ for all security levels; note that 256 divides $q - 1$ which permits using an almost complete NTT in R_q ($X^d + 1$ factors into quadratic factors). The smallest security level uses two blocks of size 256 and the number increases with the security level, so $n \in \{2, 3, 4\}$.
SMAUG-T uses MLWE and combines it with MLWR with power-of-2 cyclotomics for the ring at $d = 256$. Compared to Kyber it uses one more block for the highest security level, so $n \in \{2, 3, 5\}$. Like Saber [DKR+20] it works with a power-of-2 modulus $q \in \{1024, 2048\}$ and uses rounding.
TiMER uses the basic parameter set for SMAUG-T128. However, it uses a reduced message space of size $\{0, 1\}^{128}$ (instead of $\{0, 1\}^{256}$) to apply the D2 encoding. As a result we can have D2 decoding that expands from $\{0, 1\}^{128}$ back to $\{0, 1\}^{256}$ in the decryption process instead of rounding.
According to observations by Seongkwang Kim (KpqC-bulletin email from 29 Feb 2024) the SMAUG-T team has been made aware that instantiations of their XOF function by SHAKE128 should be replaced by SHAKE256 for higher security levels.

6.2 Security analysis

6.2.1 Generic lattice attackss

The results of the lattice estimator for both TiMER and all the versions of SMAUG-T can be found in Table 6.1. These are actually quite a bit higher than the reported security levels.

Version	Primal Core-SVP	Dual Core-SVP	BDD	Dual-Hybrid
TiMER	130	124	151	146
SMAUG-T128	128	122	149	145
SMAUG-T192	200	189	218	211
SMAUG-T256	342	320	354	339

Table 6.1: Estimated security levels for TiMER and SMAUG-T based on the lattice estimator [APS15]

6.2.2 Key search – the May attack on sparse secrets

The current parameter choices for SMAUG-T consider combinatorial attacks [May21] on earlier versions of SMAUG as pointed out by [Ber22a]. For SMAUG specifically, these attacks could not be used to push the security level under a critical threshold value for some security level. However the SMAUG team changed the parameters as well to allow for a larger security margin.

6.2.3 Decryption failures

After v2.1 of TiGER was announced, Lee [Lee23b] showed that the DFP was calculated incorrectly and that it was larger by a factor of 2^8. The TiGER team acknowledged the issue and provided a new version (v3.0) in which they changed the parameters for the highest security level. Since SMAUG-T uses SMAUG as a basis, these attacks do not transfer to the SMAUG-T without error correction.
The authors use the methodology employed by the Saber KEM system to justify the target decryption failure probability that they aim to achieve. Crucially, they differentiate between the number of users of a crypto system and the number of observable users. Then they proceed to use the latter, lower number, to bound the required decryption failure probability.

6.3 Distinctive efficiency considerations

Except for TiMER, the efficiency claims of SMAUG-T are the same as those of SMAUG. The efficiency levels that were claimed by TiGER for NIST level III and level V cannot be achieved. The only efficiency improvements gained are via the TiMER parameter set that has more aggressive parameter choices for the 128 security level.

Secret-key sizes. SMAUG-T chose a trade-off that focuses on overall good efficiency but particularly small secret key sizes. This has the advantage that secret keys can more cheaply be stored in physically protected memory locations (trusted memory). Many server applications use such memory locations in the form of hardware security modules (HSMs). Trusted memory tends to be more expensive but features additional security guarantees that are typically not reflected in common security definitions but may matter in practice. Importantly, keys stored in trusted memory locations offer better protection against physical attacks like power analysis or fault injections. Trusted memory locations usually make sure that the secret key cannot leave the safe memory location in an unprotected form. Instead, the interface to the trusted memory only allows to apply the secret key to decrypt (or generate signatures on) input values sent into the HSM. At the same time, while security is higher, efficiency is often lower than computations on general CPUs that access the secret key from RAM. However, this is compensated by the fact that the trusted location computes on the secret keys in parallel to what the general CPU computes.

The main technique to obtain small secret keys is by using a sparse distribution, where most entries of the secret key vector are zero, and only storing the non-zero entries, together with the index of their component. However, drawing sparse secret keys has the downside of increasing the attack surface, particularly allowing combinatorial attacks, see section 6.2.2.

TiMER additionally employs D2 encoding. As a result it can avoid rounding in the decryption operation. This helps to decrease the probability for decryption failures. As stated, this comes at the cost of less entropy in the key derivation.

While small ciphertexts and public keys are generally useful to reduce transmission size, it is unclear whether, in practice, small secret keys will be of much importance. Of course, small secret keys can be useful since they can be protected better physically. However, it is unclear what the additional costs are and how much weight this use case should have overall.

Another cautionary note regarding the secret-key sizes is that SMAUG-T decapsulation requires not just the secret key but also the public key. This means that SMAUG-T's secret-key sizes should not be directly compared to the secret-key sizes for, e.g., Kyber and Saber, where the secret key includes everything needed for decapsulation.

Ciphertext sizes. SMAUG-T has better ciphertext sizes than Kyber and Saber, except that it has the same ciphertext size as Saber (smaller than Kyber) at the top security level. Specifically, ciphertext sizes are 608 bytes for TiMER, 672 bytes for SMAUG-T128, 736 bytes for LightSaber, 768 bytes for Kyber-512, 1024 bytes for SMAUG-T192, 1088 bytes for Kyber-768, 1088 bytes for Saber, 1472 bytes for SMAUG-T256, 1472 bytes for FireSaber, and 1568 bytes for Kyber-1024.

Public-key sizes. For public-key size, SMAUG-T is better at the lowest security level but worse at the highest security level. Specifically, public-key sizes are 672 bytes for TiMER, 672 bytes for SMAUG-T128, 736 bytes for LightSaber, 800 bytes for Kyber-512, 992 bytes for Saber, 1088 bytes for SMAUG-T192, 1184 bytes for Kyber-768, 1568 bytes for Kyber-1024, 1664 bytes for FireSaber, and 1792 bytes for SMAUG-T256.

CPU cycles. It is safe to predict that any slowdown in CPU cycles from Kyber to SMAUG-T will be outweighed by the SMAUG-T size improvements, given the relative costs of bytes and cycles (see, e.g., [Ber23b]) and given techniques for polynomial multiplications in lattice-based cryptography (see, e.g., [CHK+21]). However, optimized constant-time SMAUG-T software is not currently available; see Section 6.5. The SMAUG-T specification suggests that SMAUG-T outperforms Kyber in CPU cycles, but this appears to be based primarily on a comparison of reference software, which is not a valid predictor of the speed of optimized software.
The authors provide a SCA-protected implementation of the discrete Gaussian sampler that they need. This increases the cycle count by around 110 percent for SMAUG-T256 and around 75 percent for the remaining parameter sets.

6.4 Considerations on provable security

SMAUG-T provides a detailed security proof in its documentation. The authors argue convincingly that the proof also applies to TiMER, where random values are simply encoded before usage in SMAUG-T.PKE. To obtain a security proof for SMAUG-T.KEM, the authors deviate from Kyber that uses the FO transform with implicit rejection and ciphertext contribution while relying on [HHK17].
SMAUG-T uses the FO transform with implicit rejection and no ciphertext contribution relying on [BHH+19]. The authors argue why avoiding ciphertext contributions can improve speed. Moreover, when deriving the shared secrets via a random oracle, they now also incorporate (a prefix of) the public key [DHK+21]. This protects better against multi-target attacks.

Unlike before, the SMAUG-T team does not invoke additional non-standard assumptions to use [HHK17].
We stress that the proof of IND-CCA security for SMAUG-T.KEM relies on a generic application of the FO transformation. This guarantees tight results in the random oracle model and non-tight results in the QROM. At the same time, this makes the crucial contribution of SMAUG-T the proof of IND-CPA security. This proof is well structured and generally in sufficient detail. The security reduction is tight. The underlying assumptions are:

- The MLWE assumption that is used to argue that real public keys are indistinguishable from random MLWE samples.
- The MLWR assumption that is used to argue that parts of the real ciphertext are indistinguishable from random MLWR samples.
- The output of XOF is pseudo-random. This assumption is not formalized.
- The output of sampling algorithms is pseudo-random. This assumption is not formalized.

The analysis includes a derivation of the completeness property of SMAUG-T.KEM.
To improve the argumentation, the SMAUG-T submission should introduce the formal definitions required for the two pseudo-randomness assumptions and a discussion of why it is plausible that they hold in practice.

6.4.1 Prefix hashing

We note that the cryptographic description of SMAUG-T does not indicate that the hashed public key is used to derive the shared secret. However, the pseudo-code and the text indicate this along with a motivation: "The public key is additionally fed into the hash function with the message to avoid multi-target decryption failure attacks". This means that, practically, for encapsulation the sender has to hash the entire public key. Using the result from [DHK$^+$21], the efficiency of SMAUG-T can be improved by using a different variant of the Fujisaki-Okamoto transform. Here, only prefixes of the public key are hashed, which can greatly increase the speed of encapsulation. Moreover, one can now rely on the results from [DHK$^+$21] to provide a security proof that does not rely on the underlying injectivity of the IND-CPA secure PKE scheme.

This new transformation reduces the running time of the scheme. It is shown that the security bounds for that FO transform using prefix hashing are exactly the same as the security bounds for the FO transform without prefix hashing, see Table 6.2.

Variant	Advantage QROM
FO SMAUG-T	$2\sqrt{(q_G+2)\mathrm{Adv}_{\mathsf{PKE}}^{\text{IND-CPA}}+\frac{16}{\lvert\mathcal{M}\rvert}}+\mathrm{Adv}_{\mathsf{PKE}}^{DF}\sqrt{\frac{q_H}{\lvert\mathcal{M}\rvert}}+\epsilon_{\text{inj}}$
FO Prefix Hash	$2\sqrt{q\mathrm{Adv}_{\mathsf{PKE}}^{\text{IND-CPA}}}+4q\sqrt{\frac{1}{\lvert\mathcal{M}\rvert}}+\frac{1}{2^l}+4(q_F+1)\sqrt{\frac{1}{2^\lambda}}+16q^2\delta+\frac{1}{\lvert\mathcal{M}\rvert}$

Table 6.2: Comparison of the security bounds for the FO transform used for SMAUG-T and the FO transform using prefix hashing. We assume that $n = 1, q_C = 1$. For FO SMAUG-T, q_G, q_H are the number of hash queries. The PKE is assumed to be ϵ_{inj}-injective. For the FO prefix hash, q_F is the number of QROM hash queries, q_D the number of decapsulation queries, $q = q_D + q_F + 1$, δ is the correctness error, l is the min-entropy of $ID(\mathsf{pk})$, and λ is the bit length of secret seed.

Observe that the security of the FO transform using prefix hashing does not include a term on injectivity. Therefore, as the security bounds are comparable, utilizing the prefix hash technique avoids the injectivity assumption necessary for the security proof of SMAUG-T.
As a reference, we include the speed improvements of using this FO transformation for Kyber and Saber [DHK+21], see Table 6.3.

6.5 Round-2 C software

The round-2 software documentation does not make clear how the software relates to the specification. The reference implementations appear to cover only SMAUG. There is an "additional" implementation that is perhaps intended to be TiMER, but this does not appear to be documented.
The software does not follow the standard decapsulation API: it changes the API to take the public key as a separate argument. An embedded device carrying out decapsulation needs to have the private key *and* the public key, as noted in Section 6.3, whereas submissions following the standard API need only the private key.
We modified the software to follow the standard decapsulation API; internally, this means including a copy of the public key inside the private key. This makes the private keys longer than reported in the documentation. This

is necessary for comparability of the private-key size to other submissions, and for future integration of the software into SUPERCOP.

Inspecting thousands of SMAUG-T128 secret keys generated by the reference and optimized software showed that the nonzero coefficients of the keys were always at positions 1, 10, 16, etc. The signs are random, but the known positions reduce the secret from 512 dimensions to 140 dimensions, which is not secure against lattice attacks.

Some manual code analysis detected another software bug: the software is using `0xfffffffff` instead of `0xffffffff`. Fixing this changed the known positions to a different set of 140 positions: 4, 7, 8, 13, etc.

Further code analysis shows that, for most choices of randomness, the software chooses positions $1+((2^{32}-1) \bmod (512-140))$, $1+((2^{32}-1) \bmod (512-139))$, etc. There are many collisions in these numbers: e.g., there are 14 occurrences of 256, coming from many divisors of $2^{32}-256$. The code also moves previously initialized array entries to the end of the array, so the collisions produce a pileup of positions near the end of the array.

We compared this to the specified algorithm (Figure 2 on page 12), concluding that the specified algorithm (1) is different and (2) does not work: "buf[idx] / i" in the specified algorithm is almost always out of range. Presumably the intent was mod instead of division, and then the specified algorithm appears to use Fisher–Yates sampling.

The SMAUG-T team indicated that it had already found and fixed the known-position bug in March 2024, in `https://github.com/hmchoe0528/SMAUG-T_public/releases/tag/v3.0.1`. Further bugs were then pointed out by Moon Sung Lee, Nari Lee, and Hansol Ryu in kpqc-bulletin email from 23 May 2024 22:59:23 -0700.

A separate problem is that there are many timing leaks in the software (including timing leaks from this sampling). We posted an attack demo (kpqc-bulletin email from 11 May 2024 20:12:45 +0200) that often recovers a long-term SMAUG-T128 secret key from a few minutes of decapsulation timings of the optimized implementation in the SMAUG-T package. The attack demo does not exploit the known-position bug.

The SMAUG-T team has indicated that a software update is in progress.

Scheme		Original	Prefix Hash Work	Speed-up
Kyber-512	KGen	23562	12883	45%
	Enc	37144	16981	54%
	Dec	28595	28529	0%
Kyber-768	KGen	40487	25272	38%
	Enc	55726	27624	50%
	Dec	43553	43442	0%
Kyber-1024	KGen	55770	38815	30%
	Enc	77011	40692	47%
	Dec	61470	61473	0%
LightSaber	KGen	42169	36220	14%
	Enc	57831	39232	32%
	Dec	57780	57806	0%
Saber	KGen	74577	64180	14%
	Enc	95958	69304	28%
	Dec	95388	95301	0%
FireSaber	KGen	116178	102101	12%
	Enc	142034	109203	23%
	Dec	142957	143090	0%

Table 6.3: Table from [DHK+21] showing median Skylake cycle counts of 10000 executions of Kyber and Saber using either the original CCA transform or the improved transform from prefix hash work. The "original" version of Kyber and Saber are the Round 3 submissions to the NIST postquantum standardization process.

Chapter 7

SUPERCOP results for KEM software

Figures 7.1, 7.2, 7.3, 7.4, 7.5, 7.6 and 7.7 are scatterplots showing various measurements from SUPERCOP version 20240909 for NTRU+ (`ntruplus`). As selected baselines for comparison, these figures also show measurements for `kyber` (not reflecting incompatible Kyber changes after the latest submission of Kyber to SUPERCOP), `mceliece`, `ntruhps`, `ntruhrss`, and `sntrup`. For reasons explained earlier in this report, PALOMA, REDOG, and SMAUG-T are not included at this time.
Figure 7.1 is a CPU-independent plot of sizes vs. sizes. Figures 7.2, 7.4, and 7.6 plot sizes vs. timings for a computer named `titan0` with an Intel Haswell CPU, while Figures 7.3, 7.5, and 7.7 plot sizes vs. timings for a computer named `cezanne` with an AMD Zen 3 CPU. The Haswell microarchitecture was introduced by Intel in 2013 and was designated as a comparison platform by NIST in January 2019. Zen 3 is an example of a newer microarchitecture: it was introduced by AMD in 2020. Both of these microarchitectures support 256-bit AVX2 vector instructions, and all dots in the figures are for code written to use those instructions. In some cases there are noticeable differences in timings between `titan0` and `cezanne` (e.g., for NTRU+ using SHA-256 instructions on `cezanne`). Timings are median cycle counts; see `https://bench.cr.yp.to` for tables including quartiles.

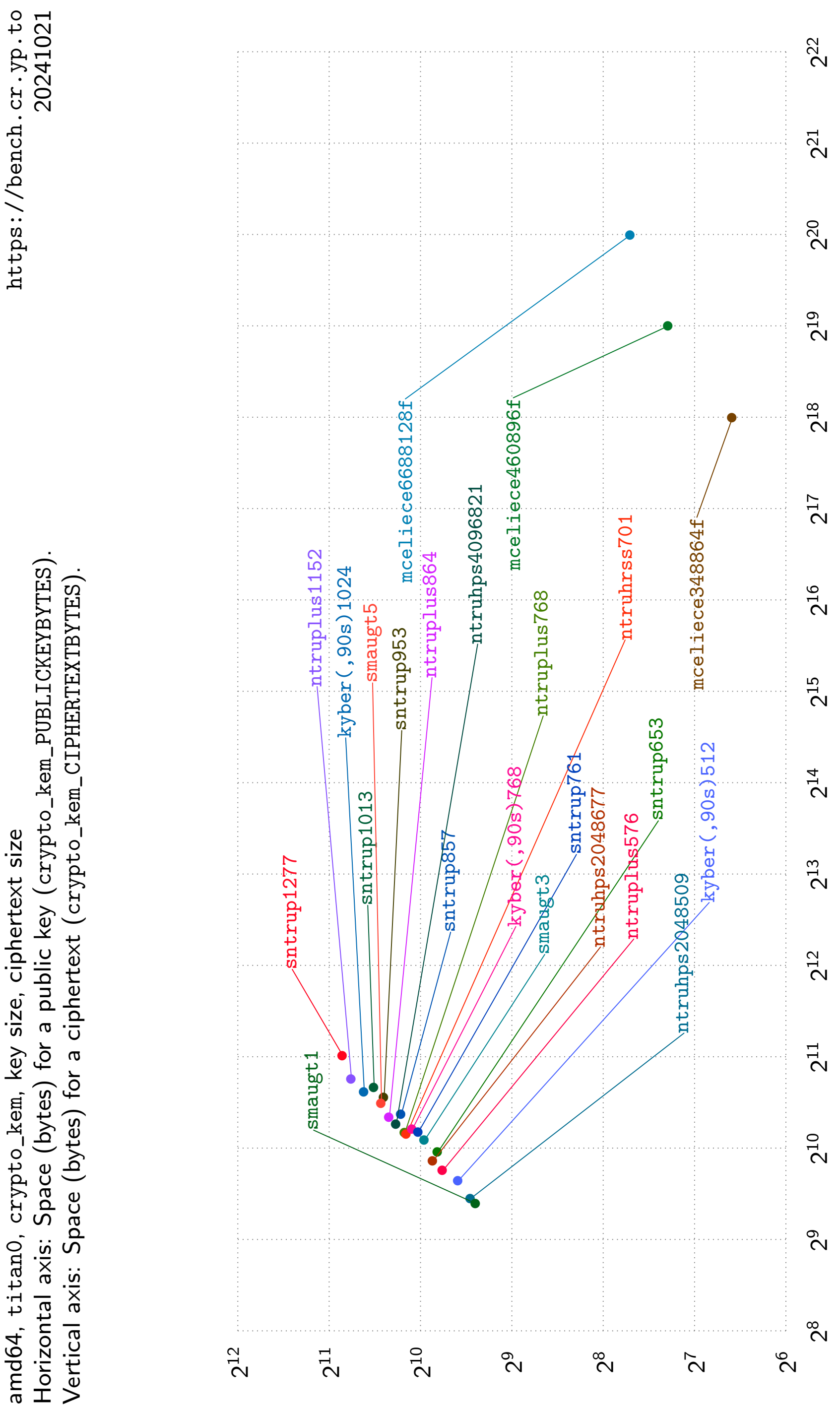


Figure 7.1: Bytes for a ciphertext vs. bytes for a public key.

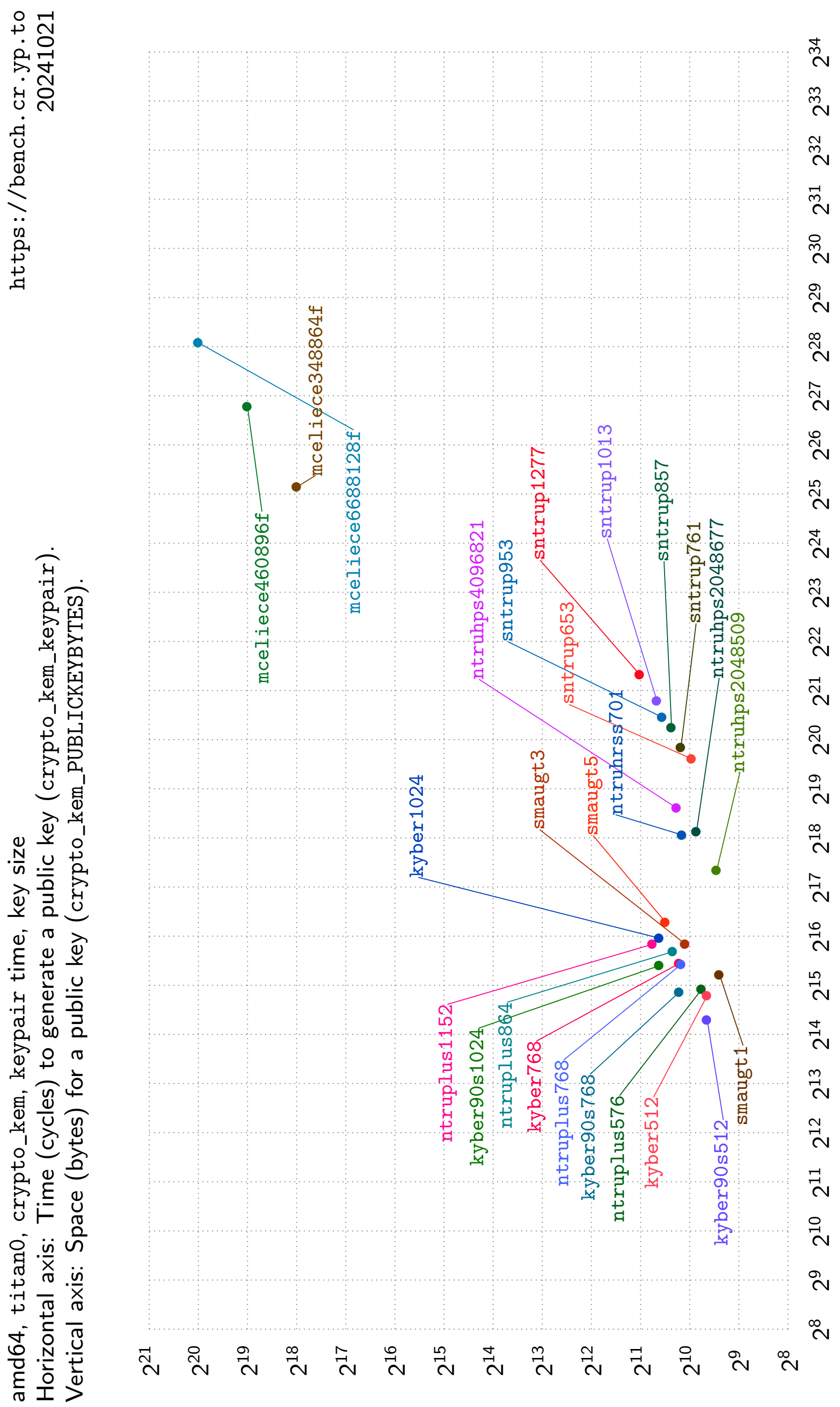


Figure 7.2: Bytes for a public key vs. cycles for key generation. Measurements collected on `titan0`, which has an Intel Haswell CPU.

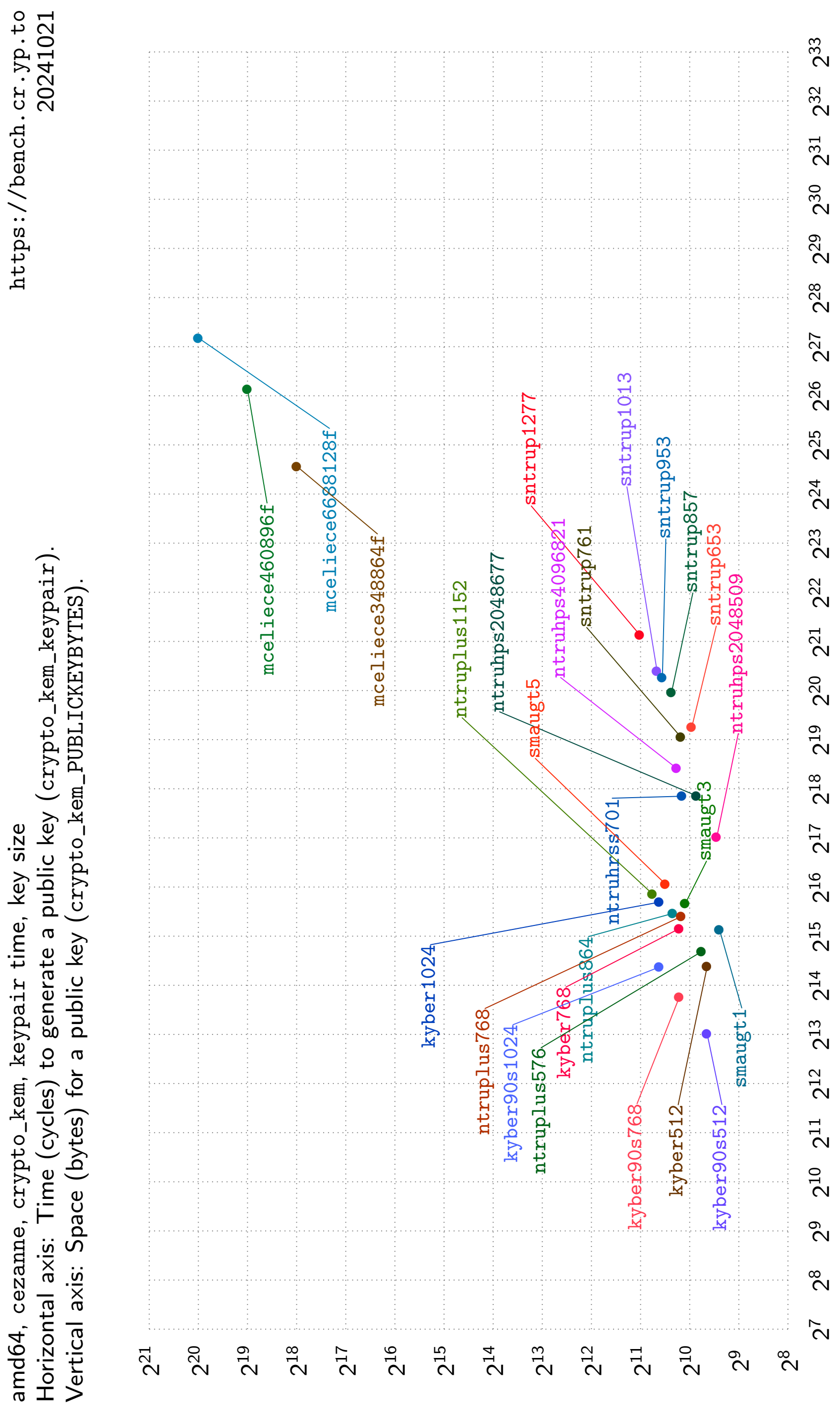


Figure 7.3: Bytes for a public key vs. cycles for key generation. Measurements collected on `cezanne`, which has an AMD Zen 3 CPU.

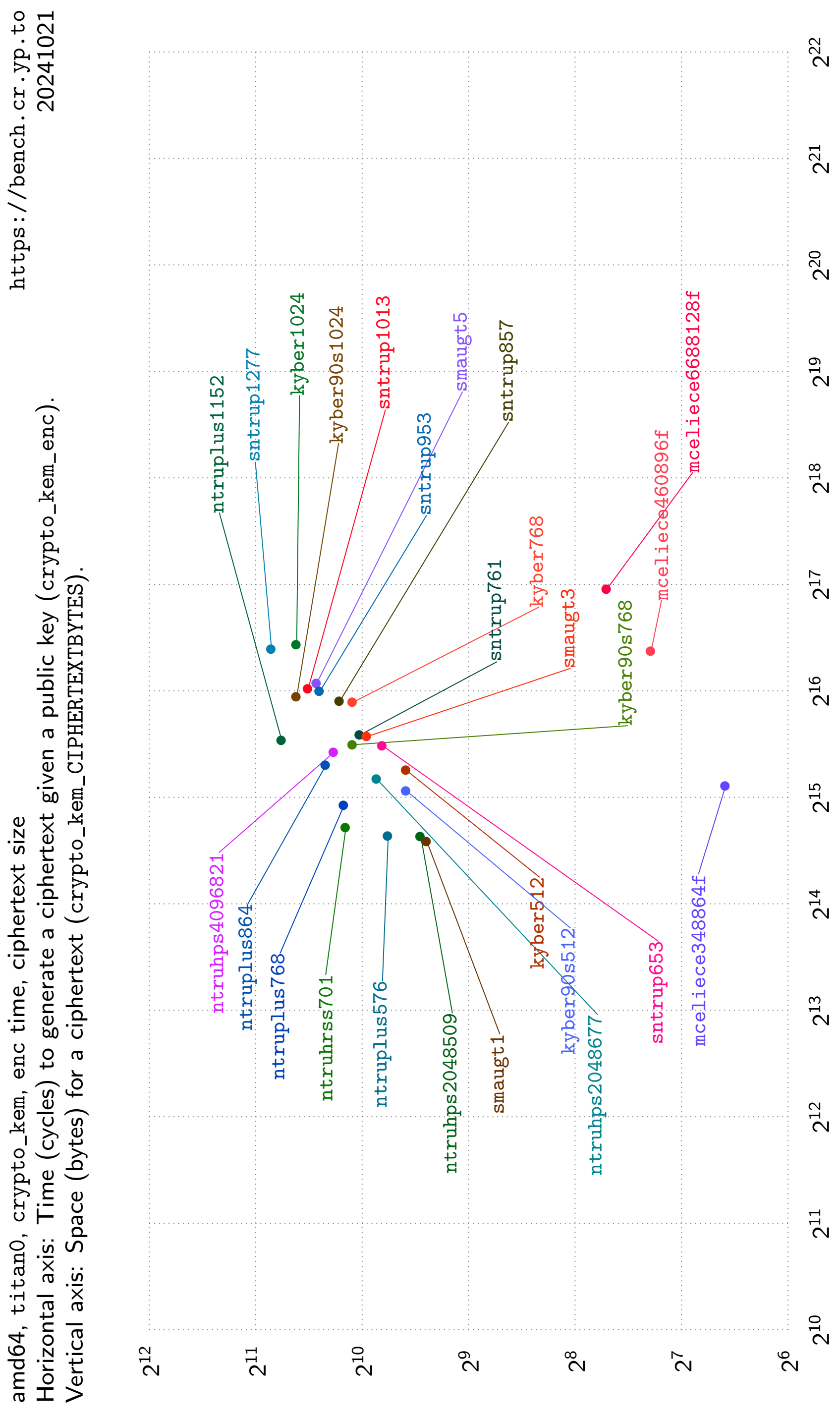


Figure 7.4: Bytes for a ciphertext vs. cycles for encapsulation. Measurements collected on `titan0`, which has an Intel Haswell CPU.

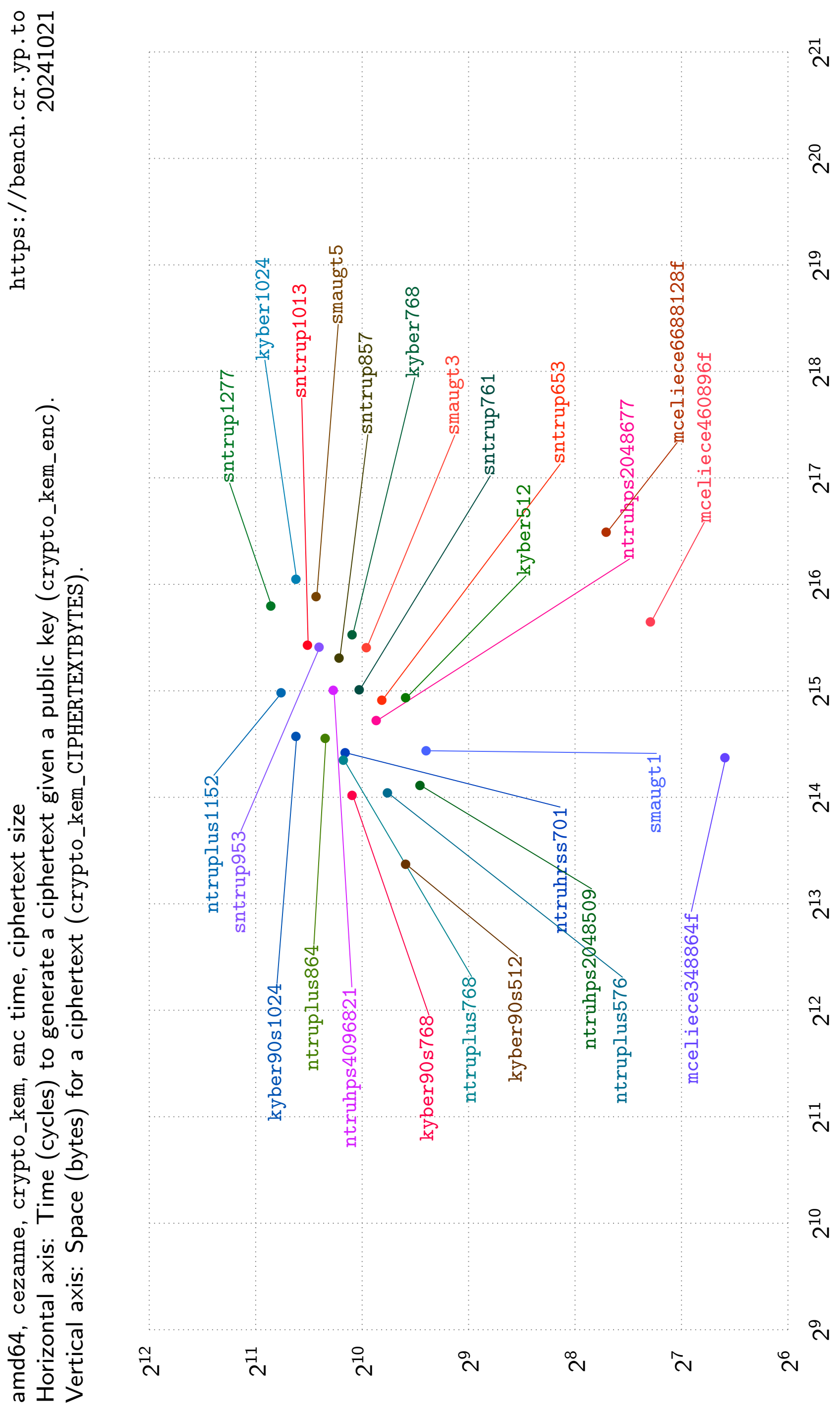


Figure 7.5: Bytes for a ciphertext vs. cycles for encapsulation. Measurements collected on `cezanne`, which has an AMD Zen 3 CPU.

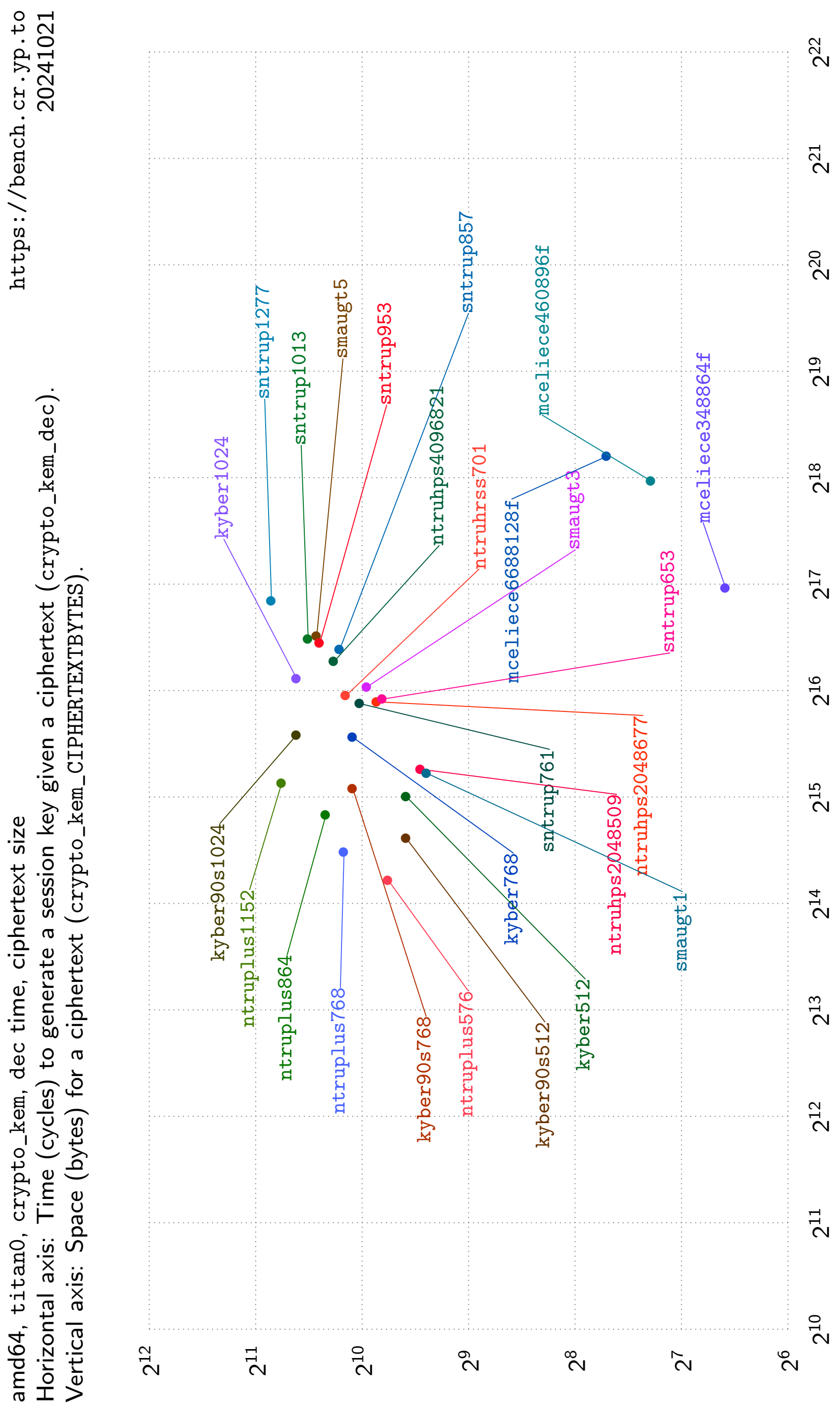


Figure 7.6: Bytes for a ciphertext vs. cycles for decapsulation. Measurements collected on `titan0`, which has an Intel Haswell CPU.

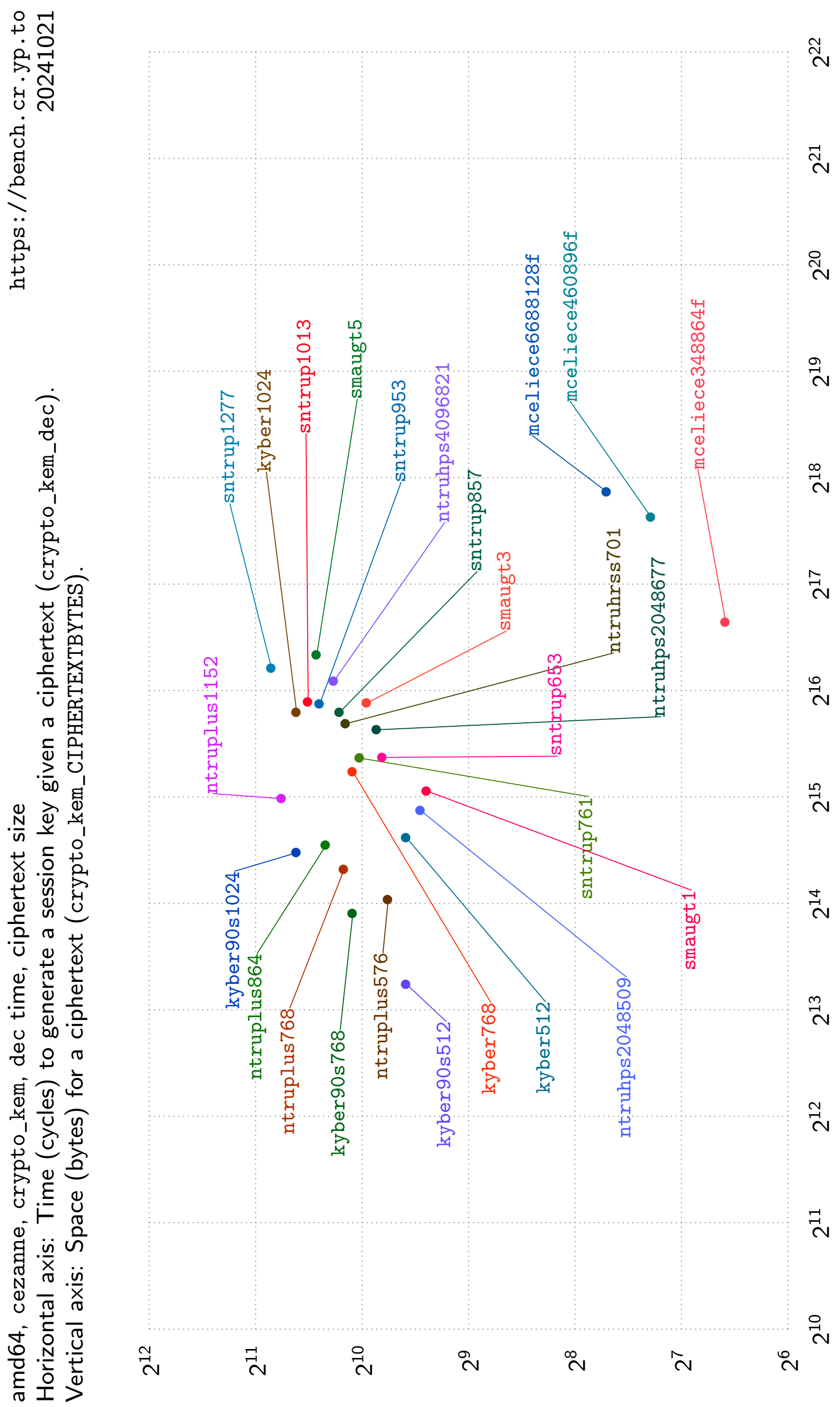


Figure 7.7: Bytes for a ciphertext vs. cycles for decapsulation. Measurements collected on `cezanne`, which has an AMD Zen 3 CPU.

Part II

Digital Signature Algorithms

Chapter 8

AIMer: The AIMer Signature Scheme

AIMer [KCC^{+}23] is a signature scheme designed using the MPC-in-the-head [IKOS09] paradigm to design an identification scheme which is then transformed into a signature scheme.
The core idea of the MPC-in-the-head paradigm in the context of signatures is to create an identification scheme that proves knowledge of a preimage for a one-way function in zero-knowledge. For this, the prover simulates a multiparty computation (MPC) that evaluates the one-way function on input of the secret-shared preimage. For this, every simulated party obtains a secret-share of the preimage. Then the MPC computation is simulated, the prover commits to the internal state of each party, and sends the commitments together with all the communication between the parties to the verifier. The verifier then challenges the prover to open the full state of all but one party, which allows the verifier to probabilistically check that the prover did not cheat in the MPC simulation. At the same time, the security of the MPC protocol guarantees that even colluding parties cannot learn anything about the input of another party except what can be derived from the computation outcome. Thereby, it is guaranteed that the verifier does not learn anything about one of the secret shares and thereby the preimage remains hidden.
This identification scheme is then made non-interactive using the Fiat-Shamir transform described in Section 2.3.3. The security of the resulting scheme is (provably) based on the security of several hash functions, use as pseudorandom generators, commitment scheme, and message hash, as well as the security of the one-way function for which knowledge of the preimage is proven. An important aspect is that while the hash functions used for the former applications can simply be instantiated with standardized hash functions / XOFs like SHA2, SHA3 or SHAKE, this does not hold for the

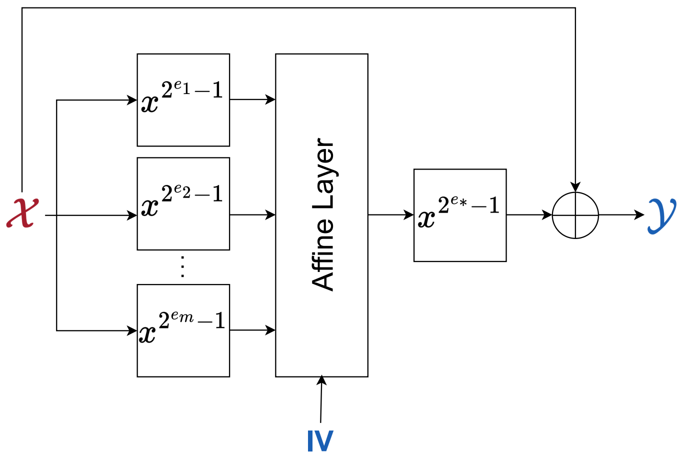


Figure 8.1: Structure of AIM. The input $\mathcal{X}$ and output $\mathcal{Y}$ are in $\mathbb{F}_{2^n}$; the S-box (non-linear) layer consists of Mersenne-number power maps; the affine layer is an $n \times mn$ binary matrix and a binary vector randomized using IV.

one-way function. Given that the latter is evaluated in an MPC, its performance in this setting determines the performance of the signature scheme. Hence, proposals following the MPCitH design make use of special one-way functions that are optimized for the use in MPC[1]. For example, Picnic makes use of the third version of LowMC which was first proposed in [ARS+15] as its underlying one-way function. In the case of AIMer, the underlying one-way function is a new design called AIM [KHS+22].
Below, we first discuss cryptanalysis results regarding AIM. Afterwards, we comment on the provable security claims made in the AIMer specification. Finally, we conclude.

8.1 Cryptanalysis results for AIM

The first version of AIMer used AIM [KHS+22] as its underlying one-way function. The design of AIM is depicted in Figure 8.1.
The parameters for different versions of AIM are described in Table 8.1. Let $\mathcal{X}, \mathcal{Y}$ be input and output of AIM respectively. Let $\mathcal{Z}_i$ be the output of the i^{th} S-box, $\mathcal{S}$ be the output of the affine layer, and $B_i(x) = \sum_{j=0}^{\lambda-1} a_{i,j} x^{2^j} + a_{i,\lambda}$ represent the affine layer applied to the state after the i^{th} Sbox. AIM can be

[1]Specifically, these designs try to minimize the use of multiplication as it is the by far most costly operation in common MPC protocols.

Scheme	λ	Field	m	e_1	e_2	e_3	e_*
AIM-I	128	$\mathbb{F}_{2^{128}}$	2	3	27		5
AIM-III	192	$\mathbb{F}_{2^{192}}$	2	5	29		7
AIM-V	256	$\mathbb{F}_{2^{256}}$	3	3	53	7	5

Table 8.1: Parameters for different instances of AIM.

described as:

$$\begin{aligned}\mathcal{Z}_i &= \mathcal{X}^{2^{e_i}-1} \text{ for } 1 \leq i \leq m,\\ \mathcal{S} &= \sum_{i=1}^{m} B_i(\mathcal{Z}_i),\\ \mathcal{Y} &= \mathcal{S}^{2^{e_*}-1} + \mathcal{X}.\end{aligned}$$

The structure of AIM was exploited in two works and in both cases, the complexity of the proposed attacks were smaller than the required security level.

Fast exhaustive search

In [LMØM23], Fukang, Mahzoun, Øygarden, and Meier used fast exhaustive search [BCC+10] to break AIM. The versions with 128/192/256-bit security are shown to be broken with complexity $2^{115}/2^{178}/2^{241}$. The attack exploits the low degree of the non-linear operations of AIM. Similar observation was made by Markku-Juhani O. Saarinen[2] which shows that AIM does not reach the claimed security level.
As shown in Figure 8.2, one can write:

$$\mathcal{X} = \mathcal{S}^{2^{e_*}-1} + \mathcal{Y}$$

And for each variable $\mathcal{Z}_i$, one can write:

$$\mathcal{Z}_i = \left(\mathcal{S}^{2^{e_*}-1} + \mathcal{Y}\right)^{2^{e_i}-1} \Rightarrow \mathcal{Z}_i = \sum_{j=0}^{2^{e_i}-1} \mathcal{Y}^j \mathcal{S}^{2^{e_*}-1(2^{e_j}-1-j)}$$

Then, $\mathcal{Z}_m$ can be written in two different ways as:

$$\mathcal{Z}_m = B_m^{-1}\left(c + \mathcal{S} + \sum_{i=0}^{m-1} B_i\left(\left(\mathcal{S}^{2^{e_*}-1} + \mathcal{Y}\right)^{2^{e_i}-1}\right)\right), \tag{8.1}$$

$$\mathcal{Z}_m = \left(\mathcal{S}^{2^{e_*}-1} + \mathcal{Y}\right)^{2^{e_m}-1}, \tag{8.2}$$

[2]https://groups.google.com/a/list.nist.gov/g/pqc-forum/c/BI2ilXblNy0

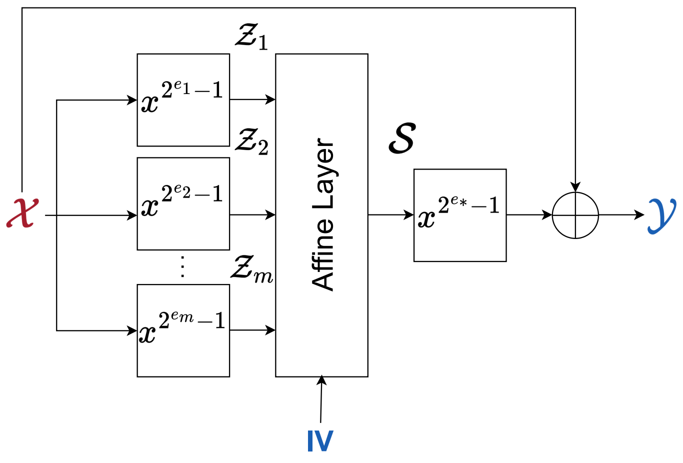


Figure 8.2: The variables used to model AIM are denoted with $\mathcal{X}$, $\mathcal{Z}_i$ for $1 \leq i \leq m$, $\mathcal{S}$.

Scheme	λ	Field	$m+1$	Algebraic Degree	Time	Memory	Complexity
AIM-I	128	$\mathbb{F}_{2^{128}}$	3	10	$2^{136.2}$	$2^{61.7}$	2^{115}
AIM-III	192	$\mathbb{F}_{2^{192}}$	3	14	$2^{200.7}$	$2^{84.3}$	2^{178}
AIM-V	256	$\mathbb{F}_{2^{256}}$	4	15	$2^{265.0}$	$2^{95.1}$	2^{241}

Table 8.2: Complexity of breaking instances of AIM using fast exhaustive search.

which gives an equation in $\mathcal{S}$, which can be solved using fast exhaustive search technique and the complexity is summarized in Table 8.2.

Linearization attack

It was shown in [ZWY+23] that the linearization method breaks AIM. The attack targets the design flaw in the first non-linear operation of AIM where all three Mersenne powers have the same input. Each S-box can be written as:

$$x^{2^{e_i}-1} = \left(x^d\right)^{s_i} \cdot x^{2^{t_i}},$$

and by guessing the value of x^d, one will have a linear system to solve. The summary of the attacks and their complexities are summarized in Table 8.3.

Scheme	λ	Field	$m+1$	d	t_1	t_2	degree of freedom	Complexity
AIM-I	128	$\mathbb{F}_{2^{128}}$	3	5	1	1	4	$2^{125.7}$
AIM-III	192	$\mathbb{F}_{2^{192}}$	3	45	8	8	12	$2^{186.5}$
AIM-V	256	$\mathbb{F}_{2^{256}}$	4	3	0	0	2	$2^{254.4}$

Table 8.3: Complexity of breaking instances of AIM using linearization method.

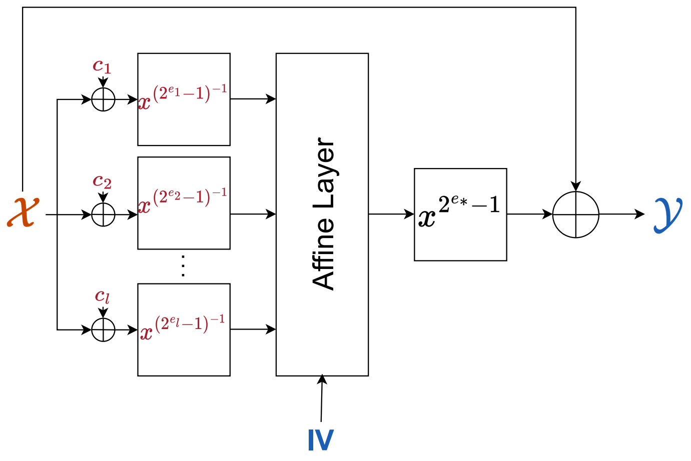


Figure 8.3: Design of AIM2

8.2 AIM2

After AIM was broken, a new design was proposed AIM2 [KHSL23]. AIM2 uses the non-linear layers with larger exponents, and each input to a non-linear function is guaranteed to be different from other inputs in the same step. AIM2 works as follows:

$$\mathrm{AIM2}(\mathrm{IV}, \mathcal{X}) = \mathcal{Y} = \left(\sum_{i=1}^{\ell} \sum_{j=0}^{\lambda-1} \left(\alpha_{i,j} (\mathcal{X} + c_i)^{(2^{e_i}-1)^{-1}} \right)^{2^j} + \alpha_\lambda \right)^{2^{e_*}-1} + \mathcal{X},$$

where $\alpha_{i,j}, \alpha_\lambda$ values are constants derived from IV, c_i values are public constants , and $\mathcal{X}, \mathcal{Y}$ and constants are elements of $\mathbb{F}_{2^\lambda}$. The design of AIM2 is depicted in Figure 8.3.
Recommended sets of parameters for AIM2 instances for three security levels $\lambda \in \{128, 192, 256\}$ are given in Table 8.4.

Scheme	λ	n	ℓ	e_1	e_2	e_3	e^*
AIM2-I	128	128	2	49	91	-	3
AIM2-III	192	192	2	17	47	-	5
AIM2-V	256	256	3	11	141	7	3

Table 8.4: Recommended sets of parameters of AIM2

8.2.1 Security of AIM2

The main focus in cryptanalysis of AIM2 is against algebraic attacks, and the attacks that broke AIM. In case of algebraic attack, different approaches to model AIM2 as a multivariate polynomial system are analyzed. The summary of the complexity of algebraic attacks against AIM2 is described in Table 8.5.

Scheme	**variables**	**(equations, Deg)**	**dreg**	**Time (bits)**
AIM2-I	n	$(n, 60)$	-	-
	$2n$	$(3n, 2)$	20	207.9
	$3n$	$(12n, 2)$	18	185.3
AIM2-III	n	$(2n, 114)$	-	-
	$2n$	$(3n, 2)$	30	301.9
	$3n$	$(12n, 2)$	26	262.4
AIM2-V	n	$(2n, 172)$	-	-
	$2n$	$(n, 2) + (2n, 38)$	50	513.5
	$3n$	$(6n, 2)$	50	503.7
	$4n$	$(18n, 2)$	41	411.4

Table 8.5: Summary of the complexity of algebraic attacks against AIM2 [KHSL23].

The AIM2 designers have thoroughly analyzed potential algebraic attack methods using various techniques based on the current parameters. The security model of AIM2 restricts attackers to $O(1)$ data complexity, making statistical attacks appear impractical at first glance.

8.3 Solving AIM2 System

The security of AIM2 is extensively analyzed against algebraic attacks. As the first step, AIM2 is modeled as an overdetermined polynomial system, and the complexity of solving that polynomial system is used to argue the security of AIM2. The reason to model AIM2 as overdetermined polynomial system is that it is believed that generally, adding linearly independent equations to a polynomial system reduces its solving complexity [LMM24]. While overdetermined systems are expected to be easier to solve, they are usually more complicated, and thus finding structures among the polynomials in the system is harder. We show that in the case of AIM2, using a determined polynomial system with fewer polynomials results in a simplified polynomial system that can be used to break AIM2 using exhaustive search.

8.3.1 Exhaustive Search on AIM2

Let $R = \mathbb{F}_{2^\lambda}[x, y_1, \dots, y_m]$. Having an output value h, the preimage problem for AIM2 can be modeled the following polynomial system over R:

$$\begin{aligned}(x \oplus c_1) &= y_1^{2^{e_1}-1}\\ (x \oplus c_2) &= y_2^{2^{e_2}-1}\\ &\vdots\\ (x \oplus c_m) &= y_m^{2^{e_m}-1}\\ \left(\alpha_\lambda \oplus \sum_{j=1}^{m}\sum_{i=0}^{\lambda-1}\alpha_{ji}y_j^{2^i}\right)^{2^{e_*}-1} &= (x \oplus h).\end{aligned}$$

We can replace x with $y_1^{2^{e_1}-1} - c_1$ and rewrite the equations as:

$$\begin{aligned}\left(y_1^{2^{e_1}-1} \oplus c_1 \oplus c_2\right) &= y_2^{2^{e_2}-1}\\ &\vdots\\ \left(y_1^{2^{e_1}-1} \oplus c_1 \oplus c_m\right) &= y_m^{2^{e_m}-1}\\ \left(\alpha_\lambda \oplus \sum_{j=1}^{m}\sum_{i=0}^{\lambda-1}\alpha_{ji}y_j^{2^i}\right)^{2^{e_*}-1} &= (y_1^{2^{e_1}-1} \oplus c_1 \oplus h).\end{aligned}$$

Which can be rewritten as:

$$\left(y_1^{2^{e_1}}y_2 \oplus y_1y_2(c_1 \oplus c_2)\right) = y_1y_2^{2^{e_2}} \tag{8.3}$$

$$\vdots$$

$$\left(y_1^{2^{e_1}}y_m \oplus y_1y_m(c_1 \oplus c_m)\right) = y_1y_m^{2^{e_m}} \tag{8.4}$$

$$\begin{aligned}y_1\left(\alpha_\lambda \oplus \sum_{j=1}^{m}\sum_{i=0}^{\lambda-1}\alpha_{ji}y_j^{2^i}\right)^{2^{e_*}} &= y_1^{2^{e_1}}\left(\alpha_\lambda \oplus \sum_{j=1}^{m}\sum_{i=0}^{\lambda-1}\alpha_{ji}y_j^{2^i}\right)\\ &\oplus y_1\left(\alpha_\lambda \oplus \sum_{j=1}^{m}\sum_{i=0}^{\lambda-1}\alpha_{ji}y_j^{2^i}\right)(c_1 \oplus h).\end{aligned} \tag{8.5}$$

Primitive	m	Solving complexity	AIM2 complexity	Attack complexity
AIM2-I	2	144.8	147	125.8
AIM2-III	2	210.2	212.3	189.9
AIM3-III	3	276.2	277.7	254.4

Table 8.6: Solving complexity refers to the complexity of solving an instance of AIM2 using our exhaustive search approach. AIM2 complexity is the bit complexity of AIM2 circuit as reported in [KHSL23]. Attack complexity is total security of instances of AIM2. The security is measured by converting the bit complexity of the attack to the number of calls to the underlying instance of AIM2.

which is a quadratic polynomial system with m equations and variables. Transforming it to a boolean polynomial system over $\mathbb{F}_2[y_{ji}], 1 \leq j \leq m, 0 \leq i < \lambda$ results in the following polynomial system:

$$\sum_{j=0}^{\lambda-1} \sum_{i=0}^{\lambda-1} \beta_{kij} y_{1i} y_{kj} + \sum_{j=0}^{\lambda-1} \gamma_{kj} y_{kj} + \gamma_{k\lambda} = 0 \quad 1 \leq k \leq m \tag{8.6}$$

which is a quadratic polynomial system with $\lambda(m)$ equations and variables. As a result, guessing the λ boolean variables $y_{11}, \ldots, y_{1\lambda-1}$ results in a linear system with $(m-1)\lambda$ variables and $m\lambda$ equations. The coefficient matrix of this linear system has the form:

$$M_{\lambda m, \lambda(m-1)} = \begin{bmatrix} A_1 & O & \cdots & O \\ O & A_2 & \cdots & O \\ \vdots & \vdots & \cdots & \vdots \\ O & O & \cdots & A_{m-1} \\ B_1 & B_2 & \cdots & B_{m-1} \end{bmatrix}, \tag{8.7}$$

where $A_i, B_i \in \mathbb{F}_2^{\lambda \times \lambda}$. To solve the system, we have the complexity

$$(m-1)\left(\lambda^{\omega} + \lambda^2\right).$$

Then, the system can be solved using $2^{\lambda} \cdot (m-1)(\lambda^{\omega} + \lambda^2)$ bit operations where 2^{λ} is the number of possible assignments for the input x. For compairison, exhaustive search takes 2^{λ} evaluations of AIM2. Hence, the above is faster than exhausitive search if solving one such liniear system to verify the guess is faster than one execution of AIM2. Assuming $\omega = 2.37$, the complexity of solving different instances of AIM2, is summarized in Table 8.6.

Primitive	m	Linear system Time(ns)	AIM2 Time(μs)	Speed-up factor
AIM2-I	2	201	330	$2^{10.7}$
AIM2-III	2	206	347	$2^{10.71}$
AIM3-III	3	412	510	$2^{10.27}$

Table 8.7: Time of solving the linear system compared to the time of AIM2 algorithm based on the optimizations of [KHSL23]. The times are computed over 7 separate runs of the algorithms, with each run consisting of 10^7 executions.

8.3.2 Experimental Verification

We compare the complexity of solving the linear system described in Equation. 8.7 with complexity of AIM2 circuit using SageMath 10.5 on Apple M4 Pro chip. The result for each instance is described in Table 8.7.

8.4 Provable security claims

The AIMer specification provides two provable security results. On the one hand, the authors present an analysis of the AIM design assuming that the S-boxes are random permutations. On the other hand, the authors present an analysis of the final signature scheme.

8.4.1 Security proof of AIM

The authors provide a security proof of one-wayness of the AIM construction assuming that the S-boxes are (public) random permutations. This is a common approach in arguing security of hash function designs.

8.4.2 Security proof of signature

AIMer can be described as using AIM within a specific MPCitH based identification scheme called BN++ [KZ22] which generically describes how to turn the arithmetic circuit of a one-way function into an efficient identification scheme. The security proof of BN++ applies (and the proof in the AIMer paper is a direct copy of the BN++ security proof with the one-way function being spelled out as AIM).

8.4.3 Impact of Round 2 changes on proofs

The authors updated parts of the writing with regards to proofs and exact parameters. However, the changes are relatively minor. At some points, only clarifications were added. As mentioned before, the proofs seem correct – our review did at least not reveal any major flaws – up to minor potential imprecisions that do not impact the correctness of the final result.

Previous issues that were handled.

In the last report we pointed out that the model used for the proof of AIM is flawed. The authors now clarified that the adversary only gets access to the function or the permutations after they picked an image. The reason is that they want to prove the notion of everywhere preimage resistant (ePRE). However, this is a questionable model. The ePRE notion is a notion for keyed hash functions where the adversary only receives the randomly chosen key after they picked an image. The authors try to apply it to an unkeyed primitive. While they can make it work in an idealized setting, it does not make sense for any real-world instantiation of the permutations. Obviously, there is nothing that can keep the adversary from evaluating the permutations before they commit to an image. However, this clearly implies security against standard preimage attacks in which the image is chosen by the game which is enough for the security of AIMer.
Another issue was the relevance of the proof and the impact of using a final permutation with feed forward. The authors clarify that they can prove a better bound than what we get from the sum of permutations which is a meaningful motivation.
Finally, we remarked that the proof only considers classical attackers as it is given in the ROM. This did not change and the authors leave it as future work.

Room for improvement

In general, the proof would benefit a lot from adding intuition. This would ease the work of reviewers trying to follow the proof to verify correctness. Also, the proof seems unnecessarily non-tight in several places which limits its relevance for practical parameters / requires bigger parameters than necessary for several functions.
Following recent works [AHJ+23, HJMN24] it seems clear that one can obtain a close to optimal (as tight as possible) proof for AIMer in the QROM. This only requires to determine HVZK and certain soundness properties of the underlying identification scheme of AIMer (see [HJMN24] for details).

Moreover, there were several recent developments that could significantly improve the size and speed of AIMer. Most importantly, the authors of [BBM+24] propose an optimization of GGM trees (or batch all-but-one vector commitments) as used by AIMer that can have a significant impact on the signature size without reducing speed. An older improvement is the hypercube technique from [AGH+23] which was generalized in [HJMN24] which also allows to reduce the signature size by limiting the number of delta values that have to be sent when boosting soundness.

Parameters

Another complaint in the last report was about the authors not detailing their parameter choices for the used hash functions. Especially, output lengths for the different security levels were not provided. This is fixed in the new report. However, the new analysis in Section 6 of the submission document actually points out flaws. Specifically, the authors claim to achieve security level L1 (L3, L5) using hash functions with 128 (192, 256) bit outputs, while requiring collision resistance of these hash functions. However, they actually achieve security level L2 (L4, and security of a generic 512-bit hash function) by definition. The AIMer argument for L1, L3, and L5 implicitly is the discussion if L1 = L2, L3 = L4, and so on. While arguments can be made in this direction, it seems wiser to stick to levels L2, L4, and what we would call L6 as it is more precise. Moreover, the soundness analysis in Section 6.2 entirely ignores quantum attacks.

8.5 Round-2 C software

The sizes of keys etc. in the round-2 software match the documentation.
We posted a series of AIMer patches (kpqc-bulletin email from 2 May 2024 18:51:36 +0200) to resolve all issues detected by a TIMECOP run on `aimer128f` on an Intel Skylake CPU:

- Used bit operations to eliminate a variable index in `nodes[index ^ 1]`. An initial analysis indicated that this was public data, meaning that there is no security issue; this was confirmed by the AIMer team.
- Rewrote `poly64_mul` to eliminate variable indices. This is presumably a security issue for the original reference and optimized code. The `avx2` code uses PCLMULQDQ and is not affected.

- Replaced `htole64` and `le64toh` with byte computations. The original functions take variable time on some platforms; also, casting byte pointers to `uint64_t` pointers can lead to crashes on some platforms.
- Replaced `_load_` and `_store_` with `_loadu_` and `_storeu_` in the `avx2` code. The `_load_` and `_store_` functions can crash when inputs are unaligned.
- Rewrote `SQR_LOW` and `SQR_HIGH` to eliminate variable indices. This is presumably a security issue for the original reference and optimized code.
- Eliminated a variable branch on `GF_getbit(a_, i)`. It was not immediately clear whether there was a security issue here.
- Eliminated variable indexing in `commits[rep][i_bar]`. An initial analysis indicated that this was public data, meaning that there is no security issue; this was confirmed by the AIMer team.
- Eliminated variable indexing in `alpha_v_shares[rep][0][i_bar]` and `alpha_v_shares[rep][i_bar][0]`. An initial analysis indicated that this was public data, meaning that there is no security issue; this was confirmed by the AIMer team.
- Cleared some `avx2` buffers that TIMECOP had identified as having uninitialized data.

Some of these patches had a noticeable effect on performance. The AIMer team said it would modify the software, so we did not submit the software for the 25 June 2024 release of SUPERCOP. The AIMer team released new software on 26 June 2024, and we submitted AIMer for the 16 July 2024 release of SUPERCOP.
SUPERCOP's median cycle counts for `aimer128f` on Intel Haswell are 49339 for key generation, 1453521 for signing, and 1426865 for verification, where the documentation says 99847, 1483624, and 1442334 respectively. Other sizes similarly show that SUPERCOP matches the documentation for signing times and verification times.
For AMD Zen 2, `https://github.com/kpqc-cryptocraft/KpqClean_ver2` lists `aimer128f` as 109474 cycles for key generation, 988392 cycles for signing, and 990973 cycles for verification. Presumably these results were reduced by overclocking. We re-ran the KpqClean software on an AMD Zen 2 with overclocking disabled; it reported 61878, 1421772, and 1410609 cycles respectively. SUPERCOP reports slightly smaller AMD Zen 2 cycle counts: 57795, 1388218, and 1378545 respectively.

Tables 8.8, 8.9, and 8.10 show the number of instructions used for key generation, signing, and verification respectively inside AVX2 software for `aimer128f`. In each case, more than 50% of instructions are used by Keccak computations.

62125	62.35%	KeccakP-1600-AVX2.s:__KeccakF1600’2
8494	8.53%	KeccakP-1600-AVX2.s:KeccakP1600_ExtractBytes
8000	8.03%	aim2.c:samsungsds_aimer128f_avx2_generate_matrices_L_and_U
6800	6.83%	fips202.c:fips202avx_shake128_squeeze
2700	2.71%	KeccakP-1600-AVX2.s:__KeccakF1600
1792	1.80%	emmintrin.h:samsungsds_aimer128f_avx2_GF_sqr
1285	1.29%	hash.c:samsungsds_aimer128f_avx2_hash_squeeze
1285	1.29%	field.c:samsungsds_aimer128f_avx2_GF_from_bytes
1280	1.28%	wmmintrin.h:samsungsds_aimer128f_avx2_GF_sqr
1160	1.16%	avx2intrin.h:samsungsds_aimer128f_avx2_GF_transposed_matmul
693	0.70%	aim2.c:samsungsds_aimer128f_avx2_GF_exp_invmer_e_2
678	0.68%	aim2.c:samsungsds_aimer128f_avx2_GF_exp_invmer_e_1
637	0.64%	emmintrin.h:samsungsds_aimer128f_avx2_GF_mul
625	0.63%	KeccakP-1600-AVX2.s:KeccakP1600_Permute_24rounds
584	0.59%	field.c:samsungsds_aimer128f_avx2_GF_transposed_matmul
343	0.34%	wmmintrin.h:samsungsds_aimer128f_avx2_GF_mul
272	0.27%	avxintrin.h:samsungsds_aimer128f_avx2_GF_transposed_matmul
256	0.26%	field.c:samsungsds_aimer128f_avx2_GF_sqr
131	0.13%	chacha.S:crypto_stream_chacha20_moon_avx2_64_constbranchindex_blocks_avx2
102	0.10%	knownrandombytes.c:randombytes
97	0.10%	aim2.c:samsungsds_aimer128f_avx2_aim2
49	0.05%	field.c:samsungsds_aimer128f_avx2_GF_mul
35	0.04%	field.c:samsungsds_aimer128f_avx2_GF_add
209	0.21%	others
99633	100%	

Table 8.8: Instructions used for AVX2 software for `aimer128f` key generation. Each table row tallies the number of instructions used directly by one function. If function F calls function G then instructions inside G are tallied for G, not for F.

1289025	37.47%	avx2intrin.h:KeccakP1600times4_PermuteAll_24rounds
643615	18.71%	KeccakP-1600-AVX2.s:__KeccakF1600'2
573012	16.66%	avx2intrin.h:samsungsds_aimer128f_avx2_GF_transposed_matmul_add_N
118380	3.44%	KeccakP-1600-times4-SIMD256.c:KeccakP1600times4_AddBytes
105768	3.07%	KeccakP-1600-times4-SIMD256.c:KeccakP1600times4_ExtractBytes
104808	3.05%	field.c:samsungsds_aimer128f_avx2_GF_transposed_matmul_add_N
74240	2.16%	avx2intrin.h:samsungsds_aimer128f_avx2_GF_transposed_matmul
53199	1.55%	KeccakP-1600-AVX2.s:KeccakP1600_AddBytes
37926	1.10%	hash.c:samsungsds_aimer128f_avx2_hash_squeeze_x4
37376	1.09%	field.c:samsungsds_aimer128f_avx2_GF_transposed_matmul
33491	0.97%	sign.c:samsungsds_aimer128f_avx2_run_phase_1
32274	0.94%	emmintrin.h:samsungsds_aimer128f_avx2_POLY_mul_add_N
28670	0.83%	memmove-vec-unaligned-erms.S:__memcpy_avx_unaligned_erms
27972	0.81%	KeccakP-1600-AVX2.s:__KeccakF1600
25129	0.73%	hash.c:samsungsds_aimer128f_avx2_hash_update_x4
23807	0.69%	field.c:samsungsds_aimer128f_avx2_GF_add
17408	0.51%	avxintrin.h:samsungsds_aimer128f_avx2_GF_transposed_matmul
13794	0.40%	emmintrin.h:samsungsds_aimer128f_avx2_GF_mul_add_N
13365	0.39%	field.c:samsungsds_aimer128f_avx2_GF_copy
13200	0.38%	sign.c:samsungsds_aimer128f_avx2_commit_and_expand_tape_x4
12870	0.37%	emmintrin.h:samsungsds_aimer128f_avx2_GF_mul_N
12672	0.37%	wmmintrin.h:samsungsds_aimer128f_avx2_POLY_mul_add_N
12234	0.36%	KeccakP-1600-AVX2.s:KeccakP1600_ExtractBytes
135527	3.94%	others
3439762	100%	

Table 8.9: Instructions used for AVX2 software for `aimer128f` signing. Each table row tallies the number of instructions used directly by one function. If function F calls function G then instructions inside G are tallied for G, not for F.

1167705	34.69%	avx2intrin.h:KeccakP1600times4_PermuteAll_24rounds
708225	21.04%	KeccakP-1600-AVX2.s:__KeccakF1600'2
573012	17.02%	avx2intrin.h:samsungsds_aimer128f_avx2_GF_transposed_matmul_add_N
124620	3.70%	KeccakP-1600-times4-SIMD256.c:KeccakP1600times4_AddBytes
104808	3.11%	field.c:samsungsds_aimer128f_avx2_GF_transposed_matmul_add_N
100584	2.99%	KeccakP-1600-times4-SIMD256.c:KeccakP1600times4_ExtractBytes
74240	2.21%	avx2intrin.h:samsungsds_aimer128f_avx2_GF_transposed_matmul
54714	1.63%	KeccakP-1600-AVX2.s:KeccakP1600_AddBytes
37376	1.11%	field.c:samsungsds_aimer128f_avx2_GF_transposed_matmul
35574	1.06%	hash.c:samsungsds_aimer128f_avx2_hash_squeeze_x4
32274	0.96%	emmintrin.h:samsungsds_aimer128f_avx2_POLY_mul_add_N
30780	0.91%	KeccakP-1600-AVX2.s:__KeccakF1600
29105	0.86%	hash.c:samsungsds_aimer128f_avx2_hash_update_x4
27971	0.83%	sign.c:samsungsds_aimer128f_avx2_crypto_sign_verify
25619	0.76%	memmove-vec-unaligned-erms.S:__memcpy_avx_unaligned_erms
17408	0.52%	avxintrin.h:samsungsds_aimer128f_avx2_GF_transposed_matmul
13794	0.41%	emmintrin.h:samsungsds_aimer128f_avx2_GF_mul_add_N
13365	0.40%	field.c:samsungsds_aimer128f_avx2_GF_copy
13200	0.39%	sign.c:samsungsds_aimer128f_avx2_commit_and_expand_tape_x4
12870	0.38%	emmintrin.h:samsungsds_aimer128f_avx2_GF_mul_N
12722	0.38%	KeccakP-1600-AVX2.s:KeccakP1600_ExtractBytes
12672	0.38%	wmmintrin.h:samsungsds_aimer128f_avx2_POLY_mul_add_N
11088	0.33%	emmintrin.h:samsungsds_aimer128f_avx2_GF_sqr_N
132393	3.93%	others
3366119	100%	

Table 8.10: Instructions used for AVX2 software for `aimer128f` verification. Each table row tallies the number of instructions used directly by one function. If function F calls function G then instructions inside G are tallied for G, not for F.

Chapter 9

HAETAE: Shorter Lattice-Based Fiat-Shamir Signatures

HAETAE [CCD+22] (Hyperball bimodAl modulE rejecTion signAture schemE) is a module lattice-based signature scheme based on the Fiat-Shamir with Aborts paradigm [Lyu09, Lyu12]. In that sense, it resembles NIST standard Dilithium [LDK+22]. The main difference is that the design of HAETAE is aimed at improving the sizes of keys and signatures – the proposal claims 29–39% shorter signatures when comparing with Dilithium – and 20–25% smaller public keys. The main changes are using a bimodal distribution as in BLISS [DDLL13] and a new sampler using hyperballs, where Dilithium uses uniform distribution and BLISS used discrete Gaussians.
In this evaluation we focus on the 2nd-round version of HAETAE, v2.1, but refer back to the previous three versions where appropriate.

9.1 System description

We give a general description of Dilithium-like signatures later, in Chapter 11. These systems use rejection sampling on the signatures to ensure that their distribution is independent of the secret key.
Concretely, HAETAE works in $R = \mathbf{Z}[x]/(x^n + 1)$ for $n = 2^r$ (here $n = 256$), and also in $R_q = (\mathbf{Z}/q)[x]/(x^n+1)$ for q a prime with $2n|(q-1)$. All HAETAE parameters use $q = 64513$. Unlike NCC-Sign but like Dilithium, HAETAE uses module lattices in which the lattice elements are vectors of elements from R. The public key is an $k \times (\ell + k)$ matrix over R. The maximum-security version has $k = 4, \ell = 7$. This matrix can be interpreted as a $kn \times (\ell + k)n$

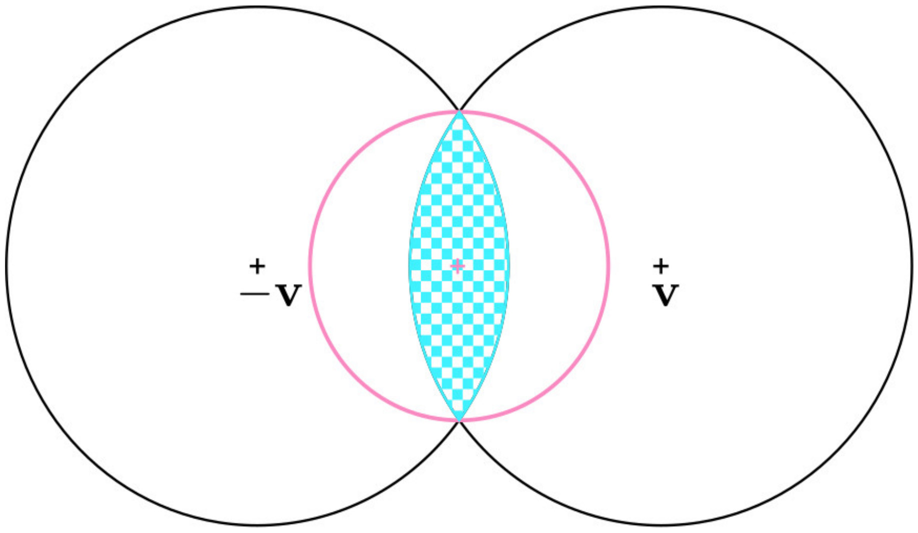


Figure 9.1: The HAETAE eyes, picture taken from [CCD+22].

lattice system and the generic attacks use this interpretation. So far no better attacks are known for module structure than generic lattice attacks.
The benefits of a bimodal distribution are that the signatures are distributed over two different centers, one linked to the secret $\mathbf{v}$ and one to $-\mathbf{v}$. Rejection sampling samples from the overlay of these two distributions and can stay in a narrower region around the middle, meaning that fewer rejections are encountered and that parameters can be chosen smaller for the same security level as the narrower size makes forgeries less likely.
In HAETAE the authors choose a hyperball distribution around the secrets and do rejection sampling to a hyperball around 0. In Figure 9.1, taken from the HAETAE v0.9 submission [CCD+22], the secrets are the pupils of the eyes of the Haetae; the originally sampled signatures are the blue circles and the signatures that pass rejection are in the pink circle. The checkered region would be sampled twice as frequently and is thus rejected with 50% probability.
Table 1 in the submission shows the expected number of repetitions until a signature passes rejection sampling, which is 6, 5, and 6 for the three versions of HAETAE.
A defining feature of HAETAE beyond the bimodal distribution is that it used a hyperball sampler. Earlier versions of HAETAE had moved from advertising avoiding Gaussian sampling to using discrete Gaussian sampling in the process of hyperball sampling to approximate sampling from a continuous Gaussian. While the submission document to the 2nd round is consis-

tent in its messaging, we are still missing a comparison to a design that uses Gaussians directly in place of hyperballs, basically a module-lattice version of BLISS. The authors have added a comparison of their Gaussian sampler with the GALACTICS sampler from [BBE$^+$19].

9.2 Security

All changes from BLISS and Dilithium are covered in the security proofs (for issues see the next section). For the 2nd-round submission we were able to run the script estimating the security of the chosen parameter sets as provided in the implementation package. It would have been nicer to have some documentation on how to use it but we managed to run it. However, as pointed out in Section 2.1, our output did not exactly match the values stated in Table 1 of the submission.
Furthermore, as found by Lee, Ryu, and Lee and announced in email to the KpqC bulletin dated 14 May 2024, 19::08:55 +0900, the parameters for HAETAE-120 do not satisfy the specification requirements for the paHVZK property requiring $B^2 \geq B'^2 + \gamma^2\tau$. The submitters acknowledged the issue and promised to provide updated parameters Another issue regarding the parameters was pointed out by Kim in email to KpqC bulletin on 28 Feb 2024 23:33:45 -0800 and concerns the length of μ, which is *the* hash involving the message and thus must be defended against collision attacks. The size of μ is not included in the parameter set but Kim noticed in the general specification and the implementation that it was limited to 256 bits while it should be longer for the higher-security parameters. The submitters have acknowledged this issue and promised an update. The hash μ is computed locally and not transmitted, hence its size is not relevant to the signature size; as the submitters point out, the implementation even derives it as a truncation of a larger value, so that the change will not have any noticeable performance implications, as the only place where μ is used is as input to another hash function.
Both of these issues are easy to fix but show the limitations of the script, in that it does not check the boundary conditions, and of the parameter table, in that it does not cover all parameters an in particular does not state the lengths of symmetric functions.
At this point we have not yet audited the script for correctness. For the other lattice-based schemes we used the lattice estimator to double-check the assertions by the authors. The lattice estimator is widely used and has received review; it is also relevant to say that this external review has found errors in previous versions. In turn, the software submitted with the imple-

mentation package has not received review and the version submitted to the 1st round of KpqC did not even work (see our report on the first round), hence, we have less confidence in the estimates. It is puzzling that our run of the same software did not reproduce the same numbers, further lowering our confidence in the results.

9.3 Implementation considerations

As stated above, v2.1 of HAETAE is consistent in avoiding floating-point arithmetic and using Gaussian sampling in order to define their hyperball sampler. This version also includes fixed rANS encoding, with smaller tables than earlier versions, after this was targeted in a bit-flipping attack by the HuFu Team, see email to NIST PQC Forum on 2 Sep 2023 22:17:28 -0700.
We noticed during round 1 that HAETAE had KAT mismatches. Several implementation issues were pointed out in a NIST PQC Forum post by Markku Saarinen. Section 9.5 describes our evaluation of the current software.

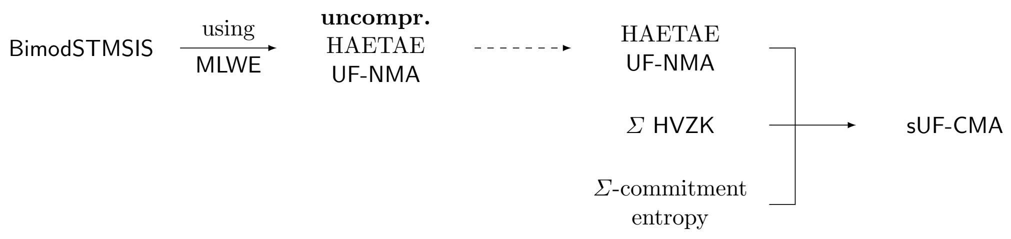
9.4 Provable security claims

On a high level, HAETAE shares with Dilithium that it first builds a lattice-based identification scheme Σ, and then turns Σ into a signature scheme using the Fiat-Shamir paradigm. Consequently, the security reasoning for HAETAE is similar to the security reasoning for Dilithium, with appropriately adapted assumptions.
The core idea of the Fiat-Shamir paradigm is as follows: signatures consist of a prover commitment, together with a prover response. To tie the prover response to the to-be-signed-message, it is built by using as its challenge the hash value of the message and the prover commitment. The Fiat-Shamir with aborts paradigm additionally introduces rejection sampling. This is done to render the distribution of signatures sufficiently independent of sensitive information (the secret key), so that observing exchanged signatures will not help an attacker with forging a signature.
In the following we first present our observations from the first-round report and then provide an update of how the submitters have handled these issues.

9.4.1 Security proof of HAETAE in round 1

Rejection sampling slightly complicates security proofs, in particular when concerning quantum attackers: two recent papers [BBD+23, DFPS23]

pointed out a flaw in the security proof of Dilithium that was directly tied to how the proof addressed rejection sampling. HAETAE v1.0 took this into account by adapting the proof to the fix provided in [DFPS23].
To argue Strong Unforgeability under Chosen Message Attacks (sUF-CMA) of (deterministic) HAETAE, the submission follows the approach of Dilithium: it goes through an "implication chain" that relates computational hardness of (appropriately adapted) problems to security properties. We summarize the chain in the figure below.



Gap in UF-NMA reasoning. The specification claims that hardness of the Bimodal Self-Target MSIS problem (BimodSTMSIS) directly translates into the fact that HAETAE satisfies the intermediate security notion UF-NMA, which is then used to argue the aimed-at security property (sUF-CMA). The definitions of BimodSTMSIS and UF-NMA, however, are only equivalent for the **uncompressed** version of HAETAE. The specification does not address whether/how compression affects UF-NMA security, and it is not obvious that compression does not decrease security.
Gap in HVZK reasoning. The specification aims to argue HVZK without presenting the underlying identification scheme Σ, thus forcing the reader to essentially re-do the proof. The proof sketch for HVZK also seems to introduce a LWR-like assumption that differs from more well-studied variants in the distribution of matrix and secret: it is assumed that $w = A\lfloor y \rceil$ is indistinguishable from a uniform element (both modulo q), where A is a public key (a randomly chosen compressed matrix). The specification does not make explicit how y is defined in this assumption. The likeliest interpretation is that y is computed as in the deterministic signing algorithm (Figure 7 in the specification), i.e., by using a – not further specified – expansion algorithm expandYbb.
Reading through the implementation package for v0.9, expandYbb is mentioned only in a comment but the steps match what is described in the main part of the specification, except that b is not made dependent on the seed but sampled uniformly random, so this should fail for recomputing the KATs. In the specification the Gaussian sampler is assumed to be continuous. Looking

at the code for `sampler_gaussian`, this uses a big table and is sure not safe against cache timing attacks (nor did it claim to be).
In v1.0 the code changed from calling `polydblveclk_uniform_hyperball` to calling a new function `polyfixveclk_sample_hyperball` using fixed-point arithmetic and the discrete Gaussian sampler. The latter is implemented with a very small CDT which might be small enough to fit in cache and to avoid cache-timing attacks. In general, there are far fewer branches in the code, but there might still be timing dependencies on the secret.
Asymptotic reasoning not necessarily applicable to parameter choices. The specification does not specify how closely the security notions are related to one another as it is missing concrete security bounds that quantify the relation. This leaves the reader with asymptotic reasoning, meaning it is only shown that the implication chain will begin to be satisfied at the point where appropriately large parameters are chosen. This makes it hard to verify security for the concrete parameter choices made in the specification. Depending on how close (or distant) the relations are, the proof might not apply.

9.4.2 Security proof of HAETAE in round 2

The update addresses all remarks on provable security made in Subsection 9.4.1, resulting in the (now-closed) 'implication chain' below:

$$\begin{tikzcd}
\mathsf{BimodSTMSIS} \arrow[r, "\text{using}", "\mathsf{MLWE}"'] & \begin{matrix}\text{HAETAE}\\ \mathsf{UF\text{-}NMA}\end{matrix} \arrow[dr] & \\
 & \Sigma\ \mathsf{HVZK} \arrow[r] & \mathsf{sUF\text{-}CMA} \\
 & \begin{matrix}\Sigma\text{-commitment}\\ \text{entropy}\end{matrix} \arrow[ur] &
\end{tikzcd}$$

Closed gap in UF-NMA reasoning. The update closes a gap in the security proof: the specification treated the hardness of the Bimodal Self-Target MSIS problem (BimodSTMSIS) as immediately related to the necessary intermediate security goal, UF-NMA of HAETAE. As pointed out in round 1, this relation only applies to the *uncompressed* version of HAETAE. The update adequately addresses this by giving a dedicated and sound-looking security reduction that captures compression.
Closed gap in HVZK reasoning. In the previous round, the specification assumed the necessary HVZK property without explicitly stating the concrete identification scheme Σ to which it assigns this property, which created

some overhead when assessing the overall security reasoning. The update adequately addresses the remark by a) specifying the identification scheme in question, and b) proving that it achieves HVZK, assuming the parameters are adequately chosen.
Now-concrete bounds allow reasoning about parameter choices. The specification now gives concrete security bounds that quantify how closely the security notions are related to one another. This allows to reason about the level of security that will be achieved for concrete parameter choices.

Interpretation of provable security results

The overall security reasoning already looked sound in round 1, up to the now-addressed missing details. If a vulnerability were to be found, it would likely stem from

- a cryptanalytical break of one of the underlying computational problems (MSIS, MLWE, BimodSTMSIS), or
- parameter choices that do not match the intended level of security.

9.5 Round-2 C software

The private-key sizes in the round-2 software do not match the documentation. The difference is 32 bytes.
To handle various TIMECOP alerts, we added `crypto_declassify` at various positions in `rej_uniform`, `crypto_sign_keypair`, `sample_gauss`, `polyfixveclk_sample_hyperball`, `poly_challenge`, `rej_eta`, and `crypto_sign_signature`. Most of these were for declassifying what appear to be rejection-sampling conditions. The exceptions were

- two variables `hb_z1` and `h` inside the `crypto_sign_signature` function (these seem to be public variables that are then encoded in a variable-time way) and
- a variable `b` in `poly_challenge` (this also seems to be public).

There was one further TIMECOP warning, namely for the following code in `sample_gauss_sigma76`:

```
const uint64_t *rand_gauss16_ptr = (uint64_t *)rand,
               *rand_rej_ptr = (uint64_t *)(&rand[2]);
const uint64_t rand_gauss16 = (*rand_gauss16_ptr) & ((1ULL << 16) - 1);
const uint64_t rand_rej = (*rand_rej_ptr) & ((1ULL << 48) - 1);
```

This is not a portable way to read 16-bit and 48-bit pieces from bytes: some CPUs prohibit unaligned reads, and big-endian CPUs will give the wrong results. We changed this to portable code.
Preliminary analysis indicates that none of the issues identified by TIMECOP for HAETAE are security issues. However, it is important to note that every use of `crypto_declassify` requires auditors to check whether the declassified variable is in fact safe to make public. The submitters are encouraged to check this for HAETAE.
Another issue, which presumably allows timing attacks on some platforms, is the division by `LN` in the following code:

```
int32_t fix_round(int32_t num) {
    num += (num >> 31) & (-LN + 1);
    num += LN / 2;
    return num / LN;
}
```

This code is called on secret inputs and can end up using division instructions (see, e.g., https://godbolt.org/z/rEEcn89x6 saying “idiv”). We changed this to

```
int32_t fix_round(int32_t num) {
  return (num + LNHALF) >> LNBITS;
}
```

with appropriate definitions of `LNHALF` and `LNBITS`.
We submitted this software for inclusion in the 25 June 2024 release of SUPERCOP. The Skylake results from SUPERCOP are on the same scale as the Core i7-10700k (Comet Lake) results reported in the HAETAE documentation.
The HAETAE team distributed new software on 17 July 2024 (not compatible with previous HAETAE software). We submitted a corresponding HAETAE update for the 8 August 2024 release of SUPERCOP.
The quartiles reported by SUPERCOP show that HAETAE key generation and signing have large variance in run times. For example, on Intel Comet Lake, $(\text{quartile}, \text{median}, \text{quartile})$ for `haetae2` are $(269003, 554861, 981331)$ for key generation, $(660171, 2325056, 3162129)$ for signing, and $(118609, 119061, 119652)$ for verification. The variations also make precise comparisons difficult. The February 2024 HAETAE documentation (before the HAETAE team's July 2024 announcement of a key-generation speedup) reported Comet Lake medians of 882350 cycles for key generation, 1323118 cycles for signing, and 115638 cycles for verification.

KpqClean reports 834651, 4946575, 67068 for an Intel Coffee Lake (which is similar to Comet Lake). We re-ran KpqClean on a Comet Lake with overclocking disabled. One run of KpqClean reported 1392489 cycles for key generation (presumably for HAETAE code before July 2024), 5044873 cycles for signing, and 123485 cycles for verification. A second run of KpqClean reported 1389823 cycles, 1216527 cycles, and 123730 cycles.
Checking the source code shows that KpqClean tries 10000 iterations of signing the same message with the same key, so new randomness appears only when KpqClean is run again. This means that a HAETAE signing result reported by KpqClean can be far above average or far below average.
Tables 9.1, 9.2, and 9.3 show the (average) number of instructions used for key generation, signing, and verification respectively inside AVX2 software for `haetae2`. Investigation of the top subroutines has identified some speedup possibilities to explore: e.g., merging FFT layers inside the `fft` function would eliminate many load/store instructions, and replacing some shift instructions with permutation instructions inside that function would make better use of "port 5" on Intel CPUs.

730964	32.46%	`fft.c:fft`
322232	14.31%	`sampler.c:cryptolab_haetae2_rej_eta`
164332	7.30%	`avx2intrin.h:fft`
125511	5.57%	`fft.c:fft_init_and_bitrev`
105611	4.69%	`avxintrin.h:fft`
78106	3.47%	`sampler.c:cryptolab_haetae2_rej_uniform`
75101	3.34%	`reduce.c:cryptolab_haetae2_freeze`
68588	3.05%	`ntt.S:cryptolab_haetae2_poly_ntt`
60081	2.67%	`decompose.c:cryptolab_haetae2_decompose_vk`
54194	2.41%	`crypto_int32.h:cryptolab_haetae2_polyvecmk_sqsing_value`
52633	2.34%	`fips202.c:KeccakF1600_StatePermute`
44317	1.97%	`invntt.S:cryptolab_haetae2_poly_invntt_tomont`
40249	1.79%	`polyvec.c:cryptolab_haetae2_polyveck_decompose_vk`
31918	1.42%	`pointwise.S:cryptolab_haetae2_poly_pointwise_montgomery`
30216	1.34%	`poly.c:cryptolab_haetae2_poly_freeze`
29727	1.32%	`avx2intrin.h:complex_fp_sqabs_add`
21347	0.95%	`polyvec.c:cryptolab_haetae2_polyvecmk_sqsing_value`
13391	0.59%	`crypto_uint32.h:cryptolab_haetae2_rej_eta`
12508	0.56%	`crypto_uint32.h:cryptolab_haetae2_rej_uniform`
9525	0.42%	`fips202.c:haetae_fips202_shake256_absorb`
7901	0.35%	`poly.c:cryptolab_haetae2_poly_add`
7823	0.35%	`fft.c:complex_fp_sqabs_add`
5809	0.26%	`sign.c:crypto_declassify`
159590	7.09%	others
2251675	100%	

Table 9.1: Instructions used for AVX2 software for `haetae2` key generation. Each table row tallies the number of instructions used directly by one function. If function F calls function G then instructions inside G are tallied for G, not for F.

820080	24.79%	`avx2intrin.h:cryptolab_haetae2_sample_gauss_N_4x`
161139	4.87%	`fips202.c:KeccakF1600_StatePermute`
154680	4.68%	`samplerx4.c:cryptolab_haetae2_sample_gauss_N_4x`
77937	2.36%	`sampler.c:cryptolab_haetae2_rej_uniform`
63448	1.92%	`invntt.S:cryptolab_haetae2_poly_invntt_tomont`
63126	1.91%	`ntt.S:cryptolab_haetae2_poly_ntt`
49397	1.49%	`fips202.c:haetae_fips202_shake256_absorb`
42352	1.28%	`avxintrin.h:cryptolab_haetae2_sample_gauss_N_4x`
41712	1.26%	`avx2intrin.h:_mul_rnd13`
22296	0.67%	`crypto_int64.h:_mul_rnd13`
22016	0.67%	`emmintrin.h:cryptolab_haetae2_sample_gauss_N_4x`
19968	0.60%	`polyfix.c:_mul_rnd13`
19002	0.57%	`rans_byte.h:cryptolab_haetae2_encode_hb_z1`
16128	0.49%	`avx2intrin.h:cryptolab_haetae2_polyfixveclk_sqnorm2`
15656	0.47%	`poly.c:cryptolab_haetae2_poly_pack_highbits`
12476	0.38%	`crypto_uint32.h:cryptolab_haetae2_rej_uniform`
12304	0.37%	`poly.c:cryptolab_haetae2_poly_pack_lsb`
11648	0.35%	`fips202x4.c:haetae_fips202x4_avx2_shake256x4_squeezeblocks_vec`
11296	0.34%	`pointwise.S:cryptolab_haetae2_polyvecl_pointwise_acc_montgomery`
10880	0.33%	`pointwise.S:cryptolab_haetae2_poly_pointwise_montgomery`
10262	0.31%	`encoding.c:cryptolab_haetae2_encode_hb_z1`
9524	0.29%	`rans_byte.h:cryptolab_haetae2_encode_h`
7368	0.22%	`polyfix.c:cryptolab_haetae2_polyfixveclk_sqnorm2`
1633763	49.38%	others
3308458	100%	

Table 9.2: Instructions used for AVX2 software for `haetae2` signing. Each table row tallies the number of instructions used directly by one function. If function F calls function G then instructions inside G are tallied for G, not for F.

77937	23.01%	`sampler.c:cryptolab_haetae2_rej_uniform`
63479	18.75%	`fips202.c:KeccakF1600_StatePermute`
22434	6.62%	`fips202.c:haetae_fips202_shake256_absorb`
14149	4.18%	`rans_byte.h:cryptolab_haetae2_decode_hb_z1`
14028	4.14%	`ntt.S:cryptolab_haetae2_poly_ntt`
12476	3.68%	`crypto_uint32.h:cryptolab_haetae2_rej_uniform`
9233	2.73%	`encoding.c:cryptolab_haetae2_decode_hb_z1`
6964	2.06%	`rans_byte.h:cryptolab_haetae2_decode_h`
5649	1.67%	`encoding.c:cryptolab_haetae2_decode_h`
5322	1.57%	`poly.c:cryptolab_haetae2_polyq_unpack`
4532	1.34%	`invntt.S:cryptolab_haetae2_poly_invntt_tomont`
3914	1.16%	`poly.c:cryptolab_haetae2_poly_pack_highbits`
3180	0.94%	`sign.c:crypto_declassify`
3076	0.91%	`poly.c:cryptolab_haetae2_poly_pack_lsb`
2824	0.83%	`pointwise.S:cryptolab_haetae2_polyvecl_pointwise_acc_montgomery`
1914	0.57%	`packing.c:cryptolab_haetae2_unpack_sig`
1740	0.51%	`sign.c:cryptolab_haetae2_verify`
1490	0.44%	`avx2intrin.h:cryptolab_haetae2_poly_reduce2q`
1280	0.38%	`crypto_int8.h:cryptolab_haetae2_unpack_sig`
1234	0.36%	`avx2intrin.h:cryptolab_haetae2_poly_freeze2q`
1188	0.35%	`poly.c:cryptolab_haetae2_poly_challenge`
1008	0.30%	`emmintrin.h:haetae_fips202x4_avx2_shake128x4_squeezeblocks`
1002	0.30%	`fips202x4.c:haetae_fips202x4_avx2_shake128x4_squeezeblocks`
78588	23.21%	others
338641	100%	

Table 9.3: Instructions used for AVX2 software for `haetae2` verification. Each table row tallies the number of instructions used directly by one function. If function F calls function G then instructions inside G are tallied for G, not for F.

Chapter 10

MQ-Sign: A New Post-Quantum Signature Scheme based on Multivariate Quadratic Equations: Shorter and Faster

The MQ-Sign digital signature scheme [SK24] is based on the trapdoor paradigm. In this setting, the secret key is composed of a central map

$$\mathcal{F} : (x_1, \ldots, x_n) \in \mathbb{F}_q^n \to \big(\mathcal{F}^{(1)}(x_1, \ldots, x_n), \ldots, \mathcal{F}^{(m)}(x_1, \ldots, x_n)\big) \in \mathbb{F}_q^m,$$

for which it is computationally easy to find a solution, aka a tuple $\mathbf{x} = (x_1, \ldots, x_n)$ such that $\mathcal{F}(\mathbf{x}) = 0$, and two bijective affine mappings $\mathcal{S} \in \mathrm{AGL}_n(q), \mathcal{T} \in \mathrm{AGL}_m(q)$. These mappings are used to hide the special structure of $\mathcal{F}$ that allows us to easily compute solutions. The public key is obtained by mapping $\mathcal{F}$ to another quadratic map $\mathcal{P}$, such that $\mathcal{P} = \mathcal{T} \circ \mathcal{F} \circ \mathcal{S}$. This results in a multivariate system that is hard to solve and indistinguishable from a randomly generated multivariate quadratic system unless the structure of $\mathcal{F}$ can be retrieved from $\mathcal{P}$.
Define a hash function to map to $\mathbb{F}_q^m$ and let $\mathbf{t} = H(M)$, then a signature on M is a preimage $\mathbf{x}$ of $\mathbf{t}$ under the public map. To verify the signature, compute $\mathbf{t}$ and accept if this matches $\mathcal{P}(\mathbf{x})$. To find such an $\mathbf{x}$, the signer uses the knowledge of the affine maps and the ability to compute preimages of the central map to compute $\mathbf{t}' = \mathcal{T}^{-1}(\mathbf{t}), \mathbf{x}' = \mathcal{F}^{-1}(\mathbf{t}')$, and $\mathbf{x} = \mathcal{S}^{-1}(\mathbf{x}')$, where $\mathcal{F}^{-1}(\mathbf{t}')$ means the computation of a (typically not unique) preimage of $\mathbf{t}'$. For a secure scheme, it should be hard to find $\mathbf{x}$ given any $\mathbf{t}$ and the public key.

10.1 Unbalanced Oil and Vinegar

One of the oldest trapdoor constructions is the Unbalanced Oil and Vinegar signature scheme, proposed by Kipnis, Patarin, and Goubin [KPG99] as a modification of the oil and vinegar scheme of Patarin [Pat97] that was broken by Kipnis and Shamir in 1998 [KS98].
In the oil and vinegar construction, the variables in the central map are divided in two distinct sets, called vinegar variables and oil variables. The vinegar variables are combined quadratically with all the variables, while the oil variables are only combined quadratically with the vinegar variables and not with other oil variables. This special structure serves as a trapdoor that allows to find preimages of the central map. Formally, the central map is defined as $\mathcal{F} : \mathbb{F}_q^n \to \mathbb{F}_q^m$, with polynomials

$$\mathcal{F}^{(k)}(x_1, \dots, x_n) = \sum_{i \in V, j \in V} \gamma_{ij}^{(k)} x_i x_j + \sum_{i \in V, j \in O} \gamma_{ij}^{(k)} x_i x_j + \sum_{i=1}^{n} \beta_i^{(k)} x_i + \alpha^{(k)} \quad (10.1)$$

where $n = v + m$, and $V = \{1, \dots, v\}$ and $O = \{v + 1, \dots, n\}$ denote the index sets of the vinegar and oil variables, respectively.
The affine mapping $\mathcal{T}$ is omitted, as it can be shown that if an oil and vinegar central map is used in the standard MQ construction, $\mathcal{T}$ does not add to the security of the scheme. Hence, the secret key consists of a linear transformation $\mathcal{S}$ and central map $\mathcal{F}$, while the public key is defined as $\mathcal{P} = \mathcal{F} \circ \mathcal{S}$. It is also common to use homogeneous polynomials in UOV, so the affine map becomes a linear map represented by the matrix $S \in \mathrm{GL}_n(q)$. As with any trapdoor-based multivariate signature scheme, to sign a message, we need to find a preimage of $\mathcal{F}$. This can be done by simply fixing the vinegar variables to some random values. The resulting system is a linear system of m equations in m variables, and thus has a solution with probability around $1 - 1/q$. If the obtained system does not have a solution, we repeat the procedure with different values for the vinegar variables, otherwise we apply the inverse affine transformation and obtain a signature.

10.2 MQ-Sign

UOV signature schemes are attractive because they have very small signatures and fast verification. On the downside, they have large public and secret keys. As a result, variations of the traditional UOV scheme are usually developed with the goal to reduce the size of the public key. These variations have additional structure that might compromise the security of the scheme. A notable example of such a scheme is Rainbow [DS05], which was a finalist in the NIST PQC standardization process as [DCP$^+$20], before Beullens showed that it does not meet the security requirements [Beu22]. It is a great challenge to develop UOV variations with additional structure that do not compromise the security of the scheme or where the trade-off results in smaller key sizes.

MQ-Sign is a UOV-based signature scheme, where the main focus is to reduce the size of the public and secret key compared to traditional UOV. In the first-round proposal of MQ-Sign, this was achieved by using sparse polynomials for the quadratic part of the central map. In the central map, we distinguish two main parts: the vinegar-vinegar part, which is comprised of monomials that contain two vinegar variables, and the vinegar-oil part, comprised of the monomials containing one vinegar variable and one oil variable. There were initially four variations of the scheme: MQ-Sign-SS, MQ-Sign-RS, MQ-Sign-SR, and MQ-Sign-RR, where the suffix specifies, for the two parts of the quadratic maps, whether they are defined with sparse or random polynomials. Three attacks on MQ-Sign were published during the first round of the competition, which eliminated all but the last variant of MQ-Sign, denoted MQ-Sign-RR. This is the most conservative variant as it is built on the UOV trapdoor without any additional structure. The first algebraic attack on MQ-Sign was proposed by Aulbach, Samardjiska, and Trimoska [AST24] and it exploits the sparseness of the vinegar-oil part of the secret key. The attack also relies on the fact that the map $\mathcal{S}$ is chosen to be given by a matrix of the following form

$$S = \begin{pmatrix} I_{v\times v} & S_1 \\ 0_{m\times v} & I_{m\times m} \end{pmatrix}. \tag{10.2}$$

This typically does not reduce the security of a UOV-based scheme because it was shown in [Pet13] that for any instance of a UOV secret key $(\mathcal{F}', S')$, there is an equivalent key $(\mathcal{F}, S)$ where S has the form as in (10.2). However, coupling this optimization technique with the specific structure of the central map in MQ-Sign yields many linear constraints that allow for a polynomial-time key recovery. The attack was fully implemented and it was reported to run in 0.6 seconds for the proposed parameters for the security level I, 2.3 seconds for the security level III, and 6.9 seconds for the security level V.

Following this, Ikematsu, Jo, and Yasuda proposed another algebraic attack that also targets the MQ-Sign-{S/R}S variants but is not dependent on S having the equivalent keys structure [IJY23]. These two attacks eliminated both variants where the vinegar-oil part of the secret key is sparse.

Another algebraic attack was proposed in [AST24] that targets specifically the variant in which only the vinegar-vinegar part of the secret key is sparse.

This attack is not practical, but shows that the security of MQ-Sign-SR does not meet the required security level, and this variant is also removed from the updated submission in round 2.

10.3 Design updates

MQ-Sign announces two updates for the second round. One major update is the design of the variant MQ-Sign-LR that has a specific structure only in the vinegar-vinegar part of the central map and the vinegar-oil part is generated randomly, same as in MQ-Sign-RR. Both variants use the equivalent keys optimization. The second update is the introduction of a binding technique.

10.3.1 A new variant MQ-Sign-LR

MQ-Sign-LR is the new non-conservative variant of MQ-Sign that has smaller secret keys while maintaining the same signature size as MQ-Sign-RR. This variant reportedly yields better performance for both key generation and signing. Note that since the reduction of the size of the secret key is in the vinegar-vinegar part, when the equivalent keys optimization is used, this yields a reduction in the public key size as well. This is because when S is of the form as in (10.2), the vinegar-vinegar part of the public key is equal to the vinegar-vinegar part of the secret key. This can be observed from the equation defining the computation of the public key

$$\begin{pmatrix} P_1^{(k)} & P_2^{(k)} \\ 0 & P_4^{(k)} \end{pmatrix} = \begin{pmatrix} I & 0 \\ S_1^\top & I \end{pmatrix} \begin{pmatrix} F_1^{(k)} & F_2^{(k)} \\ 0 & 0 \end{pmatrix} \begin{pmatrix} I & S_1 \\ 0 & I \end{pmatrix},$$

which in upper-triangular matrix form simplifies to

$$\begin{pmatrix} P_1^{(k)} & P_2^{(k)} \\ 0 & P_4^{(k)} \end{pmatrix} = \begin{pmatrix} F_1^{(k)} & (F_1^{(k)} + F_1^{(k)\top})S_1 + F_2^{(k)} \\ 0 & \mathsf{Upper}(S_1^\top F_1^{(k)} S_1 + S_1^\top F_2^{(k)}) \end{pmatrix}$$

adding the entry at (j,i) to the entry of (i,j) for $j > i$.

The main difference between the two MQ-Sign variants is in the structure of the vinegar-vinegar part of the central map. In MQ-Sign-LR, the vinegar-vinegar part of the central map is constructed as a product of a circulant matrix where the entries are the vinegar variables $(x_1, \ldots, x_v)$ and a vector whose entries are linear combinations of the vinegar variables. Specifically, the central map is defined as

$$\begin{pmatrix} x_1 & x_2 & ... & x_v \\ x_v & x_1 & \dots & x_{v-1} \\ \dots & \dots & \dots & \dots \\ x_{v-m+2} & x_{v-m+3} & \dots & x_{v-m+1} \end{pmatrix} \cdot \begin{pmatrix} L_1 \\ L_2 \\ \dots \\ L_v \end{pmatrix}, \qquad (10.3)$$

where $L_i = \sum_{j=1}^{v} \gamma_{ij} x_j$, for $i \in \{1, \dots, v\}$ and each row of the product matrix gives a polynomial in $\mathcal{F}$. As a result, the vinegar-vinegar part of the central map can be represented with v^2 field elements[1], instead of the $\frac{v^2 m}{2}$ field elements that are required in the MQ-Sign-RR variant.

10.3.2 A new binding technique

The binding technique that is introduced aims to prevent the following scenario. Let us assume that there are two equivalent public keys, so public keys $\mathcal{P}_A$ and $\mathcal{P}_B$ such that $\mathcal{P}_B = \mathcal{P}_A \circ S$. The corresponding secret keys are denoted $(\mathcal{F}_A, S_A)$ and $(\mathcal{F}_B, S_B)$. If $\sigma_A = (\mathbf{z}_A, r)$ (r denotes the salt used in signing) is a signature on a message M under the public key $\mathcal{P}_A$, then anyone who knows S can generate a valid signature $\sigma_B = (\mathbf{z}_B, r)$ on the same message M under the public key $\mathcal{P}_B$ by computing $\mathbf{z}_B = S^{-1}\mathbf{z}_A$.
The binding technique aims to prevent this attack by identifying a signature with a unique public key in the following manner. In the usual setting, the target value $\mathbf{t}$ for which the signer needs to find a preimage $\mathbf{z}$ is computed as a hash $\mathbf{t} = H(M||r)$. The binding technique consists in adding the hash of the public key of the signer to the input to the hash function, so $\mathbf{t} = H(M||r||H(\mathcal{P}))$. As a result, $\mathbf{z}_B = S^{-1}\mathbf{z}_A$ will not form a valid signature under the public key $\mathcal{P}_B$, as $\mathbf{z}_A$ was computed with $\mathcal{P}_A$ as input to the hash.

An attack under this assumption. Note that this binding technique protects against adversaries that only know the two equivalent public keys, but it does not prevent the owner of the public key $\mathcal{P}_B$ to produce a valid signature under the public key $\mathcal{P}_A$. Since the two public keys are equivalent, being able to invert any of the two maps allows us to invert both of them. Specifically, the owner of $\mathcal{P}_B$ can forge a signature under $\mathcal{P}_A$ by computing a preimage of $\mathbf{t} = H(M||r||H(\mathcal{P}_A))$ using their own secret key as $\mathbf{z}_B = S_B^{-1} \cdot \mathcal{F}_B^{-1}(\mathbf{t})$, and then computing $\mathbf{z}_A = S\mathbf{z}_B$ using their knowledge of the equivalence. We emphasize that this is not a security concern because the probability of having two equivalent public keys is negligibly small. We

[1]The submission counts $v(v+m)$ elements, which corresponds to the evaluation costs, but not to the number of stored elements.

verified that this is the case for the parameters of MQ-Sign. We first count the number of possible public keys. Each key is a set of m $n \times n$ matrices, but not all maps of this size constitute a UOV public key, since a UOV public key contains the hidden oil space. We can equivalently count the number of secret keys and this gives rise to an easier reasoning because the secret keys are of a specific form. Specifically, a central map is comprised of matrices

$$F^{(k)} = \begin{pmatrix} F_1^{(k)} & F_2^{(k)} \\ 0_{m\times v} & 0_{m\times m} \end{pmatrix},$$

where $F_1^{(k)}$ is an upper-triangular matrix of size $v \times v$ and $F_2^{(k)}$ is a rectangular matrix of size $v \times m$. Hence, for MQ-Sign-RR, we have $q^{mv(v-1)/2+vm^2}$ possible public keys. For MQ-Sign-LR, we have $q^{v^2+vm^2}$ public keys. Note that for this variant, the v^2 in the exponent does not have a factor m, due to the specific way the secret key is created. For the parameters of security level 1, this leads to $2^{1260288}$ possible public keys for MQ-Sign-LR and $2^{2159424}$ for MQ-Sign-RR. To get the probability that two public keys are equivalent, we divide the total number of possible maps between equivalent public keys by the number of public keys. These possible maps are simply invertible $n \times n$ matrices, of which there are $\prod_{i=0}^{n-1}(q^n - q^i)$. So we have 2^{111391} of these maps, which results in a probability of two public keys being equivalent of $2^{-1148897}$ for MQ-Sign-LR and $2^{-2048033}$ for MQ-Sign-RR. This shows that equivalent keys are not a security concern. Note that the negligible probability is a crucial argument, because checking whether two public keys are equivalent is an easy problem. Since MQ-Sign (and UOV in general) does not use a transformation on the output (denoted as $\mathcal{T}$ in the introduction), finding the equivalence between two quadratic maps is the isomorphism of polynomials with one secret (IP1S) problem, which can be solved with a polynomial-time algorithm [BFFP11].
The binding technique is used only in the variant MQ-Sign-RR. No rationale is provided for this choice, but the analysis in [SK24] shows that there is a non-negligible performance loss, as well as a slight increase of the public key, as a result of the addition of the binding technique. However, adding the binding technique has the advantage of protecting against attacks that exploit hash collisions.

10.4 A forgery attack on MQ-Sign-LR

We start the description of the attack by showing how an attacker can forge a signature in the case where $\mathbf{t} = H(M||r)$ is the zero vector. This attack exploits the structure of the vinegar-vinegar part and relies on the matrix S being of the form as in (10.2). The attack is probabilistic, however, the probability of success is high enough that we were able to perform it multiple times for all security levels. Forging a signature for $\mathbf{t} = \mathbf{0}$ consists in finding a preimage of $\mathbf{0}$ under the public map $\mathcal{P}$. We will write in block matrix representation since we are using the specific structure of the upper-left block of the public map. We need to find a vector $(\mathbf{x}_v, \mathbf{x}_m) \in \mathbb{F}_q^n$ such that

$$\begin{pmatrix}\mathbf{x}_v & \mathbf{x}_m\end{pmatrix}\begin{pmatrix}P_1^{(k)} & P_2^{(k)} \\ 0 & P_4^{(k)}\end{pmatrix}\begin{pmatrix}\mathbf{x}_v \\ \mathbf{x}_m\end{pmatrix} = \mathbf{x}_v P_1^{(k)} \mathbf{x}_v + \mathbf{x}_v P_2^{(k)} \mathbf{x}_m + \mathbf{x}_m P_4^{(k)} \mathbf{x}_m = 0$$

holds for every $k \in \{1, \ldots, m\}$. Note here that $P_1^{(k)}$ has the same shape as $F_1^{(k)}$ because of the equivalent keys form of S and this structure is used in the attack. Let us denote by $V(\mathbf{0})$ the variety (the set of solutions) of the ideal generated by $\mathcal{P}(\mathbf{x}) = 0$. Since we have m equations in n variables with $n > m$, we can afford to add another $v = n - m$ affine constraints and still expect, heuristically, to have a solution. We use only m (recall that $m < v$) of those to eliminate the nonstructured part of this system. Specifically, we assign all variables in $\mathbf{x}_m$ to zero. We are thus left with a homogenous system that has the structure depicted in (10.3). Looking closely at this structure, we see that the matrices representing these quadratic forms are equal up to cyclic row-shifts. Let us denote by P_i the matrix representing the permutation corresponding to a cyclic upward row shift of i. The system of equations that we aim to solve takes the following form

$$\begin{aligned}
&\mathbf{x}_v P^{(1)} \mathbf{x}_v = 0, \\
&\mathbf{x}_v P_1 P^{(1)} \mathbf{x}_v = 0, \\
&\ldots \\
&\mathbf{x}_v P_{m-1} P^{(1)} \mathbf{x}_v = 0.
\end{aligned} \tag{10.4}$$

Equivalently, we can imagine that we have the same equation m times, but the permutation is on the $\mathbf{x}_v$ vectors on the left side, performing cyclic shifts on vector entries (a permutation on columns, since the P_i multiply on the right). It is then evident that for vectors $\mathbf{x}_v$ where all of the entries are equal to each other, if $\mathbf{x}_v$ is a solution to the first equation, then $\mathbf{x}_v$ is a solution to the entire system. We generalize this observation to other specific vectors that have a repeating subsequence.

Let us denote by $\sim$ the binary relation on $\mathbb{F}_q^v$ described informally as "$\mathbf{a}$ is equal to $\mathbf{b}$ up to a cyclic right-shift (without loss of generality)". We add a further restriction that if $\mathbf{b}$ is obtained by performing a right-shift of k

entries on $\mathbf{a}$, then k is the smallest number for which this holds. This is indeed an equivalence relation because (i) $\mathbf{a} \sim \mathbf{a}$ by a shift of zero, (ii) if $\mathbf{a}$ is obtained by performing a right-shift of k on $\mathbf{b}$, then $\mathbf{b}$ can be obtained by a right-shift of $v - k$ on $\mathbf{a}$, and (iii) if $\mathbf{a}$ is a k-shift away from $\mathbf{b}$ and $\mathbf{b}$ is an l-shift away from $\mathbf{c}$, then $\mathbf{a}$ is a $((k+l) \bmod v)$-shift away from $\mathbf{c}$. We also know that, for a given v, the number of such equivalence classes in $\mathbb{F}_q^v$ can be derived by looking at the divisors of v. For each divisor d of v, we have up to q^d equivalence classes of size d. We now make the following observation for the system in (10.4). If $\mathbf{x}$ belongs to an equivalence class of size d and $\mathbf{x}$ is a solution to the first d equations in (10.4), then $\mathbf{x}$ is a solution to the entire system. We also have that all of the vectors in the equivalence class are a solution to the first equation. This observation tells us that the system does not behave like a *random* system and that there are some vectors that are probabilistically more likely to be a solution to the system than others. We exploit this by looking for such solutions using the following strategy.
For each divisor d of v, excluding v and taken in ascending order, build a smaller system by taking the first d equations of the initial system in (10.4) and replacing the unknown $\mathbf{x}_v = (x_1, \ldots, x_v)$ by $\mathbf{x} = (x_1, \ldots, x_d, x_1, \ldots, x_d, \ldots, x_1, \ldots, x_d)$. This is a quadratic system of d equations in d variables. The complexity of solving such systems is (usually) exponential, but for a parameter v that has some small divisors, as in the case of MQ-Sign parameters, we can solve it in practice. The results of applying this strategy to the parameters of MQ-Sign are presented in the next section.
With this attack, we are able to find one (or a few) out of many elements in $V(\mathbf{0})$. The secret oil subspace, denoted O, is contained in $V(\mathbf{0})$ and in the case that the obtained vector is part of the oil subspace this would lead to a full key recovery. Consider the polar form $\mathcal{P}^{(k)\prime}$ of a quadratic form $\mathcal{P}^{(k)}$ defined as

$$\mathcal{P}^{(k)\prime}(\mathbf{x}, \mathbf{y}) = \mathcal{P}^{(k)}(\mathbf{x} + \mathbf{y}) - \mathcal{P}^{(k)}(\mathbf{x}) - \mathcal{P}^{(k)}(\mathbf{y}).$$

When $(\mathbf{x}, \mathbf{y})$ in O^2, we have that $\mathcal{P}^{(k)}(\mathbf{x}) = 0$ and $\mathcal{P}^{(k)}(\mathbf{y}) = 0$ because $O \subseteq V(\mathbf{0})$. In addition, we have that $\mathcal{P}^{(k)}(\mathbf{x} + \mathbf{y}) = 0$ because O is a linear subspace of $\mathbb{F}_q^n$, so $\mathbf{x} + \mathbf{y}$ is also in O. This gives us the additional constraint that $\mathcal{P}^{(k)\prime}(\mathbf{x}, \mathbf{y}) = 0$ for all pairs of vectors in the oil space. When the first oil vector is known, this constraint gives us m linear equations in the variables of the other vector. Coupled with the equations from $\mathcal{P}^{(k)}(\mathbf{y}) = 0$, we obtain a system that is significantly easier to solve than obtaining the first oil vector. This is for instance used to build the reconciliation attack [DYC+08], and recently it was shown that once we have found the first oil vector, the remaining steps to recover the entire oil space can be

Security level	I	III	V
Parameters (v, m)	$(72, 46)$	$(112, 72)$	$(148, 96)$
Tried divisors	1, 2, 3, 4, 6, 8, 9, 12	1, 2, 4, 7, 8, 14	1, 2, 4
Max. running time (s)	2.23	72.36	0.06
Success rate	132/300	32/300	5/300

Table 10.1: Experimental results for finding a preimage of $\mathbf{0}$ in MQ-Sign-LR.

done in polynomial time [Péb24]. This reasoning can not be used when we have a vector that is in the variety of 0 but not in the oil space because the variety of 0 is not a linear subspace. In this case, we can only infer that $\mathcal{P}^{(k)\prime}(\mathbf{x}, \mathbf{y}) = \mathcal{P}^{(k)}(\mathbf{x} + \mathbf{y})$ for $\mathbf{x}, \mathbf{y}$ in $V(\mathbf{0})^2$, which does not give any additional constraints. The attack could indeed recover an oil vector, but the probability of this lucky guess is too low to exploit. Since $V(\mathbf{0})$ is of dimension $n - m$ and O is of dimension m, the probability that the vector we obtain is in O is q^{-n+2m}. For MQ-Sign parameters, this is 2^{-208}, 2^{-320}, and 2^{-416} for the three security levels respectively. Hence, we conclude that this forgery attack does not lead to a key-recovery attack.

10.4.1 Experimental results

We implemented the attack in Magma and experimented with the parameters of all three security levels. MQ-Sign is defined over the finite field $\mathbb{F}_{2^8}$ for all parameter sets. We ran 300 experiments for each set of parameters and the results are presented in Table 10.1. We opted for solving systems of at most 14 equations in 14 variables, which has a running time of about a minute on our machine. We tried only small systems, so that we can have many runs and get a clear view of the success rate of the attack. The probability of success would naturally increase if we use more computational power and attempt the attack with higher divisors of v. The third row in Table 10.1 shows for which divisors we attempted the attack and we see that the more choices we have, the better the success rate. For instance, we can see that for security level V we have the lowest success rate because we could only perform the attack with divisors 2 and 4; the next divisor is 37, which is not practical for any algebraic solver. We conclude that the attack is practical with nonnegligible probability for all security levels and that countermeasures need to used to prevent such an attack.

10.4.2 Forging a signature for any message

We generalize these findings in [RT24] and use them to compute a preimage of many such target vectors that have a particular shape. In fact, the strategy explained here can be extended to any periodic $\mathbf{t}$. Let $d \mid v$ with $d \leq m$ and let $\mathbf{t}$ be a d-periodic vector, that is, $t_i = t_{i+d}$ for all $1 \leq i \leq m - d$. Then for any d' with $d \mid d'$ and $d' \mid v$ we can apply the same trick and look for d'-periodic solutions $\mathbf{x} = (\mathbf{x}_{d'}, \ldots, \mathbf{x}_{d'})$. For each security level, we compute the number of targets for which a preimage can be found with complexity less than the corresponding security threshold. We find that for all levels, it is possible to find a preimage of a $v/2$-periodic hash $\mathbf{t}$. Recall that $\mathbf{t}$ is obtained from a hash function taking as input the message concatenated by a chosen salt r. The salt is chosen by the signer. Hence, we can recompute $\mathbf{t}$ using different values for r until we find one that results in a $v/2$-periodic $\mathbf{t}$. Obtaining a $\mathbf{t}$ with a shorter period is even more advantageous for the rest of the attack, but the number of expected attempts is greater than 2^{λ}, where λ is the security parameter. Thus, we focus on finding a $v/2$-periodic target, and the expected number of attempts for this goal is given in Table 10.2.

Level	q	v	m	salts
I	2^8	72	46	2^{80}
III	2^8	112	72	2^{128}
V	2^8	148	96	2^{176}

Table 10.2: The average number of salts to try before finding a $v/2$-periodic hash.

The complexity of finding a preimage of a $v/2$-periodic target can be computed by taking a straightforward FXL algorithm [CKPS00] as reference. FXL is a hybrid algorithm where we first guess the value of some variables (the number of which is determined as the optimal trade-off) and then we compute a Gröbner basis and extract a solution. In addition, we introduce an improved guessing strategy tailored to this problem in [RT24]. We give the complexity of this solving algorithm for all security levels in Table 10.3.

Level	q	v	m	$\log_2$ cost
I	256	72	46	112
III	256	112	72	173
V	256	148	96	220

Table 10.3: The theoretical complexity of forging MQ-Sign-LR signatures.

Comparing the complexities in both tables, we conclude that the bottleneck of the attack is the resolution of a polynomial system to find the preimage. The complexity of this part of the attack is lower than the required security level for all MQ-Sign parameter sets.

10.4.3 Countermeasures

A straightforward countermeasure for this attack would be to choose parameters such that v is a prime number. In this case, the only targets for which we are able to find preimages easily are the 1-periodic targets. Obtaining a 1-periodic $\mathbf{t}$ by recomputing the hash using different salts is out of reach. In this case, the cost of sampling salts would dominate the cost of the attack and exceed the security requirement.
This attack is also countered in the case where the equivalent keys optimization is not used and the secret linear transformation S is a randomly generated invertible matrix. However, we would not recommend this as the only countermeasure, because in the case of MQ-Sign-LR, the Extended Isomorphism of Polynomials with One Secret (EIP1S) problem underlying the security of the system is different than for traditional UOV. As a result, further analysis is needed to determine whether this difference can be exploited by an attacker.

10.5 Secure MQ-Sign variant

None of the attacks on MQ-Sign in the first or second round affect the conservative variant MQ-Sign-RR. There is strong confidence that this variant is secure, especially because it is equivalent to the traditional UOV scheme. In parallel work, UOV with the implementation and parameter choices outlined in [BCH+23] was submitted to NIST's additional call for signatures in summer 2023. These two instantiations of traditional UOV have almost identical parameter choices and some differences in implementation choices. We outline here those differences. We refer to the UOV instantiation in [BCH+23] as Modern UOV.

- MQ-Sign proposes a nonconservative variant MQ-Sign-LR that aims at reducing the size of the secret key. There are doubts about the security of this variant and we express some of them in previous sections. Modern UOV does not have a variant that differs from traditional UOV.
- MQ-Sign uses a traditional representation of the secret key, storing the central map $\mathcal{F}$ and the linear transformation S. Modern UOV stores

Security level	MQ-Sign-RR	Modern UOV
I	276649	237896
III	1044385	1044320
V	2436769	2436704

Table 10.4: Size (in Bytes) of the secret key of MQ-Sign-RR and Modern UOV.

only a basis of the oil space and presents a corresponding signing algorithm. This also allows Modern UOV to offer a variant with compressed public keys.

- For solving linear systems of equations, MQ-Sign uses the Block Matrix Inversion (BMI) method proposed in [SLK22]. Modern UOV uses Gaussian Elimination techniques instead, and this implementation choice is substantiated both theoretically and experimentally in [BCH$^+$23] with a comparison to the BMI technique.
- MQ-Sign uses offline precomputation to improve the signing runtime.
- MQ-Sign uses a binding technique and Modern UOV does not. We do not know of an attack scenario that would target Modern UOV and would be avoided by MQ-Sign because of the binding technique.
- Even though the secret keys of Modern UOV and MQ-Sign-RR consist of entirely different data structures, they have comparable sizes. Table 10.4 shows this comparison.

10.6 Security proofs

The round 1 submission of MQ-Sign did not include any claim about provable security despite a paragraph discussing a work by Sakumoto, Shirai and Hiwatari [SSH11] who gave a security proof for a minor variation of UOV that includes a random nonce in the message hash following the proof for Full Domain Hash (FDH). This proof reduces the UOV problem to the hardness of the modified UOV signature. The round 2 submission still does not include a security proof for MQ-Sign. The authors collect a number of assumptions and refer again to [SSH11]. However, the authors now claim explicitly that the proof from [SSH11] applies to MQ-Sign (without giving any detail).
From the provided text, it is unclear what the implications of the proof from [SSH11] for MQ-Sign are, and what the roles of the remaining two assumptions (MQ- and EIP-problem) play. Moreover, the proof does not hold

against quantum adversaries, as it is given in the conventional random oracle model (ROM), instead of the quantum-accessible ROM (QROM). Furthermore, the UOV assumption is lacking precision in not defining the distribution from which the $\mathbf{y}$ values are picked.
A minimum of arguments, required to apply the proof from [SSH11], is:

- Change the definition of the UOV problem to use a uniformly random $\mathbf{y}$. This is necessary to ensure that the random oracle can be simulated for the challenge value. Note, it is also necessary to analyze the hardness of exactly this problem later (e.g., by relating it to the hardness of another, more general problem).

- Rephrase MQ-Sign in terms of GenUOVfunc and argue that the resulting scheme is indistinguishable from the specified scheme. This is necessary to show that the public maps used in MQ-sign are actually defining random UOV instances.

- Show that the map $\mathcal{P}$ is a preimage sampleable trapdoor function. For this it is necessary to show that there exists a distribution over the preimage space $\mathbb{F}_q^n$ which
  - is efficiently sampleable without knowledge of a trapdoor,
  - is indistinguishable from the distribution of preimages generated using the trapdoor sampler (i.e., the distribution of the $\mathbf{z}$ values produced by the signing algorithm), and
  - is mapped to the uniform distribution over $\mathbb{F}_q^m$ by $\mathcal{P}$ (or a computationally indistinguishable distribution).

  This is necessary to argue that the simulation of the random oracle is sound.

If these requirements are fulfilled, the result can be lifted to the QROM following the recent work of Kosuge and Xagawa [KX24].
Without a detailed proof that includes detailed arguments for the above points, there are no provable security guarantees for MQ-Sign.

10.7 Editorial Issues

The two loops in Algorithm 2 are defined in a very confusing way: We believe that this is meant to be one loop reaching from line 1 to line 9, with line 6 being an early restart if A is not invertible. This should instead be phrased along the lines of "**if** $\nexists A^{-1}$: **continue**".

In the same Algorithm we suspect that the S_O^T in line 15 should be spelled as s_O^T (with a lowercase s).

The *replacement* attack described in the submission document [SK24, Section 3.2] is dedicated to the new MQ-Sign-LR variant, but it is not fully developed, it has no complexity estimates and it is not taken into account in the parameter choice of MQ-Sign. We did not find a way to develop this attack further. It seems that this description serves as an argument for the robustness of MQ-Sign-LR, showing that a possible linearization and rewrite of the system in (10.3) also results in a central map where all matrices are of full rank. We believe that the exposition would be improved if this is introduced as an intuitive security argument, instead of as an attack.

10.8 Round-2 C software

The sizes of keys etc. in the round-2 software match the documentation.

One of SUPERCOP's signature tests is whether a signed message can be stored in the same array as the original message. The original MQ-Sign software does not allow this. We modified the software to allow this.

Other SUPERCOP tests detect failures for MQLR that we eliminated by modifying the software to 0-initialize the buffers used to store the public key and the private key. This could indicate a more serious bug. The submitters are encouraged to investigate this.

A portability issue is that the software sometimes uses variable declarations after labels; this is allowed by some compilers but not others. We changed `label: int foo` to `label: ; int foo` to fix this.

A correctness issue, and potentially a security issue, is that the software stores message lengths in 32-bit variables. Presumably it is possible to fix this by replacing the 32-bit variables with 64-bit variables.

TIMECOP identified two issues in the reference MQ-Sign software. First, there are some rejection-sampling loops in `mqs.c`; we added `crypto_declassify`. Second, there is a real timing leak in `rng.c`, which is software that NIST provided and encouraged people to use for seed expansion but that was never designed to run in constant time.

We rewrote part of `rng.c` to use `crypto_uint8_zero_mask`, a constant-time support function provided by SUPERCOP. The usage of AES inside the NIST code is expected to produce further timing variations on some platforms, as illustrated by [CKP+20].

A preliminary assessment indicates that, on platforms where AES runs in constant time, the timing leak is limited to a small number of bytes from a fresh buffer for each message. However, it would still be better to eliminate `rng.c` and call SHAKE256 with the seed as input to generate the desired number of bytes of output.

TIMECOP identified a further issue in the `avx2` software. The software uses table lookups to multiply in $\mathbb{F}_{256}$. This should be fixed; perhaps it is exploitable. We decided to submit the software for the 25 June 2024 release of SUPERCOP, marking the `ref` software but not the `avx2` software as constant-time; SUPERCOP reports the results for both, with a red "T:" for the `avx2` software.

The MQ-Sign team distributed new software on 17 July 2024 (not compatible with previous MQ-Sign software). We submitted a corresponding MQ-Sign update for the 8 August 2024 release of SUPERCOP.

We subsequently modified `avx2` to obtain an `avx2ct` that passes TIMECOP, and submitted that for the 9 September 2024 release of SUPERCOP. This still has noticeably slower signing than `avx2`.

For example, for `mqsignlr2567246`, the SUPERCOP signing results for Skylake show (quartile, median, quartile) of $(103313, 107424, 117000)$ cycles (from `avx2`) with "T:", and $(118967, 124404, 131977)$ cycles (from `avx2ct`) without "T:". Before the September 2024 update, the results showed more than 500000 cycles (from `ref`) without "T:".

The MQ-Sign documentation says 65300+51744 cycles for the AVX2 code for sign+verify and 451262+774652 cycles for the reference code; this is on an Intel Xeon Gold 6234 (Cascade Lake). We tried running the MQ-Sign benchmarking tool on Skylake; it reported 78445+58329 cycles for the original MQ-Sign code, and 81421+57972 cycles after our patches.

KpqClean reports 43696+35239 cycles on Comet Lake. We re-ran KpqClean on a Comet Lake with overclocking disabled; it reported 99940+80662 cycles. We also ran KpqClean on a Skylake with overclocking disabled; it reported 76426+58099 cycles.

The discrepancies here warrant further investigation. For example, if some compiler options are producing better speeds, then the faster assembly-language results can be added to SUPERCOP. Another reason for a discrepancy is that the MQ-Sign and KpqClean benchmarking tools do not measure the time to generate a "seed" and "salt" used in signing.

Tables 10.5, 10.6, and 10.7 show the number of instructions used for key generation, signing, and verification respectively inside AVX2 software for `mqsignlr2567246`. Arithmetic in $\mathbb{F}_{256}$ accounts for more than 50% of the instructions in MQ-Sign. The software does not appear to be using the speedup techniques from, e.g., [KS09] and [BMP13].

1934208	9.73%	blas_avx2.h:batch_trimat_madd_multab_gf256_avx2
1450656	7.30%	blas_avx2.h:batch_trimatTr_madd_multab_gf256_avx2
1329768	6.69%	avx2intrin.h:batch_trimatTr_madd_multab_gf256_avx2
1208880	6.08%	avx2intrin.h:batch_trimat_madd_multab_gf256_avx2
1087995	5.47%	avxintrin.h:batch_trimatTr_madd_multab_gf256_avx2
1077938	5.42%	parallel_matrix_op_avx2.c:batch_trimat_madd_multab_gf256_avx2
1066890	5.37%	emmintrin.h:AES_256_Key_Expansion
966066	4.86%	rng.c:randombytes_with_state
846216	4.26%	parallel_matrix_op.h:batch_trimatTr_madd_multab_gf256_avx2
827642	4.16%	memset-vec-unaligned-erms.S:__memset_avx2_unaligned_erms
809835	4.07%	parallel_matrix_op_avx2.c:batch_trimatTr_madd_multab_gf256_avx2
788164	3.96%	avx2intrin.h:batch_matTr_madd_multab_gf256_avx2
754632	3.80%	crypto_uint8.h:randombytes_with_state
725328	3.65%	gf16_avx2.h:batch_trimat_madd_multab_gf256_avx2
725328	3.65%	gf16_avx2.h:batch_trimatTr_madd_multab_gf256_avx2
687884	3.46%	blas_avx2.h:batch_matTr_madd_multab_gf256_avx2
665712	3.35%	gf16_avx2.h:batch_matTr_madd_multab_gf256_avx2
493185	2.48%	aes.c:AES_ECB_encrypt
443210	2.23%	avxintrin.h:batch_matTr_madd_multab_gf256_avx2
392472	1.97%	avxintrin.h:batch_trimat_madd_multab_gf256_avx2
292937	1.47%	memmove-vec-unaligned-erms.S:__memcpy_avx_unaligned_erms
191235	0.96%	rng.c:AES256_ECB
161040	0.81%	aes.c:AES_256_Key_Expansion
952863	4.79%	others
19880084	100%	

Table 10.5: Instructions used for AVX2 software for `mqsignlr2567246` key generation. Each table row tallies the number of instructions used directly by one function. If function F calls function G then instructions inside G are tallied for G, not for F.

45594	13.88%	crypto_uint32.h:gf256mat_mul_avx2
32577	9.92%	blas_matrix_avx2.c:gf256mat_prod_multab_avx2
28578	8.70%	avx2intrin.h:gf256mat_inv_avx2
22307	6.79%	avxintrin.h:gf256mat_mul_avx2
19727	6.01%	avx2intrin.h:gf256mat_prod_multab_avx2
19532	5.95%	fips202.c:KeccakF1600_StatePermute
18366	5.59%	blas_matrix_avx2.c:gf256mat_inv_avx2
16090	4.90%	blas_avx2.h:gf256mat_mul_avx2
11766	3.58%	gf16_avx2.h:gf256mat_inv_avx2
10730	3.27%	blas_matrix_avx2.c:gf256mat_gaussian_elim_avx2
8282	2.52%	blas_matrix_avx2.c:gf256mat_mul_avx2
8013	2.44%	avx2intrin.h:gf256mat_gaussian_elim_avx2
7002	2.13%	avx2intrin.h:gf256mat_mul_avx2
6013	1.83%	memset-vec-unaligned-erms.S:__memset_avx2_unaligned_erms
5559	1.69%	blas_matrix_avx2.c:gf256mat_prod_avx2
5528	1.68%	avx2intrin.h:gf256mat_prod_avx2
4266	1.30%	blas_sse.h:gf256v_set_zero
4084	1.24%	fips202.c:shake256
4004	1.22%	blas_sse.h:gf256mat_solve_linear_eq_avx2
3592	1.09%	blas_avx2.h:circmat_eval_multab_avx2
3520	1.07%	blas_sse.h:circmat_eval_multab_avx2
3362	1.02%	fips202.c:keccak_absorb_once.constprop.1
3339	1.02%	gf16_avx2.h:gf256mat_gaussian_elim_avx2
36636	11.15%	others
328467	100%	

Table 10.6: Instructions used for AVX2 software for `mqsignlr2567246` signing. Each table row tallies the number of instructions used directly by one function. If function F calls function G then instructions inside G are tallied for G, not for F.

71154	35.51%	`avx2intrin.h:mqlr_verify`
42952	21.44%	`gf16_avx2.h:mqlr_verify`
42362	21.14%	`blas_avx2.h:mqlr_verify`
29383	14.66%	`mpkc_avx2.h:mqlr_verify`
7148	3.57%	`avxintrin.h:mqlr_verify`
4883	2.44%	`fips202.c:KeccakF1600_StatePermute`
1224	0.61%	`fips202.c:keccak_absorb_once.constprop.1`
1021	0.51%	`fips202.c:shake256`
109	0.05%	`mqs_simd.c:mqlr_verify`
47	0.02%	`utils_hash.c:hash_msg`
32	0.02%	`sign.c:crypto_sign_open`
25	0.01%	`memset-vec-unaligned-erms.S:__memset_avx2_unaligned_erms`
22	0.01%	`memmove-vec-unaligned-erms.S:__memcpy_avx_unaligned_erms`
10	0.01%	`sign.c:main`
0	0.00%	others
200372	100%	

Table 10.7: Instructions used for AVX2 software for `mqsignlr2567246` verification. Each table row tallies the number of instructions used directly by one function. If function F calls function G then instructions inside G are tallied for G, not for F.

Chapter 11

NCC-Sign: A New Lattice-based Signature Scheme using Non-Cyclotomic Polynomials

The main idea of NCC-Sign is to take the Dilithium design and replace the mathematical structure used. While Dilithium uses a module lattice, NCC-Sign moves to ideal lattices.

11.1 System description

NCC-Sign [SKA22] is based on Lyubashevky's signature scheme [Lyu12] using Fiat–Shamir with aborts, a signature compression technique by Bai and Galbraith [BG14], and the public-key compression technique from Dilithium [LDK+20] on the signature side and on NTRU Prime [BCLv17, BBC+20b] and NTTRU [LS19] on the ideal lattice side.
The main part of the submission, as also reflected in the title, uses ideal lattices over the NTRU Prime field $R = \mathbf{Z}[x]/(x^p - x - 1)$ modulo a prime q, where q is chosen such that $x^p - x - 1$ is irreducible modulo q and that q is inert in $\mathbf{Q}[x]/(x^p - x - 1)$. The authors follow NTRU Prime in pointing to security concerns related to the cyclotomic structure and the many subfields present in the typical choice of $x^n + 1$ for $n = 2^d$.
The submission also considers a version using cyclotomic polynomials of the form $x^{2n} - x^n + 1$ for $n = 2^a 3^b$. These cyclotomic polynomials were proposed in [LS19] to add flexibility in the dimension beyond $n = 2^d$ while keeping the speed benefits of NTT-friendly rings. In the original NCC-Sign submission

from October 2022 this version is presented in Section 3.5 and Table 6 while all implementation considerations cover only the non-cyclotomic case. In the updated version [SKA23], labeled v1.0 in that document, two sets of parameters are proposed for this case and implementation results indeed show better speeds, however, most of the text still focuses on the non-cyclotomic case.

The round-2 NCC-Sign submission has some additional comments on the cyclotomic case, and changes the choice of parameters for that case. The round-1 submission proposed cyclotomics of degrees 1152, 1536, 2048, and 2304; the round-2 submission proposes cyclotomics of degrees 1024, 1458, and 1944. For the non-cyclotomic case, the proposed degrees are 1021, 1201, 1429, 1607, 1913, and 2039, and we did not find any changes in the further parameter-set details (q, d, etc.). Our evaluations of concrete parameters focus on the non-cyclotomic case.

KeyGen, Sign, and Verify as well as the supporting algorithms match those of Dilithium. The difference is that where Dilithium uses module lattices, NCC-Sign uses ideal lattices. The supporting algorithms are defined for the coefficients and thus match 1-to-1. For the other functions, matrices of polynomials are replaced by polynomials.

KeyGen generates a public polynomial $\mathbf{a} \in R_q$ from some seed ζ, this seed forms the first part of the public key. The second part is an RLWE sample using $\mathbf{a}$: Pick small $(\mathbf{s}_1, \mathbf{s}_2)$ and compute $\mathbf{t} = \mathbf{a}\mathbf{s}_1 + \mathbf{s}_2$, where "small" means that the coefficients are in $\{0, \pm 1, \pm 2\}$; the July 2023 version added an option for using $\{0, \pm 1\}$. The public key includes only the top part of $\mathbf{t}$ while the bottom part is included in the secret key along with the small polynomials $\mathbf{s}_1$ and $\mathbf{s}_2$. The secret key additionally includes the seed ζ, the hash $ph = \mathrm{H}(\zeta, \mathbf{t}_1)$ of the public key (the round-2 submission renames this hash "tr"), and a string dK to generate pseudo-random numbers in signing.

Top and bottom parts of $\mathbf{t}$ are defined as follows: Assume that the coefficients of $\mathbf{t}$ are in $[0, q-1]$. Let $\mathbf{t}_0$ be the polynomial whose coefficients are computed from the coefficients of $\mathbf{t}$ taking the remainder under division by 2^d using representatives inside $(-2^{d-1}, 2^{d-1}]$. Then $\mathbf{t}_1 = (\mathbf{t} - \mathbf{t}_0)/(2^d)$. Since the division here is by a power of 2, the top part basically means the top bits of each coefficient, apart from the detail that the remainder can be negative which then adds 1 to the top part. Other functions use top and bottom parts for more general moduli γ, using the same approach of computing the remainder centered around 0 and then taking the quotient of division by γ after subtracting the remainder.

The signature should show that the signer indeed knows $\mathbf{s}_1, \mathbf{s}_2$ matching $\mathbf{t}_1$. To sign message M, first a random commitment $\mathbf{y} \in R_q$ with restricted coefficients is sampled; this process may need to be repeated (the aborts

part), hence the sampling includes a counter κ. The randomized version version picks a 512-bit ρ at random while for the deterministic version ρ depends on M, ph, and dK. Here, "restricted" is not as small as in the key generation but coefficients of $\mathbf{y}$ are in $(-2^{17}, 2^{17}]$ for the 128-bit security level and the range doubles for each level. Then $\mathbf{w} = \mathbf{ay}$ is computed and only the quotient after division by some γ_2 is taken, the remainder is centered around 0. The parameter γ_2 is chosen to be co-prime to q and to be between $q/2^7$ and $q/2^5$. (The full details include one corner case, see the Decompose function.)
The challenge is then given by $\tilde{c} = \mathrm{H}(\mu, \mathbf{w}_1) \in \{0,1\}^{256}$, where $\mu = \mathrm{H}(ph, M)$. This $\tilde{c}$ is then used to deterministically sample a fixed-weight polynomial $\mathbf{c}$, having τ coefficients in $\{-1, 1\}$ and the rest being 0, where $\tau = 25$ for the smallest parameters and 32 for the largest.
The polynomial $\mathbf{z} = \mathbf{y} + \mathbf{cs}_1$ then uses the secret key. However, the public key is computed using also $\mathbf{s}_2$ and the public key only includes the top part of $\mathbf{t}$, hence the next steps in signature generation ensure that verification can proceed. First it is checked that $\mathbf{z}$ does not leak information on $\mathbf{s}$, for that it is checked that none of the coefficients are larger than some bound $\gamma_1 - \beta$, where γ_1 was the bound on the coefficients of $\mathbf{y}$ and $\beta = 4\tau$. This means that $\mathbf{z}$ does not depend on the secret. Signature verification for Bai–Galbraith signature compression reconstructs the top part of $\mathbf{w}$ as the top part of $\mathbf{az} - \mathbf{ct} = \mathbf{ay} + \mathbf{acs}_1 - \mathbf{cas}_1 - \mathbf{cs}_2 = \mathbf{w} - \mathbf{c} - \mathbf{s}_2$ which matches the top part of $\mathbf{w}$ if $\mathbf{s}_2$ is sufficiently small. This is checked by checking that the centered remainder of $\mathbf{w} - \mathbf{cs}_2$ after division by $2\gamma_2$ has no coefficient larger than $\gamma_2 - \beta$, because $\beta = 2\eta\tau$ is the maximum size a coefficient of $\mathbf{cs}_2$ can have (note that reductions modulo $x^p - x - 1$ cause the extra factor of 2 here). If either of these are violated, the counter κ is incremented and a new $\mathbf{y}$ is sampled.
There is an indentation error in the signing function in both versions of NCC-Sign [SKA22, SKA23] as Step 20 needs to be indented less (be at the same level as **if** and **else**). We pointed this out in the round-1 report. This still has not been fixed in the round-2 submission. The algorithm as written will often loop forever.
NCC-Sign follows Dilithium in additionally compressing the public key (including only $\mathbf{t}_1$ instead of $\mathbf{t}$), which means that only $\mathbf{ct}_1$ is available, leading to $\mathbf{w} - \mathbf{cs}_2 + \mathbf{ct}_0$. The signature includes a vector $\mathbf{h}$ of hints, which are 1 in the positions in which the high parts of $\mathbf{w} - \mathbf{cs}_2$ and of $\mathbf{w} - \mathbf{cs}_2 + \mathbf{ct}_0$ differ. Valid signatures are limited in how large the Hamming weight of $\mathbf{h}$ is permitted to be as these hints give extra flexibility to a forger. If the calculated $\mathbf{h}$ has too large weight κ is incremented and a new $\mathbf{c}$ is sampled.
Eventually all checks succeed and the signature is $(\tilde{c}, \mathbf{z}, \mathbf{h})$.

To verify signature $(\tilde{c}, \mathbf{z}, \mathbf{h})$ on M compute $\mathbf{c}$ from $\tilde{c}$ and compute the high part $\mathbf{w}'_1$ of $\mathbf{az} - \mathbf{ct}_1 \cdot 2^d$ using the hint vector $\mathbf{h}$. If this computation worked correctly, $\mathbf{w}_1 = \mathbf{w}'_1$ and $\tilde{c} = \mathrm{H}(\mu, \mathbf{w}'_1)$, hence this forms the verification check along with checking the weight of $\mathbf{h}$ and the coefficient sizes of $\mathbf{z}$.
By the above, an honestly generated signature passes verification.

11.2 Security

The suitability of the underlying lattice problem has been argued in NTRU Prime and NTTRU respectively. While NTRU Prime cautions of using cyclotomic lattices no actual attacks on RLWE or RLWR with power-of-2 cyclotomics are known. The general strategy of NCC-Sign equals that of Dilithium and is thus well studied. As we comment in the section on security proofs, the differences are not fully explored and it is not clear that the proofs hold, however, we have not been able to turn the differences into attacks. The most visible difference, caused by the asymmetry in how reductions modulo $x^p - x - 1$ affect the coefficients, has been taken into account by the designers. There is no matching counterpart for the cyclotomic polynomial. This leaves generic attacks as the main attack avenue and guidance on choosing parameters.
A general issue with the security evaluations of Dilithium, HAETAE, NCC-Sign, etc. is the following gap:

- The proofs hypothesize the hardness of "self-target" problems such as SelfTargetRSIS and SelfTargetMSIS.
- The analyses of security levels focus on the simplified problem SIS.

There has been some attention to the question of whether the ring structure in RSIS and MSIS can reduce security, but it is also important to ask whether the "self-target" structure can reduce security.
Regarding the self-target structure, the Dilithium documentation says that, for a strong hash function, "the only approach for obtaining a solution appears to be" picking a hash input and then solving an SIS-like problem for the hash output. In 2022, Wang, Xia, Shi, Wang, Zhang, and Gu [WXS+22] pointed out that one can instead pick *many* hash inputs and then solve *one of many* SIS-like problems for the same lattice. There are known techniques to save time in attacking multi-target lattice problems (see, e.g., [Ber22b] and [WXS+22]), and further analysis is required of the concrete impact. For a generic hash function, this approach cannot save more than a factor q when there are q hash queries, but q can be very large.

11.2.1 Generic lattice attacks

The results of the lattice estimator for the non-cyclotomic version of NCC-Sign (split up over per security level and the choice of η) can be found in Table 11.1. The Trionomial version can be found in Table 11.2. These results are quite close to the reported security levels.

Security level	η	Primal Core-SVP	Dual Core-SVP	BDD	Dual-Hybrid
1	2	144	142	167	167
3	2	209	204	229	227
5	2	280	272	298	294
1	1	132	129	156	154
3	1	192	186	214	209
5	1	259	249	278	270

Table 11.1: Estimated security levels for NCC-Sign Non-Cyclotomic based on the lattice estimator [APS15]

Security level	Primal Core-SVP	Dual Core-SVP	BDD	Dual-Hybrid
1	132	129	156	154
3	191	185	212	208
5'	274	262	292	284
5	316	302	333	322

Table 11.2: Estimated security levels for NCC-Sign Trinomial based on the lattice estimator [APS15]

11.3 Implementation considerations

The designers observe that reducing $\mathbf{cs}_i$ modulo $x^p - x - 1$ leads to higher weight in the bottom half of the result. They modify the sampler to split $\mathbf{c}$ into top and bottom parts $\mathbf{c} = \mathbf{c}_2 + x^{p_2}\mathbf{c}_1$ with weight τ_i in $\mathbf{c}_i$. Taking $\mathbf{s}_i$ with extremal coefficients $\pm\eta$ leads to a polynomial with constant term and coefficients of x^j with $j \geq p_2$ bounded by $\beta_2 = (2\tau_1 + \tau_2)\eta$ and coefficients of x^j with $1 \leq j < p_2$ bounded by $\beta_1 = 2(\tau_1 + \tau_2)\eta$. They thus suggest to change the distribution of $\mathbf{c}$ to have $\tau_1 < \tau_2$ and to change the size constraints to using β_1 and β_2 instead. There is an error in Table 7 of [SKA22] in that the columns labeled τ_1, τ_2 should be labeled β_1, β_2. The choice of variable name κ for the challenge entropy is bad as κ in the signing algorithm is a counter. While it is confusing that $\tau_1 + \tau_2 \neq \tau$ and no comment is made to

this regard, the examples have $\tau_1 + \tau_2 > \tau$ which is good as the sampler loses some entropy by fixing a split of the weight, e.g., the first parameter set has $\tau_2 = 14, \tau_1 = 12$ for $\tau = 25$.
NCC-Sign gains flexibility over power-of-two cyclotomics by permitting any prime p as length. NTRU Prime has shown that arithmetic in these rings can be competitive with implementations using NTT but NCC-Sign needs larger parameters than the KEM. It is likely that further speedups are possible, using the tooling for code generation in NTRU Prime, but at this moment the speed is much slower. The cyclotomic rings can use the implementation from NTTRU and do obtain better speed in the experiments reported in [SKA23]. In general, there is room for improvement. The round-2 submission reports speedups; our preliminary assessment is that further speedups are possible.

11.4 Provable security of the signature scheme

Given that NCC-Sign closely follows Dilithium, the straight-forward way to prove security is to adopt existing proofs for Dilithium. In the round 1 submission, the authors followed the original proof of Kiltz, Lyubashevsky, and Schaffner [KLS18] for this. However, the proof in [KLS18] was later found to be flawed but also immediately fixed [BBD$^+$23, DFPS23]. In the round 2 submission, the NCC-Sign authors updated their security argument to now follow the proof presented in [BBD$^+$23]. The proof factors into showing security of the signature scheme under No-Message Attacks (UF-NMA) and accepting Honest-Verifier Zero-Knowledge (acHVZK) of the underlying identification scheme. The authors added the analysis of the commitment min-entropy which is a part of the security evaluation in the fixed proof [BBD$^+$23].
Although the presented proof sketch discusses several changes, such as selecting parameters $k = 1$ and $l = 1$, compared to the Dilithium scheme, it does not provide a full proof. First, it is difficult to evaluate the proof without a formal statement of what is being proven. A theorem statement with specified terms in the upper bound on the adversary's forgery capabilities is crucial for a formal evaluation of the scheme. Moreover, such a statement will clarify the choice of parameters for the needed security levels in the following sections.
Regarding the security proof, in [BBD$^+$23], the upper bound consists of four main parts: UF-NMA security, acHVZK, Random Oracle reprogramming terms, and commitment min-entropy requirement. For NCC-Sign, the given proof sketch of UF-NMA security appears to be correct, but it is so com-

pressed that it is not possible to recover a meaningful proof without redoing the proof. For a proper proof, one needs to provide formal statements and a concrete reduction with all coefficients and terms that appear from the reduction itself. The same applies to the HVZK part. An explicit presentation of the underlying identification scheme is also required for a formal treatment. For the proof of HVZK, a discussion on the min-entropy of the commitment and the probability of rejection is required. This min-entropy needs to be analyzed in the general case and conditioned on the occurrence of a "good" event. The Random Oracle reprogramming terms are not mentioned in the security section. It is important not to drop the terms that appeared from the security reduction, as they can have a significant impact on the security parameters of the scheme. Regarding the min-entropy evaluation, the authors claim that variable a is invertible, which is done through rejection sampling for the trinomial case. This case must be discussed in more detail, as a is generated not from the whole domain. Also, this affects the underlying RLWE assumption used.
We also point out several details that need clarifications or improvement:

1. The SampleInBall algorithm should be updated to align with the descriptions provided in the text.

2. Regarding cryptographic asumptions in Definition 1 and Definition 2, for the decisional RLWE assumption, the adversary is typically provided with multiple samples in the security game, with the number of samples specified as a parameter. Additionally, the parameter q for the ring R_q is usually not included in the game parameters.

We did not redo the proofs. While it is reasonable to believe that the proofs go through with the changes in assumptions made by the authors, we do not think one can rely on this without someone carefully doing the proofs and writing them out to enable public scrutiny.

11.5 Round-2 C software

The optimized round-2 software appears to cover only the trinomial cyclotomic options in the documentation (`NIMS_TRI_NTT_MODE` in the software), not the non-cyclotomic options in the documentation.
The private-key sizes in the software are longer than the sizes in the documentation. For example, for the lowest security level, we find 1760-byte public keys, 2688-byte private keys, and 2912-byte signatures where the documentation reports 1760-byte public keys, 2400-byte private keys, and 2912-byte signatures.

SUPERCOP's tests report that different (implementation, compiler) pairs produce different checksums: i.e., these pairs are producing different results even when SUPERCOP's `randombytes` returns the same results. This indicates that the software is obtaining data from uninitialized memory or from external sources of randomness.

Some manual code review identified further reasons for concern regarding the extent to which the software has been checked against the specification. For example, the following code, for the case that `ETA` is 1, makes the output array much less random than it should be:

```
if (t0 < 3) {
        a[ctr++] = 1 - t0;
}
if (t1 < 3 && ctr < len) {
        a[ctr++] = 1 - t0;
}
if (t2 < 3 && ctr < len) {
        a[ctr++] = 1 - t0;
}
if (t3 < 3 && ctr < len) {
        a[ctr++] = 1 - t0;
}
```

The intent was clearly to use `1 - t0`, `1 - t1`, `1 - t2`, and `1 - t3`, producing independent outputs in $\{-1, 0, 1\}$ as specified.

We did some initial work to resolve some TIMECOP complaints. There are many branches that appear to be from rejection sampling (in `poly_chknorm`, `poly_uniform`, etc.), presumably not a security issue. There is Fisher–Yates sampling, presumably a security issue.

The NCC-Sign team issued a 17 July 2024 software update that resolved most issues we had reported, and we submitted NCC-Sign for the 8 August 2024 SUPERCOP release.

We subsequently modified the software to eliminate secret divisions, and submitted the result for the 9 September 2024 SUPERCOP release. This passes TIMECOP on various machines, including machines that are using the TIMECOP patches from [BBB+24] to check for secret divisions.

For `nccsign1aes` on Skylake, SUPERCOP reports medians of 191041 cycles for key generation, 472358 cycles for signing (with quartiles 341500 and 512246), and 231666 cycles for verification. For comparison, on Cascade Lake, the NCC-Sign documentation reports medians of 240496 cycles for key generation, 616746 cycles for signing, and 339698 cycles for verification.

For Comet Lake, KpqClean reports 246285 cycles for key generation, 250040 cycles for signing, and 165103 cycles for verification. We re-ran KpqClean on a Comet Lake with overclocking disabled; it reported 236061, 363541, 246957 in one run, and 235941, 541095, 245345 in a second run.

Tables 11.3, 11.4, and 11.5 show the number of instructions used for key generation, signing, and verification respectively inside AVX2 software for `nccsign1`. Keccak computations use over 50% of the instructions for key generation but under 50% for signing and verification. The NTT and inverse-NTT functions in signing and verification appear to have eliminated most loads and stores but, as in Section 9.5, may still benefit from replacing some shift instructions with permutation instructions.

300649	47.15%	`fips202.c:KeccakF1600_StatePermute`
60322	9.46%	`poly.c:rej_eta.constprop.0`
42048	6.59%	`poly.c:poly_uniform_avx_4way`
18800	2.95%	`avx2intrin.h:invntt_tomont_avx_4way`
18423	2.89%	`crypto_uint32.h:poly_uniform_avx_4way`
16588	2.60%	`poly.c:ntt_avx_4way`
15372	2.41%	`crypto_uint32.h:rej_eta.constprop.0`
15024	2.36%	`avx2intrin.h:ntt_avx_4way`
15011	2.35%	`poly.c:invntt_tomont_avx_4way`
13842	2.17%	`poly.c:poly_caddq`
12672	1.99%	`rounding.c:power2round`
12451	1.95%	`memset-vec-unaligned-erms.S:__memset_avx2_unaligned_erms`
11742	1.84%	`fips202.c:shake128_squeezeblocks`
11520	1.81%	`reduce.c:caddq`
8080	1.27%	`poly.c:poly_power2round`
8064	1.26%	`crypto_int32.h:power2round`
8033	1.26%	`avxintrin.h:ntt_avx_4way`
7489	1.17%	`fips202.c:keccak_absorb_once.constprop.1`
7434	1.17%	`avxintrin.h:invntt_tomont_avx_4way`
5375	0.84%	`sign.c:crypto_declassify`
5224	0.82%	`packing.c:polyeta_pack`
3840	0.60%	`avxintrin.h:vector_4way_to_3way`
2439	0.38%	`packing.c:pack_pk`
17269	2.71%	others
637711	100%	

Table 11.3: Instructions used for AVX2 software for `nccsign1` key generation. Each table row tallies the number of instructions used directly by one function. If function F calls function G then instructions inside G are tallied for G, not for F.

620141	27.75%	`fips202.c:KeccakF1600_StatePermute`
150400	6.73%	`avx2intrin.h:invntt_tomont_avx_4way`
149292	6.68%	`poly.c:ntt_avx_4way`
145152	6.49%	`rounding.c:decompose`
138240	6.19%	`crypto_int32.h:decompose`
135216	6.05%	`avx2intrin.h:ntt_avx_4way`
120088	5.37%	`poly.c:invntt_tomont_avx_4way`
72297	3.23%	`avxintrin.h:ntt_avx_4way`
59472	2.66%	`avxintrin.h:invntt_tomont_avx_4way`
55368	2.48%	`poly.c:poly_caddq`
46281	2.07%	`packing.c:polyz_unpack`
46120	2.06%	`poly.c:poly_chknorm`
46080	2.06%	`reduce.c:caddq`
41583	1.86%	`poly.c:poly_uniform_avx_4way`
37108	1.66%	`fips202.c:shake256_absorb`
34605	1.55%	`poly.c:poly_reduce`
34560	1.55%	`crypto_int32.h:poly_chknorm`
32640	1.46%	`avxintrin.h:vector_4way_to_3way`
28800	1.29%	`reduce.c:reduce32`
24240	1.08%	`poly.c:poly_decompose`
23404	1.05%	`memset-vec-unaligned-erms.S:__memset_avx2_unaligned_erms`
20736	0.93%	`crypto_int32.h:make_hint`
18213	0.81%	`crypto_uint32.h:poly_uniform_avx_4way`
154837	6.93%	others
2234873	100%	

Table 11.4: Instructions used for AVX2 software for `nccsign1` signing. Each table row tallies the number of instructions used directly by one function. If function F calls function G then instructions inside G are tallied for G, not for F.

302746	36.87%	`fips202.c:KeccakF1600_StatePermute`
49764	6.06%	`poly.c:ntt_avx_4way`
46080	5.61%	`crypto_int32.h:use_hint`
45072	5.49%	`avx2intrin.h:ntt_avx_4way`
44046	5.36%	`rounding.c:use_hint`
41583	5.06%	`poly.c:poly_uniform_avx_4way`
37600	4.58%	`avx2intrin.h:invntt_tomont_avx_4way`
30022	3.66%	`poly.c:invntt_tomont_avx_4way`
24099	2.94%	`avxintrin.h:ntt_avx_4way`
20763	2.53%	`poly.c:poly_caddq`
18213	2.22%	`crypto_uint32.h:poly_uniform_avx_4way`
17280	2.10%	`reduce.c:caddq`
15427	1.88%	`packing.c:polyz_unpack`
14868	1.81%	`avxintrin.h:invntt_tomont_avx_4way`
13981	1.70%	`memset-vec-unaligned-erms.S:__memset_avx2_unaligned_erms`
12278	1.50%	`fips202.c:shake256_absorb`
11574	1.41%	`fips202.c:shake128_squeezeblocks`
9600	1.17%	`avxintrin.h:vector_4way_to_3way`
9224	1.12%	`poly.c:poly_chknorm`
8080	0.98%	`poly.c:poly_use_hint`
7032	0.86%	`fips202.c:keccak_absorb_once.constprop.1`
6912	0.84%	`crypto_int32.h:poly_chknorm`
5333	0.65%	`packing.c:polyw1_pack`
29484	3.59%	others
821061	100%	

Table 11.5: Instructions used for AVX2 software for `nccsign1` verification. Each table row tallies the number of instructions used directly by one function. If function F calls function G then instructions inside G are tallied for G, not for F.

Chapter 12

SUPERCOP results for signature software

Figures 12.1, 12.2, 12.3, 12.4, 12.5, 12.6, 12.7, 12.8, and 12.9 are scatterplots showing various measurements from SUPERCOP version 20240909 for AIMer (`aimer`), HAETAE (`haetae`), MQ-Sign (`mqsign`), and NCC-Sign (`nccsign`). As selected baselines for comparison, these figures also show measurements for `dilithium` (not reflecting incompatible Dilithium changes after the latest submission of Dilithium to SUPERCOP), `falcon` (similar issue), `ed25519`, `ed448goldilocks`, and one `sphincs` parameter set.

"`T:`" means that an implementation is not marked as constant-time. MQ-Sign is shown with and without "`T:`" because SUPERCOP includes both variable-time software from the MQ-Sign team and our patched constant-time software; the constant-time software has slower signing.

Some of the baseline software was submitted before SUPERCOP added support for constant-time markers, and has not been updated, so it is also shown with "`T:`".

As in Chapter 7, there is first a CPU-independent plot of sizes vs. sizes, and then timing graphs split between (1) a computer named `titan0` with an Intel Haswell CPU and (2) a computer named `cezanne` with an AMD Zen 3 CPU.

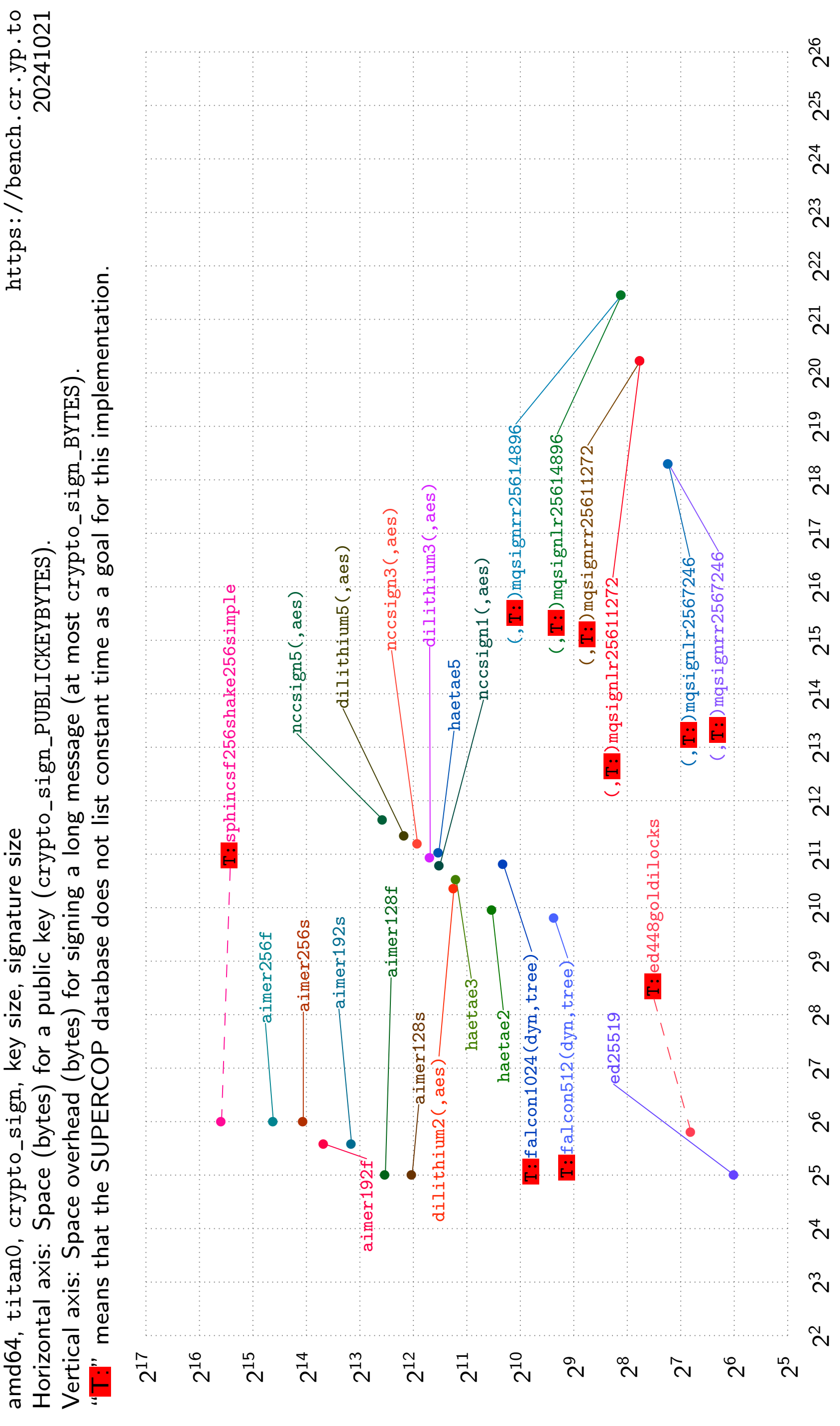


Figure 12.1: Bytes for a signature vs. bytes for a public key.

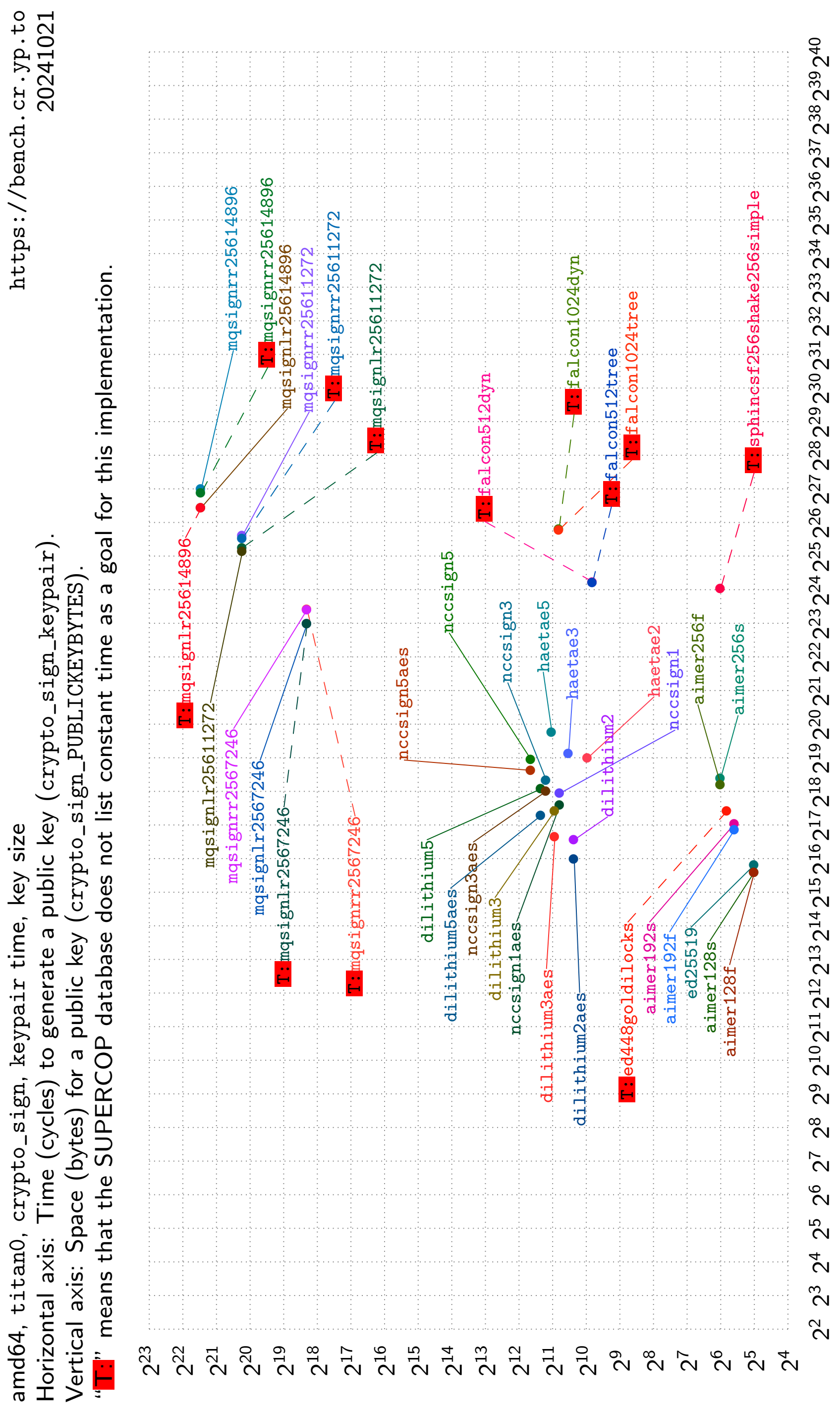


Figure 12.2: Bytes for a public key vs. cycles for key generation. Measurements collected on `titan0`, which has an Intel Haswell CPU.

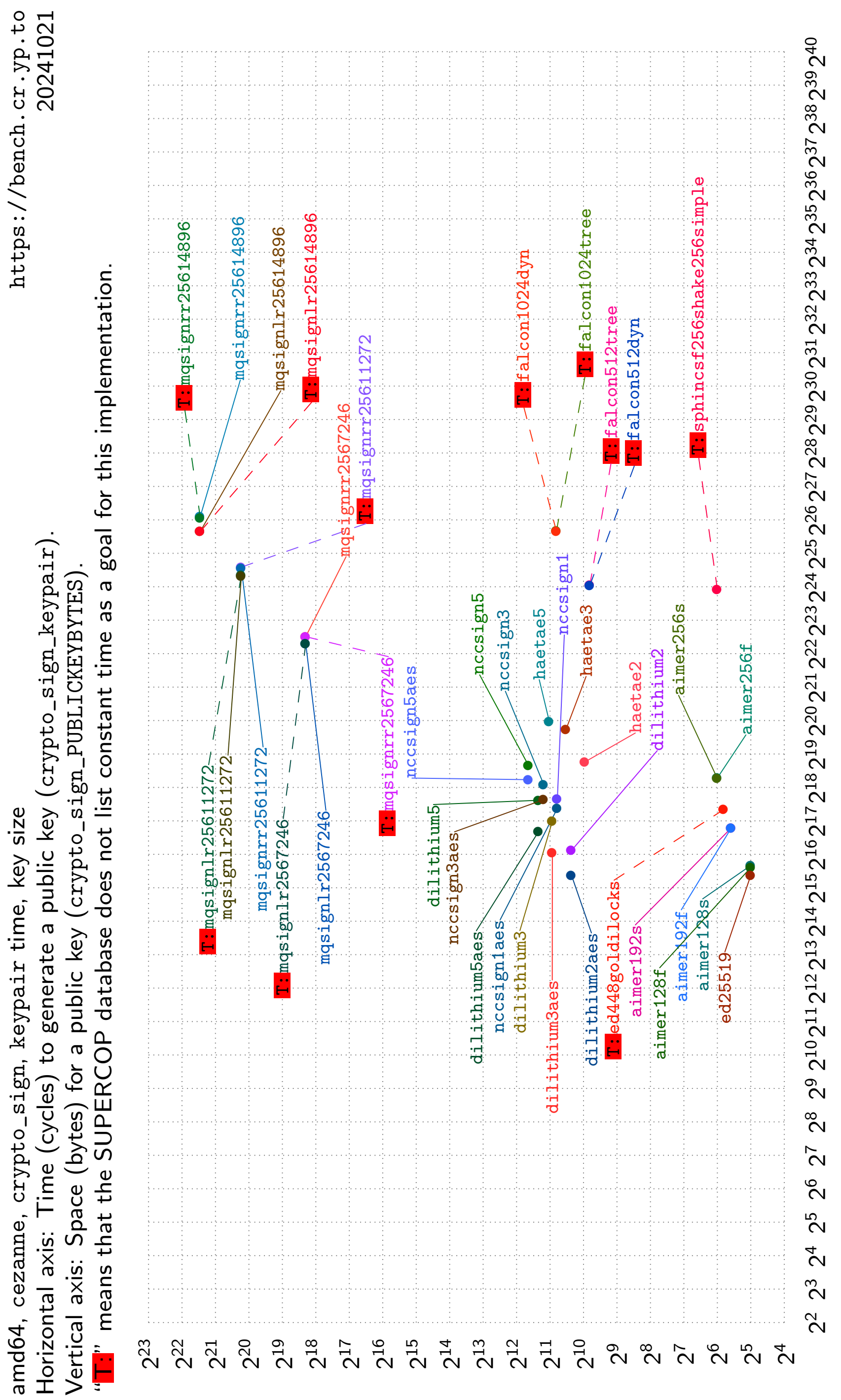


Figure 12.3: Bytes for a public key vs. cycles for key generation. Measurements collected on `cezanne`, which has an AMD Zen 3 CPU.

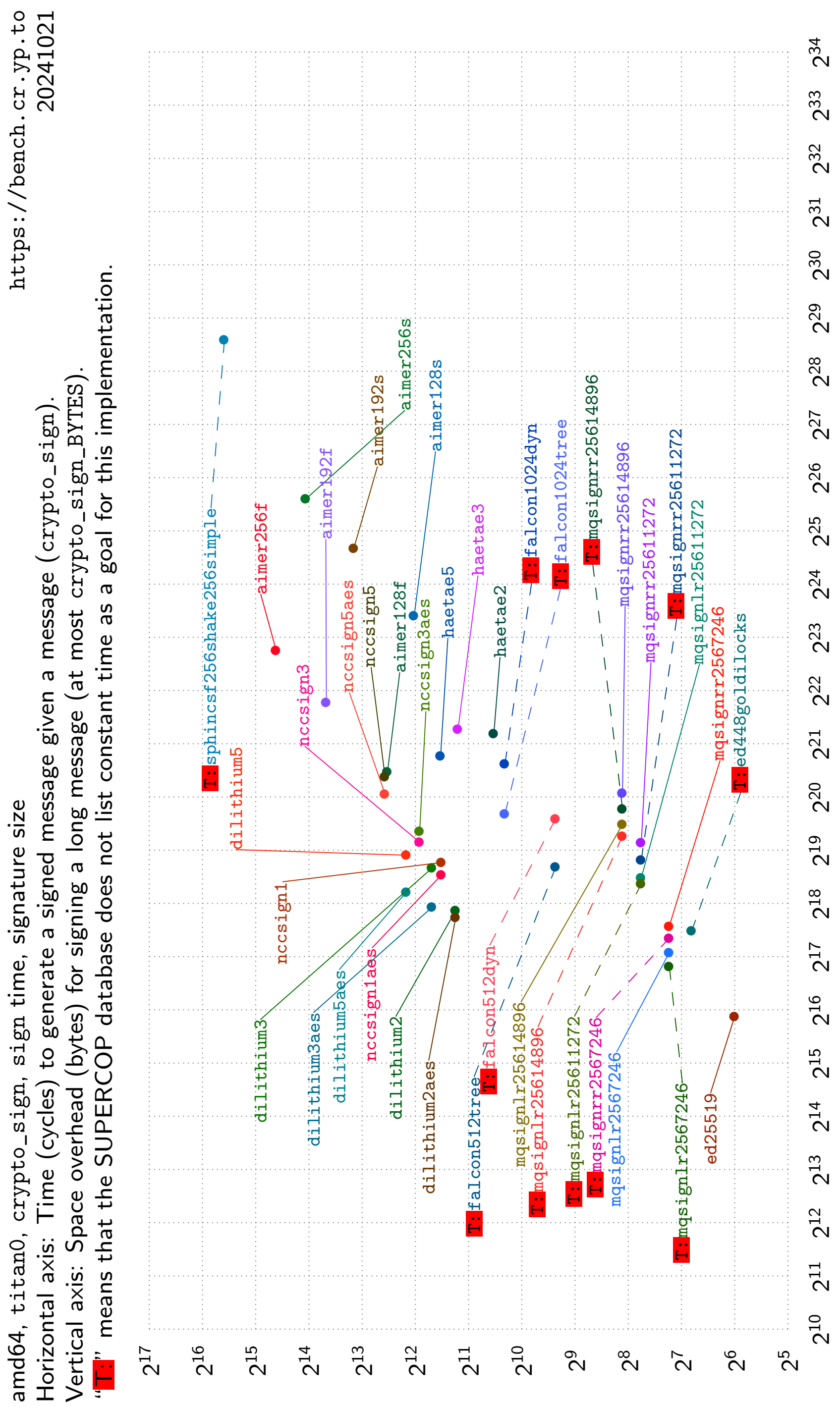


Figure 12.4: Bytes for a signature vs. cycles for signing. Measurements collected on `titan0`, which has an Intel Haswell CPU.

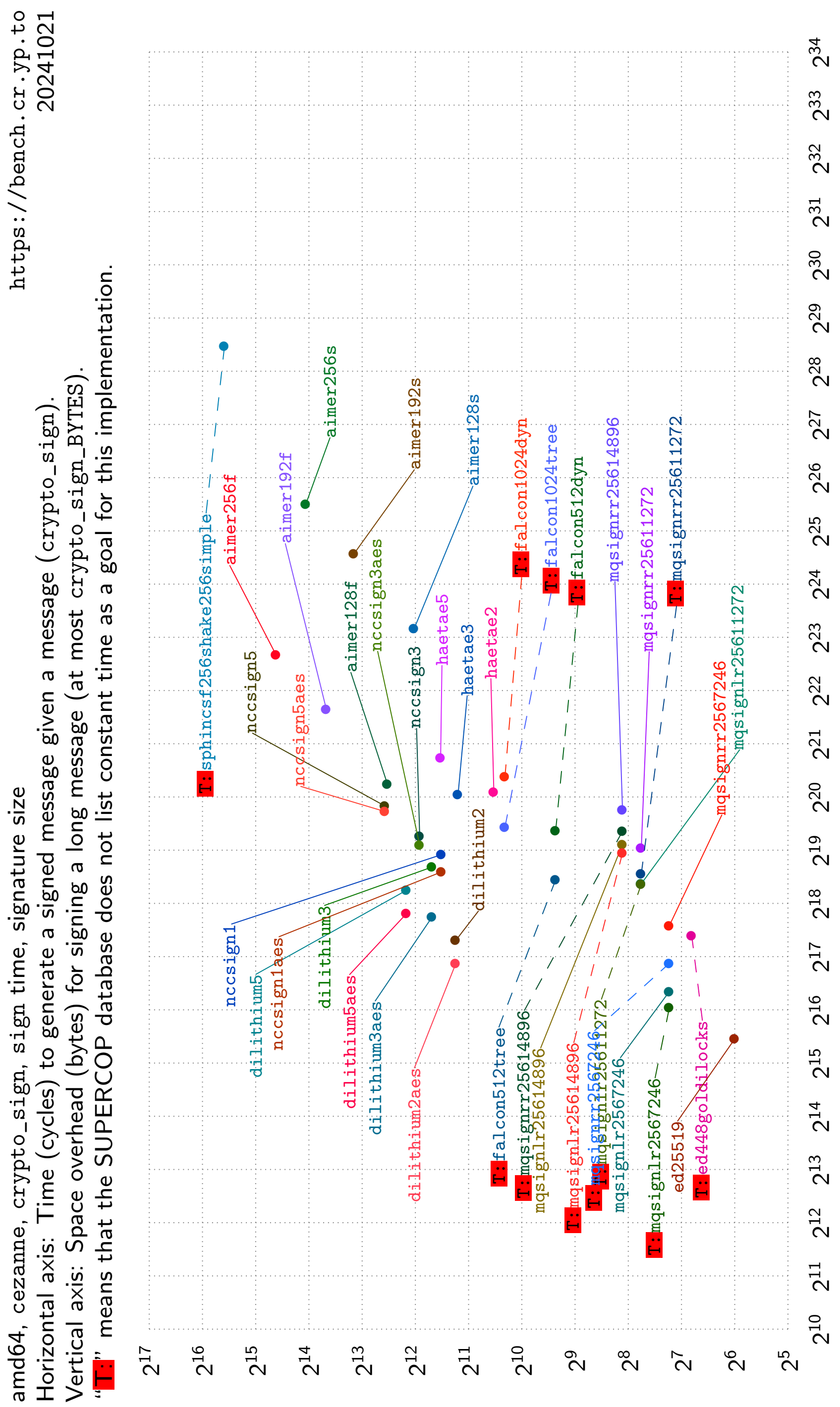


Figure 12.5: Bytes for a signature vs. cycles for signing. Measurements collected on `cezanne`, which has an AMD Zen 3 CPU.

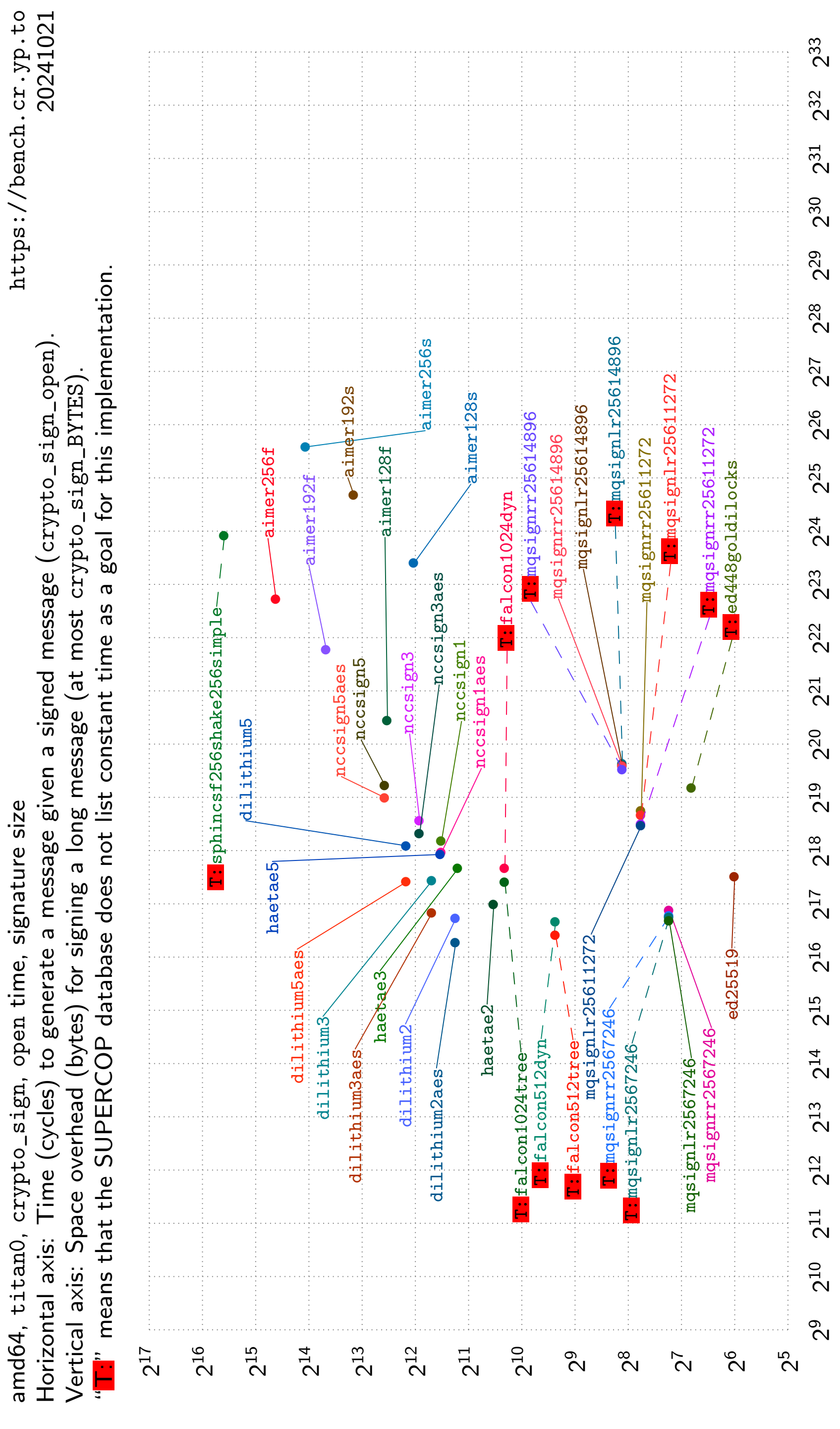


Figure 12.6: Bytes for a signature vs. cycles for verification. Measurements collected on `titan0`, which has an Intel Haswell CPU.

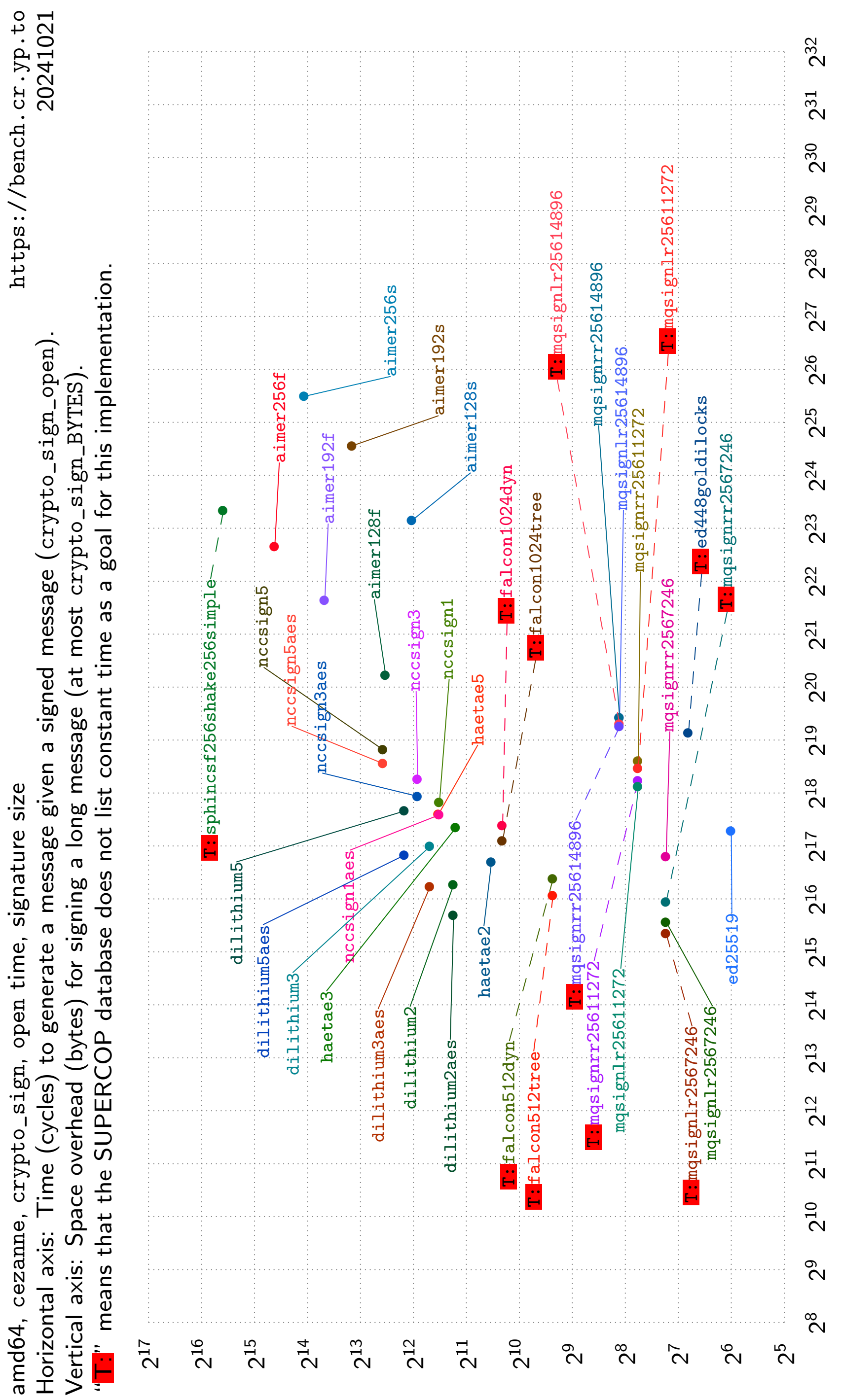


Figure 12.7: Bytes for a signature vs. cycles for verification. Measurements collected on `cezanne`, which has an AMD Zen 3 CPU.

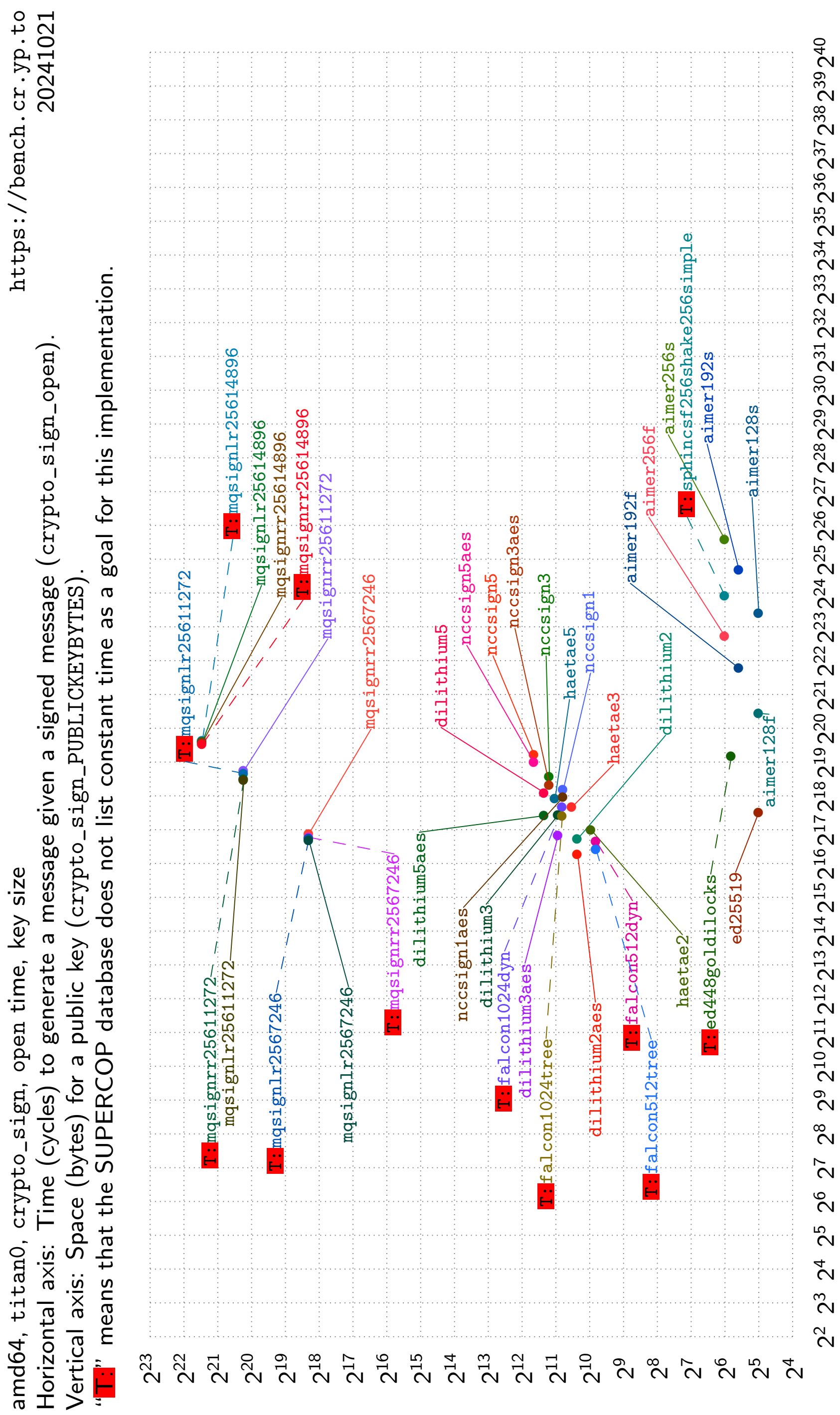


Figure 12.8: Bytes for a public key vs. cycles for verification. Measurements collected on titan0, which has an Intel Haswell CPU.

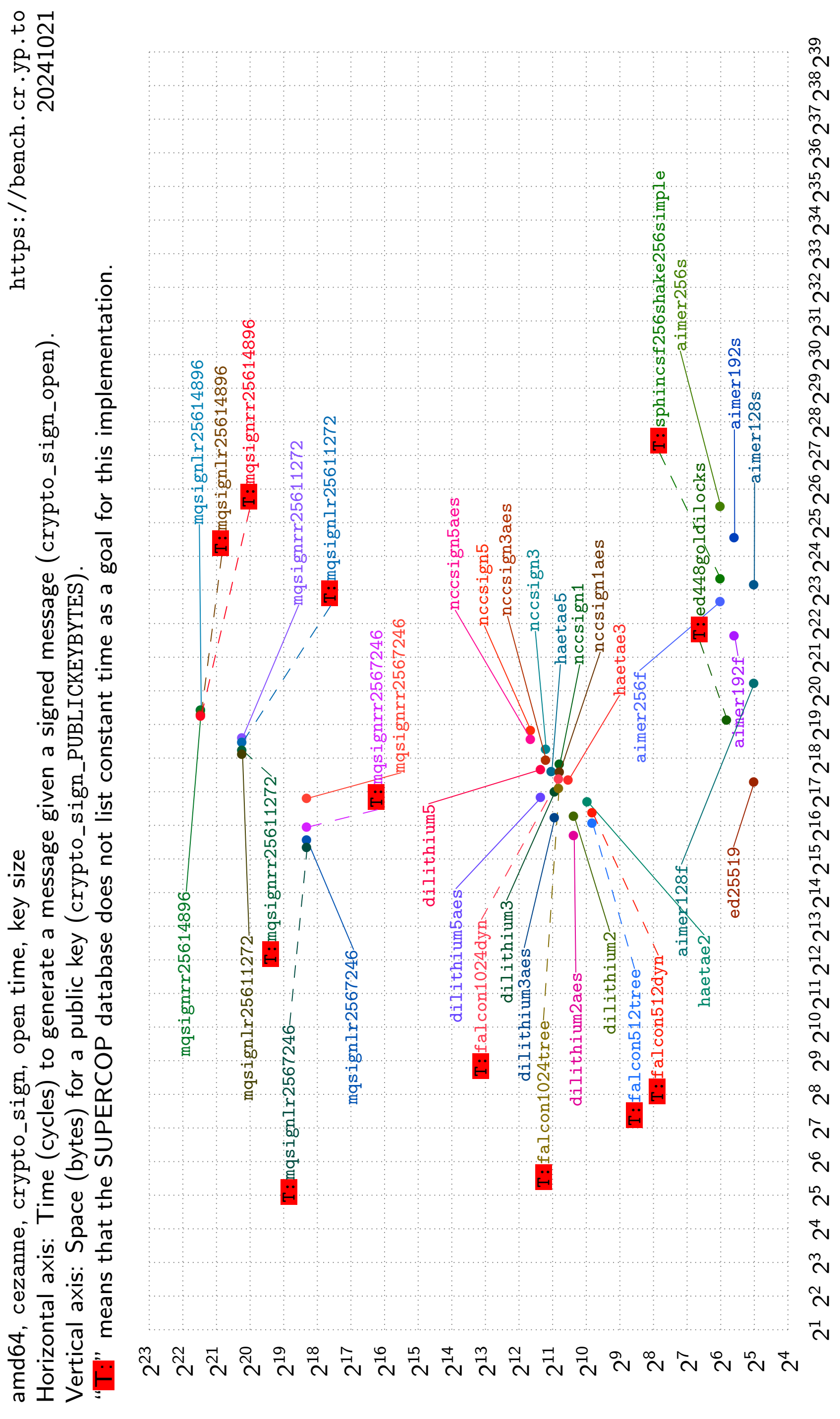


Figure 12.9: Bytes for a public key vs. cycles for verification. Measurements collected on `cezanne`, which has an AMD Zen 3 CPU.

Part III

Patents

Chapter 13

Introduction to patents

13.1 The concept of a patent

A patent is a government-issued monopoly on an "invention". Use of the "invention" without authorization from the patent holder is unlawful.
This document uses the United States as a running example (although the reader is cautioned that there are some variations in patent laws across countries). United States law says that anyone who "without authority makes, uses, offers to sell, or sells any patented invention, within the United States or imports into the United States any patented invention during the term of the patent therefor, infringes the patent".[1]

13.2 Penalties for patent infringement

If a United States court decides that a patent has been infringed, it forces the infringer to pay the patent holder "damages adequate to compensate for the infringement",[2] optionally times a factor 3. For example, if a court decides that an infringement reduced the patent holder's income by 10 million USD, the court will force the infringer to pay the patent holder between 10 million and 30 million USD.
It does not matter whether the infringer received any income. Free software, such as a cryptographic library or a web browser, is *not* exempt from patent law.

[1] 35 U.S.C. §271(a).
[2] 35 U.S.C. §284.

13.3 The doctrine of equivalents

The "inventions" covered by a patent are described by a series of "claims" in the patent. Each claim lists one "invention", and the patent is infringed if any claim is infringed.
For example, claim 1 of the RSA patent (filed in 1977, expired in 2000) is as follows (occasional italics and "1 cm" as in original):

> A cryptographic communications system comprising:
>
> - A. a communications channel,
> - B. an encoding means coupled to said channel and adapted for transforming a transmit message word signal M to a ciphertext word signal C and for transmitting C on said channel,
>   - where M corresponds to a number representative of a message and
> $$0 \leq \mathsf{M} \leq \mathsf{n}-1$$
>   - where n is a composite number of the form
> $$n = p \cdot q$$
>   - where p and q are prime numbers, and
>   - where C corresponds to a number representative of an enciphered form of said message and corresponds to
> $$\mathsf{C} \equiv \mathsf{M}^e (\mathsf{mod}\ n)$$
>   - where e is a number relatively prime to 1 cm(p-1,q-1), and
> - C. a decoding means coupled to said channel and adapted for receiving C from said channel and for transforming C to a receive message word signal M′
>   - where M′ corresponds to a number representative of a deciphered form of C and corresponds to
> $$\mathsf{M}' \equiv \mathsf{C}^d (\mathsf{mod}\ n)$$
>   - where d is a multiplicative inverse of e(mod(1 cm((p-1),(q-1)))).

To evaluate an accusation of infringement, a court begins by holding hearings (called "*Markman* hearings" in the United States) to decide what the words in the claim mean. The court then asks whether the infringing use includes each element of the claim **or something "equivalent"**:

- The "doctrine of equivalents" states [Uni02, page 732] that a patent "is not limited to its literal terms but instead embraces all equivalents to the claims described".
- The patent holder can show infringement of a patent claim under this doctrine by showing for each element of the claim that "the accused product performs substantially the same function in substantially the same way with substantially the same result" [Uni09, page 1312].

Consider, for example, the RSA patent. There is descriptive text separate from the claims, and this text mentions that "the present invention may use a modulus n which is a product of three or more primes (not necessarily distinct)". Meanwhile claim 1 of the patent, quoted above, explicitly requires n to be pq where p and q are prime numbers. Scientists tend to think that a patent covers only what is literally included in the patent's claims, so simply taking $n = 3pq$ avoids infringement of the RSA patent; i.e., requiring $n = pq$ was a foolish mistake by whoever wrote the claims in the patent. But that is not how patent law works. RSA with $n = 3pq$ is performing substantially the same function as claim 1 in substantially the same way with substantially the same result, so it infringes the claim.

Ensnarement. In the United States, accused infringers sometimes raise an "ensnarement" defense. This defense says that "substantially" would cover the prior art, i.e., publications before the patent. Courts then require the patent holder to define the boundaries of "substantially" by stating an expanded "hypothetical" claim that literally covers the infringement without covering those publications. It is important to realize that this hypothetical claim is written *by the patent holder* [Uni00], not by the accused infringer: "Under a hypothetical claim analysis, a patentee proposes a hypothetical claim that is sufficiently broad in scope to literally encompass the accused product or process. ... If that claim would have been allowed by the PTO over the prior art, then the prior art does not bar the application of the doctrine of equivalents." In other words, the patent holder comes up with *some* dividing line having the following three properties:

- the prior art is on one side of the line;
- what the accused infringer did is on the other side of the line;
- the claim is also on the other side of the line.

The infringer then has to argue that crossing this line was obvious.

13.4 Basic limits on patents

Historically, patent law is based on the belief that patents promote progress. For example, the constitution of the United States gives the legislature the power to "promote the Progress of Science and useful Arts, by securing for limited Times to Authors and Inventors the exclusive Right to their respective Writings and Discoveries". There are some corresponding limits on patents:

- Patents are issued only to "inventors". For example, in the United States, a patent can be issued only to someone who "invents or discovers any new and useful process, machine, manufacture, or composition of matter, or any new and useful improvement thereof".
- Patents have time limits. Typically patents expire 20 years after the "priority date" (the date when the "inventor" filed a patent application), although the United States often adds "patent term adjustments" to this 20-year period to account for delays by the patent office.

Showing that a patent did not promote progress—for example, showing that the same "invention" was independently published a day after the patent application was filed—does *not* invalidate the patent. The following quote from [Lem12] illustrates why this is important: "Surveys of hundreds of significant new technologies show that almost all of them are invented simultaneously or nearly simultaneously by two or more teams working independently of each other."

13.5 Novelty

In the United States, a patent claim is not permitted if "the claimed invention was patented, described in a printed publication, or in public use, on sale, or otherwise available to the public before the effective filing date of the claimed invention",[3] or if the claimed invention was described in a patent filed earlier.[4]
In other words, showing that someone else[5] published an "invention" *before* the patent application was filed invalidates the patent. The patent application is not on something "new".

[3] 35 U.S.C. §102(a)(1).
[4] 35 U.S.C. §102(a)(2).
[5] An "inventor" can publish something and then file a patent application up to a year later, so the patent effectively covers 21 years. In most countries, the publication has to be preceded by a "provisional patent application".

It is important to understand that, to eliminate a patent claim as not being new, courts need to see *one* reference that discloses *every* element of a patent claim. See, e.g., [Uni87, page 631]:

> A claim is anticipated only if each and every element as set forth in the claim is found, either expressly or inherently described, in a single prior art reference.

Providing two references, where one reference has some of the elements of the claim and the other reference has the other elements, is not good enough.

13.6 Specializations

Imagine, after the RSA patent, a new patent application being filed with claim X that looks just like claim 1 of the RSA patent but adds an extra element: "p and q are 1 modulo 2^{64}".

A court asking whether claim X is new will ask whether there is a single prior publication that is an example of *every* element of X. The RSA patent itself has *most* elements, but doesn't say that p and q are 1 modulo 2^{64}. Unless there's some other publication that has *every* element, the court will conclude that X is new. In other words, a patent does not prevent subsequent patents on specializations.

Granting a patent on X means that anyone using RSA with primes p and q chosen to be 1 modulo 2^{64} will be infringing the new patent—*and* will be infringing the RSA patent. This type of overlap happens all the time; patents are *not* required to be disjoint.

As a simpler example, the original Hellman–Diffie–Merkle patent on public-key cryptography didn't prevent the subsequent RSA patent. RSA is an example of public-key cryptography, so anyone using RSA needed permission from both patent holders.

13.7 Unobviousness

Beyond being new, patents have to be unobvious. "A patent for a claimed invention may not be obtained" if "the differences between the claimed invention and the prior art are such that the claimed invention as a whole would have been obvious before the effective filing date of the claimed invention to a person having ordinary skill in the art to which the claimed invention pertains."[6]

[6] 35 U.S.C. §103.

An obviousness analysis can look at multiple prior-art references. However, the question of what is obvious to someone of "ordinary skill in the art" is decided by courts. The people making these decisions—judges and, for resolving disputes about the facts in the United States, juries—generally have no training in the area, and are always faced with a battle between expert witnesses for one side saying something was obvious at the time of the patent application and expert witnesses for the other side saying it wasn't.
For many years, United States courts were also requiring claims of obviousness to meet a restrictive "TSM test", demonstrating that "some motivation or suggestion to combine the prior art teachings" can be found in "the prior art, the nature of the problem, or the knowledge of a person having ordinary skill in the art". However, this was rejected by a unanimous Supreme Court decision in 2007 [Uni07], which held that something "obvious to try" can qualify as obvious:

> When there is a design need or market pressure to solve a problem and there are a finite number of identified, predictable solutions, a person of ordinary skill has good reason to pursue the known options within his or her technical grasp. If this leads to the anticipated success, it is likely the product not of innovation but of ordinary skill and common sense. In that instance the fact that a combination was obvious to try might show that it was obvious under §103.

13.8 Algorithm patents

It is not clear whether algorithm patents, and more broadly software patents, are valid in the United States. A unanimous 2014 decision by the Supreme Court [Uni14] invalidated a patent on "a computer-implemented scheme for mitigating 'settlement risk' (i.e., the risk that only one party to a financial transaction will pay what it owes) by using a third-party intermediary". The decision held that "the mere recitation of a generic computer cannot transform a patent-ineligible abstract idea into a patent-eligible invention".
As an example of applying the principles from that court case, consider US patent 9912479, which claims a particular code-based key-encapsulation algorithm plus "providing the ciphertext, the confirmation value, and the encrypted message for transmission between nodes in a communication network". This is reciting a generic network, just like reciting a generic computer; it does not transform a patent-ineligible algorithm into a patent-eligible invention.

There are many examples of court cases after 2014 invalidating software patents on this basis. Consider, e.g., [Uni23], where a court invalidated a patent on an improved method of "determining the location of a mobile device by comparing previously-gathered calibration data with observed data that has been modified". The questions of novelty and unobviousness did not matter; the critical point for invalidating the patent was simply that this was "modifying or manipulating data" using "conventional computer technology".
US patent 9912479 was filed in 2017, and was initially rejected by the patent office. *However*, the patent examiner wrote "It is recommended to incorporate an explicit recitation of hardware into the claimed to overcome this portion of the rejection". The applicant did this, and the patent office issued the patent.[7] This is obviously subverting the Supreme Court's 2014 decision, but the only way to enforce the decision is with an expensive court case. Furthermore, companies cannot be confident of success in court: [Sal19] reported that challenges to software patents under the 2014 decision succeeded only about half the time.

[7]The patent office receives more money for issuing a patent than for rejecting it.

Chapter 14

Specific post-quantum patents

14.1 Warnings

The following sections list various patents that appear potentially relevant to post-quantum cryptography. This list is not comprehensive: a search on Google Patents indicates more than 100000 search results for cryptography, and more than 4000 search results for "post-quantum".
The sections are organized by priority date. Publications *before* that date are relevant to analyses of whether the claimed "inventions" are new and unobvious.
Patents are typically filed in multiple countries, often with different claims, and are often further split into "divisional" patents. Specific patent numbers are provided here to identify the patent families. Sometimes multiple countries are listed here, but this has not been done systematically.
The patent analyses in this document are preliminary and are not legal advice. Comments on prior art are not comprehensive.

14.2 25 July 2000: pqNTRUSign

Application WO 2002009348 A3. Was listed as applicable by the pqNTRUSign team in a NIST filing.

Abandonment. The patent application appears to have been abandoned long before the NIST submission.

14.3 7 December 2001: FALCON

US patent 7308097. 1-year provisional period and 889-day patent-term adjustment, so expires in May 2025. Listed as applicable by the FALCON team in a NIST filing.

14.4 7 December 2001: pqNTRUSign

US patent 7913088. Listed as applicable by the pqNTRUSign team in a NIST filing.

Expiration. This patent had a 1-year provisional period and a 654-day patent-term adjustment, so it expired in September 2024.

14.5 11 April 2002: Gui and Rainbow

US patent 7158636. Ding. 1-year provisional period and 704-day patent-term adjustment, so expires in May 2025. Listed as applicable by the Gui and Rainbow teams in a NIST filing.

Expiration. The patent has already expired because of non-payment of fees.

14.6 24 April 2003: NTRU without decryption failures

US patent 7929688. Yamamichi, Futa, Ohmori, Tatebayashi (Panasonic). Expires 23 July 2028 in the United States; has already expired in all other countries.
Roughly: Claims generating and using NTRU parameters "causing no decryption error", using the condition "$2 \cdot p \cdot d + 2df - 1 < q/2$" where df is "the number of coefficients in a private key polynomial f whose coefficient values equal to 1". This might be stretched to cover similar formulas to eliminate decryption failures in other variants of NTRU, so it's a potential problem for the 2005 NTRU parameter sets and most newer variants of NTRU.
Given the doctrine of equivalents, a patent on NTRU without decryption failures can cover a version of NTRU for which decryption failures are so

unlikely as to never be observed. The patent might also be stretched to lattice-based systems beyond NTRU.

Prior art 1. Hoffstein, Pipher, and Silverman handed out a preprint "NTRU: a new high speed public key cryptosystem" in 1996, in particular at Crypto 1996. Section 4.3 of this draft (page 18) says "NTRU with 0% decoding failure. It is possible to eliminate gap failure entirely by choosing the parameter q sufficiently large. . . . a trivial analysis shows that the coefficient range is less than $d^2 + 2dp$ for binary NTRU and less than $r^5d^2 + r^2dp$ for symmetric NTRU. So if we choose q larger than this bound, gap failure disappears."
Handing out documents at a conference open to the public (even if there are registration fees), without confidentiality restrictions, should count as prior art under United States patent law. Conference handouts were treated as prior art in, e.g., [Uni85]:

> We agree with the ITC's conclusion that the Birmingham paper is prior art. As the Commission noted, between 50 and 500 persons interested and of ordinary skill in the subject matter were actually told of the existence of the paper and informed of its contents by the oral presentation, and the document itself was actually disseminated without restriction to at least six persons.

In [Uni04], a three-day conference-poster display—never handed out—was treated as prior art, with the court saying that it was "considering and balancing" the following factors:

> the length of time the display was exhibited, the expertise of the target audience, the existence (or lack thereof) of reasonable expectations that the material displayed would not be copied, and the simplicity or ease with which the material displayed could have been copied.

Prior art 2. In 2000, Jaulmes and Joux [JJ00] made the following statement about NTRU: "How Decryption Works. . . . For appropriate parameter choices, we can ensure that all coefficients of the polynomial . . . lie between $-q/2$ and $q/2$. So the intermediate value . . . is in fact the true (non modular) value of this polynomial. This means that when we compute a and reduce its coefficients into this interval, we recover exactly the polynomial . . . Hence . . . retrieves the message m."
The patent description says that "while the existing technique presents conditions for generating NTRU parameters that do not cause any decryption

errors, such conditions are not formulated, which makes it difficult to generate NTRU parameters that do not cause any decryption errors". In other words, the patent holder is asserting that it isn't obvious *how* to write down what Jaulmes and Joux call "appropriate parameter choices". Meanwhile the 1996 NTRU draft correctly called this "trivial" and gave an example of the conditions.

14.7 3 November 2003: decryption using pairings

US patent 7499544. 959-day patent-term adjustment, so expires in 2026. Mentioned by the SIKE team in a NIST filing. Official statements from the SIKE team do not claim that this patent covers SIKE. SIKE is broken, but the same patent seems to apply to some other isogeny-based systems.

14.8 11 January 2005: Gui and Rainbow

US patent 7961876. Ding. Nearly 1-year provisional period and 1335-day patent-term adjustment, so will expire in August 2029. Listed as applicable by the Gui and Rainbow teams in a NIST filing.

14.9 8 June 2005: WalnutDSA

US patent 7649999. 772-day patent-term adjustment. Listed as applicable in a NIST filing by the WalnutDSA team. WalnutDSA is broken.

14.10 8 June 2005: WalnutDSA

US patent 9071427. 0-day patent-term adjustment. Listed as applicable in a NIST filing by the WalnutDSA team. WalnutDSA is broken.

14.11 18 February 2010: the GAM/LPR family of cryptosystems (noisy DH + reconciliation)

Patents EP 2537284, US 9094189, FR 10/51190. Gaborit and Aguilar Melchor. The patent holder, CNRS, also paid fees for DE, GB, CH. Expires January 2032.
Listed as applicable in a NIST filing for various small-key code-based cryptosystems (BIKE, HQC, RQC, and Ouroboros). Apparently also applicable to various small-key lattice-based cryptosystems.
This is a very broad patent on encryption via the noisy Diffie–Hellman system plus reconciliation. This is typically called the "LPR10" cryptosystem and credited to a 2010 paper [LPR10a] by Lyubashevsky, Peikert, and Regev, but LPR published that cryptosystem *after* the 18 February 2010 priority date for this patent.
The original version of the LPR paper had a deadline in late February 2010 for Eurocrypt, was published a few months later by Springer, and had a more complicated cryptosystem with larger keys. The simple "LPR10" system (with small keys, noisy DH, and reconciliation) appeared in, e.g., the Eurocrypt 2010 slides [LPR10b] from LPR in May 2010. Peikert had posted slides in 2009 on noisy DH, but those slides did not have reconciliation.
In 2017, on behalf of an undeclared client, a British law firm named Keltie filed an opposition to the European version of this patent. Declared consultants for Keltie included Peikert (who wrote "In the likely event that the court accepts this argument, the patent is invalidated" in 2020) and two GCHQ cryptographers. Keltie's opposition was rejected. Keltie appealed. The appeal board issued a preliminary assessment in 2021, not in Keltie's favor. Keltie withdrew the appeal on 20 October 2021. `https://register.epo.org/application?number=EP11712927&lng=en&tab=doclist` shows the full history of documents filed in the dispute.
Keltie's rejected arguments included the following:

- One general requirement in patent law is for patents to claim "inventions" that actually work. Keltie argued that the GAM claim doesn't work, since the specific formulas given fail unless the underlying ring is commutative. The patent holders responded that the formulas work for various commutative rings given as examples in the patent description, and that anyone skilled in the art can figure out that the commutative case makes the formulas work, while (obviously) the formulas require adjustment to handle the non-commutative case.

- Keltie argued that the patented "invention" wasn't novel. However, the first publications of the GAM/LPR cryptosystem (by GAM, and independently by LPR) were after the GAM patent was filed. For each prior publication provided by Keltie, it is easy to see that the publication is missing some element of the GAM claim.

- Keltie argued that the patented "invention" was obvious. Peikert filed a statement under oath stating that someone skilled in the art "and wishing to improve the efficiency of the described lattice/LWE-based cryptographic system, would have arrived at the invention set out in the claims of the patent". For comparison, [Pei16] says that the LPR10 cryptosystem "was first described in [LPR10]". The Eurocrypt 2010 slides from LPR [LPR10b], after presenting a "suitable 'compact' version of LWE called Ring-LWE", used that to produce "an entirely new & even more efficient PKE scheme"—"a new kind of LWE cryptosystem", namely LPR10.

It is important to realize that all of these proceedings were regarding the *validity* of the claims in the patent. Recall from Section 13.3 that, when a court considers *infringement* of a patent claim, the court does not ask merely whether the claim is literally infringed; doing substantially the same as the patented invention is enough.
Hoping to avoid the GAM patent by adding some non-commutativity, as in Kyber, Saber, etc., is like hoping to avoid the RSA patent by using $3pq$ instead of pq: this avoids the literal meaning of the claim, but is doing substantially the same thing as the claim. An ensnarement defense (see Section 13.3) based on LWE will not work: it is easy to formulate an edited claim that uses efficiency to cover RLWE and typical examples of MLWE while excluding LWE.
There have been some public statements dismissing the risk from this patent. These statements are generally missing how patent law works. For example, it is asserted in [LS21] that the "patent applicability claim to Kyber and Saber is baseless". No references to patent law appear in [LS21], and the main arguments in [LS21] erroneously conflate the question of infringement with the question of literal coverage. A separate argument in [LS21] quotes the statement "The owner indicates that in no case would the patent as granted prevent the protection of a development based on non-commutative rings"; this is saying that the GAM patent does not prevent a followup patent regarding non-commutative rings (see Section 13.6), but is misinterpreted in [LS21] as a statement about what would infringe this patent.

Partial licensing. NIST reportedly purchased a license for this patent family *specifically for Kyber*.

14.12 22 April 2010: McEliece variant

US patent 8958553, GB patent 2469393. "This invention provides improved security of the McEliece Public Key encryption system adding features which make full use of random number generation for given message and cryptogram parameters, using this invention the encrypted message (i.e. the cryptogram) is a truly random function, not a pseudo random function of the message so that even with the same message and the same public key, a different, unpredictable cryptogram is produced each time."
This is one of the basic patents on "NTS-KEM", a variant of McEliece; "NTS" stands for "never the same".

Abandonment. The patent holders agreed to abandon these patents as a condition of being allowed to merge NTS-KEM with Classic McEliece. Google Patents now lists the patents as expired for non-payment of fees by the patent holder.

14.13 16 November 2010: McEliece variant

US patent 8891763, GB patent 2473154. "This invention provides improved security and improved throughput of the McEliece public key encryption system and reduces the public key size. . . . It is possible using this invention that the encrypted message, the cryptogram is a truly random function, not a pseudo random function of the message so that even with the same message and the same public key, a different, unpredictable cryptogram is produced each time."

Abandonment. The patent holders agreed to abandon these patents as a condition of being allowed to merge NTS-KEM with Classic McEliece. Google Patents now lists the patents as expired for non-payment of fees by the patent holder.

14.14 9 February 2011: secret-sharing signatures

JP 5736816, US 8522033, US 8959355, CN ZL201110145023.8. Sakumoto, Shirai, Hiwatari (Sony). Listed as applicable in a NIST filing by the MQDSS team. Expires 2031 or 2032.

Many claims are on authentication, but there is no real dividing line between authentication and signatures, and claim 18 of US patent 8522033 is explicitly on signatures. Reading claim 18 finds many words that do not really limit which signature systems are covered. For example:

- "Signature generation device": This covers any computer generating signatures.
- "Processor": This is the CPU in the computer.
- "Key setting unit": This is the part of the computer generating a public key.
- Secret key "$s \in K^n$" for a "ring K": Any secret key can be viewed as a string of bits, with $K = \mathbb{F}_2$.
- A "multi-order multi-variable polynomial $f_i(x_1, \ldots, x_n)$": Any finite computation can be viewed as evaluating a polynomial; there is no requirement of the polynomial being, e.g., a random quadratic polynomial.
- A "message generation unit for generating N messages c based on the multi-order multi-variable polynomial $f_i(x_1, \ldots, x_n)$ and the secret key s": Any signing computation will generate data from the secret key.
- A "verification pattern" obtained by "applying a document M and the message c to a one-way hash function": Signature systems typically hash something together with the document being signed.

Claim 18 has one restriction that sounds more specific: "the digital signature σ is information that enables calculation of the secret key s in a case verifications performed using the digital signature σ corresponding to $(k-1)^N+1$ verification patterns have all been successful."

It is not clear how restrictive this actually is, especially given the doctrine of equivalents (see Section 13.3). It seems possible to cover, e.g., Schnorr signatures by choosing $k = 2$ and $N = 1$ and interpreting the words reasonably. This would also invalidate this patent claim (since Schnorr signatures

are older), so the patent holder would argue for a narrower interpretation of the words.
It seems possible that the patent holder could interpret words so as to cover any MPC-in-the-head signature scheme. MPC-in-the-head *zero-knowledge proofs* appeared in 2007 [IKOS07], but [IKOS07] did not mention signatures, so stopping a claim interpreted in this way would require convincing a court that MPC-in-the-head signatures were obvious given (1) MPC-in-the-head zero-knowledge proofs and (2) standard transformations from zero-knowledge proofs to signatures.

14.15 12 April 2012: compressing GAM/LPR ciphertexts

US patent 9246675. Ding. 1-year provisional period and 0-day patent-term adjustment, so expires 11 April 2033. Roughly, this patent claims noisy DH with compressed reconciliation data. This produces shorter ciphertexts than GAM/LPR.
A paper by Peikert said it was introducing noisy DH with compressed reconciliation data:

> One of our main technical innovations (which may be of independent interest) is a simple, low-bandwidth *reconciliation* technique that allows two parties who "approximately agree" on a secret value to reach *exact* agreement, a setting common to essentially all lattice-based encryption schemes. Our technique reduces the ciphertext length of prior (already compact) encryption schemes nearly twofold, at essentially no cost.

However, this paper was from 2014 [Pei14], two years after this patent was filed. The patent seems very likely to be upheld in court if there is any litigation.
In July 2016, Google rolled out an experiment with New Hope, saying "we plan to discontinue this experiment within two years, hopefully by replacing it with something better". The patent holder reportedly contacted Google to ask for money. In November 2016, Google announced that it was turning off the experiment.
In December 2016, a paper "NewHope without reconciliation" [ADPS16] said that it "avoids the error-reconciliation mechanism originally proposed by Ding". This led to a perception that the cryptosystem avoids Ding's patent.

Various cryptosystems such as Kyber were based on "NewHope without reconciliation". However, it is unclear why the "v" quantity in "NewHope without reconciliation" does not qualify as "reconciliation". Even if it is possible to articulate a clear technical difference, "NewHope without reconciliation" and its successors perform substantially the same function as Ding's patented system in substantially the same way with substantially the same result.

Partial licensing. As in the case of the GAM patent discussed above, NIST reportedly purchased a license for this patent family *specifically for Kyber.*

14.16 13 November 2014: McEliece variant

GB patent 2532242. "This invention is concerned with providing additional features to the original McEliece system which enhance the bandwidth efficiency and security of the Public Key Encryption arrangement, and to applications thereof."

Abandonment. The patent holders agreed to abandon this patent as a condition of being allowed to merge NTS-KEM with Classic McEliece. Google Patents now lists the patent as expired for non-payment of fees by the patent holder.

14.17 5 January 2015: pqNTRUSign

US patent 9722798. Expires in 2035. Listed (20150229478) as applicable in a NIST filing for pqNTRUSign.

14.18 30 March 2015: Gui and Rainbow

US patent 11290273. Ding. Expires in 2036. Listed as applicable (under the application number, US 15/562034) by the Gui and Rainbow teams in a NIST filing.

14.19 8 August 2016: multiple signing

US patent 9794249 by Truskovsky, Yamada, Brown, and Gutoski (ISARA). Claims combining multiple signature systems.

Prior art. [Ber16] (February 2016). The general idea of combining cryptosystems is much older than [Ber16]. The specific idea of combining pre-quantum and post-quantum cryptosystems is probably much older than this too.

14.20 15 September 2016: randomizing public lattice parameters

US patent 9942040 by Kalach (ISARA). Claim 1: "A lattice-based cryptography method comprising: obtaining a first value of a public parameter previously used in a first execution of a lattice-based cryptography protocol; generating, by operation of one or more processors, a second value of the public parameter based on the first value of the public parameter and random information; and using the second value of the public parameter in a second execution of the lattice-based cryptography protocol, wherein the second execution of the lattice-based cryptography protocol comprises: generating a public key based on the second value of the public parameter; and sending the public key to a correspondent over a channel."

14.21 7 November 2016: isogeny-based key exchange

US patent 10673631. This appears to be a rather broad patent on isogeny-based key exchange, although it cannot cover systems published earlier, such as CRS.

14.22 18 November 2016: GAM/LPR with rounding

US patent 11329799. Expires in 2037. Listed (PCT/KR2017/013119, WO2018093203A1) as applicable in a NIST filing by the Lizard team.
Roughly, the patent claims schemes where the public key uses addition of errors (as in GAM/LPR) but the ciphertext uses rounding. A question of whether the patent also covers, e.g., SABER has not been answered.

Prior art. Rounded ciphertexts already appeared in [BCLv17, May 2016 version].

14.23 18 November 2016: GAM/LPR variants

China patent 107566121. Yunlei Zhao. Expires in 2036. Listed as applicable in a NIST filing for KCL (OKCN, AKCN, CNKE).
This patent sounds very similar to “NewHope without reconciliation” and was filed a month before “NewHope without reconciliation” was published. The patent holder stated in May 2022 (https://groups.google.com/a/list.nist.gov/g/pqc-forum/c/Fm4cDfsx65s/m/F63mixuWBAAJ) that “Kyber is covered by our patents”. There has been no statement from NIST regarding this.

14.24 15 February 2017: GAM/LPR with rounding

US patent 11070367. Bhattacharya, Garcia Morchon, Tolhuizen, Rietman. Listed (EP17156214, EP17170508, EP17159296, EP17196812, EP17196926) as applicable in a NIST filing for Round5.

14.25 10 May 2017: GAM/LPR with rounding

Europe patent 3622663. Appears similar to the 15 February 2017 filing, except for using polynomials rather than matrices.

14.26 9 June 2017: QC-MDPC

US patent 9912479. ISARA. Listed as applicable in a NIST filing for QC-MDPC. Expires 2037.
This patent was discussed in Section 13.8 as an example of how the US patent office continues to allow algorithm patents.
During the application process, the patent examiner also identified prior art (namely patent application 20150163060 from Tomlinson). ISARA filed an amendment, replacing “applying an error vector derivation function to a random value to produce an error vector” with the following text:

> applying an error vector derivation function to a random value to produce an error vector, wherein applying the error vector derivation

function to the random value comprises:

- applying a pseudorandom function to the random value to produce a pseudorandom function output;
- applying a filter to the pseudorandom function output to produce a filtered pseudorandom function output, the filtered pseudorandom function output comprising t integers; and
- generating the error vector based on the filtered pseudorandom function output, the error vector having a Hamming weight equal to t[.]

ISARA argued that the amended claims did not include any prior art: "The Office Action relies on Tomlinson as allegedly disclosing a previous version of these features, but Tomlinson does not disclose an error vector derivation function that includes a pseudorandom function and a filter applied to a pseudorandom function output as claimed, and thus, does not disclose an error vector based on a filtered pseudorandom function output."
Of course, there is ample literature explaining that the job of pseudorandom functions is to simulate randomness; one can freely replace any random value with the output of a pseudorandom function. There is also ample literature explaining how to apply various filters to random strings to obtain other types of randomness, this is the standard way of obtaining, e.g., random error vectors. But the patent office issued the patent.
The original Classic McEliece submission to NIST in 2017 started with a ciphertext format from Niederreiter and added "plaintext confirmation". In 2022, Classic McEliece added software that removes "plaintext confirmation", and explained this as patent avoidance:

> The purpose of this tweak is to proactively eliminate any concerns regarding U.S. patent 9912479. The threat posed by the patent was already very low, for at least two reasons:
>
> - The patent does not literally apply to any version of Classic McEliece. For example, one of the requirements in the patent is to obtain a ciphertext from "the error vector and the plaintext value"; Classic McEliece has only one of these two objects. The decryption mechanism in the patent is even farther away from what Classic McEliece does.
> - All of the overlap between the patent and Classic McEliece is already in the prior art, such as `https://eprint.iacr.org/2015/610`.

However, users may be concerned about the risk of a court (1) declaring Classic McEliece's encapsulation "equivalent" to the patented encryption mechanism and (2) mishandling the prior art. Eliminating plaintext confirmation creates a much more clear separation between Classic McEliece and the patent, and returns Classic McEliece to the traditional Niederreiter form of encryption.

Prior art. [BCS13]; [BCLv17].

14.27 13 December 2017: RLCE

US patent application 15/840,121. Listed as applicable in a NIST filing for RLCE.

Abandonment. The patent application has been abandoned.

14.28 20 September 2016: WalnutDSA

US patent 10523440. Listed as applicable in a NIST filing by the WalnutDSA team. WalnutDSA is broken.

14.29 22 May 2017: Compact LWE

Australia patent application 2017901941. Listed as applicable in a NIST filing for Compact LWE.

Abandonment. The patent application has been abandoned.

14.30 6 February 2018: multiple signing and encryption

US patent 11716195. Claim 1: "A method comprising: generating, by a first device and based on a secret sharing algorithm, a first secret and a second secret from a first communication; encrypting, using a first encryption algorithm, the first secret; encrypting, using a second encryption algorithm, the second secret; generating a first signature of the first and second encrypted secrets; generating a second signature of the first and second encrypted secrets;

and transmitting, by the first device to a second device, the first encrypted secret, the second encrypted secret, the first signature, and the second signature."
The wording here sounds broad, for example covering TLS if encryption *and* signatures are both upgraded to post-quantum hybrids.

Prior art. [Ber16].

14.31 12 September 2018: polynomial multiplication

US patent 11798435. Claim 1: "A method for executing a cryptographic operation on a security device, the method comprising: sampling a first polynomial, wherein coefficients of the first polynomial are determined based on a first distribution such that: a value 0 of the first polynomial occurs with a probability amounting to ζ, a value $-$lim1 of the first polynomial occurs with a probability amounting to α, and a value lim1 of the first polynomial occurs with a probability amounting to β, wherein $\zeta + \alpha + \beta$=near 1; sampling a second polynomial, wherein a selection of k coefficients of the second polynomial is determined based on a second distribution; multiplying the first polynomial with the second polynomial to determine a result; and executing the cryptographic operation using the result of the multiplication, wherein the method is performed to increase the robustness of the security device against one or more side-channel attacks."

Prior art. The NTRU Prime software from [BCLv17] was already multiplying a ternary polynomial by another polynomial with protection against timing attacks.

14.32 12 February 2019: multiple encryption

US patent 11431498. Claim 1: "A method, comprising: concatenating, by a processor system of a computing device, a message to be encrypted with a random value; generating a symmetric key and a first ciphertext comprising an encapsulation of the symmetric key using a value derived from the random value using at least a first one-way function and a first public key of a first asymmetric key pair; encrypting, by the processor system, the combination of the message to be encrypted and the random value with the symmetric key to provide an intermediate ciphertext; encapsulating, by the processor system, the intermediate ciphertext using a different value derived from the random value and a second public key of a second asymmetric key pair to provide a second ciphertext, the different value being derived from the random value using at least a second one-way function; and storing the first and second ciphertexts in memory of the computing device."

Prior art. [Ber16].

14.33 28 June 2019: precomputation in hash-based signatures

US patent 11218320. Intel. Roughly: claims hash-based signatures with "accelerator logic to pre-compute at least one set of inputs to the signature logic".

Prior art. [WJW$^+$19], already online in 2018.

14.34 28 June 2019: multiple signing

US patent 11456877. Intel. Roughly: claims signing a message twice on a "unified hardware accelerator hosted by a trusted platform of the computing device".

14.35 28 June 2019: hash chains

US patent 11770258. Intel. Roughly: claims a "hardware processor" for hash chains.

14.36 28 June 2019: hash-based signatures

US patent 11770262. Intel. Roughly: claims a "hardware processor" for hash-based signatures.

14.37 28 June 2019: hash-based signatures

US patent 11917053. Intel. Roughly: claims a combined SHA-2/SHA-3 accelerator for hash-based signatures.

14.38 28 February 2020: multiple encryption

US patent 11343084. Roughly: claims double encryption with ECDH and post-quantum cryptography.

Prior art. [Ber16].

14.39 29 July 2020: multiple encryption

US patent 11722296. John A. Nix, who has licensed other patents to a patent-enforcement company "Network-1 Technologies". Roughly: claims a device that (1) announces an ephemeral KEM public key, (2) receives a ciphertext, (3) decapsulates the ciphertext, (4) extracts a public key for a different KEM, and (5) encapsulates to that public key.

Prior art. [SSW20].

Bibliography

[AAB$^+$17a] Carlos Aguilar-Melchor, Nicolas Aragon, Slim Bettaieb, Loic Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Phillipe Gaborit, Adrien Hauteville, and Gilles Zémor. Ouroboros-R. Technical report, National Institute of Standards and Technology, 2017. available at https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-1-submissions.

[AAB$^+$17b] Carlos Aguilar-Melchor, Nicolas Aragon, Slim Bettaieb, Loic Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Phillippe Gaborit, and Gilles Zémor. RQC. Technical report, National Institute of Standards and Technology, 2017. available at https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-1-submissions.

[AASA$^+$20] Gorjan Alagic, Jacob Alperin-Sheriff, Daniel Apon, David Cooper, Quynh Dang, John Kelsey, Yi-Kai Liu, Carl Miller, Dustin Moody, Rene Peralta, Ray Perlner, Angela Robinson, and Daniel Smith-Tone. Status report on the second round of the NIST post-quantum cryptography standardization process. NIST IR 8309, 2020. https://doi.org/10.6028/NIST.IR.8309.

[ABC$^+$22] Martin R. Albrecht, Daniel J. Bernstein, Tung Chou, Carlos Cid, Jan Gilcher, Tanja Lange, Varun Maram, Ingo von Maurich, Rafael Misoczki, Ruben Niederhagen, Kenneth G. Paterson, Edoardo Persichetti, Christiane Peters, Peter Schwabe, Nicolas Sendrier, Jakub Szefer, Cen Jung Tjhai, Martin Tomlinson, and Wen Wang. Classic McEliece. Technical report, National Institute of Standards and Technology, 2022. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-4-submissions.

[ABD$^+$17a] Nicolas Aragon, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, Adrien Hauteville, Olivier Ruatta, Jean-Pierre Tillich, and Gilles Zémor. LAKE. Technical report, National Institute of Standards and Technology, 2017. available at https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-1-submissions.

[ABD$^+$17b] Nicolas Aragon, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, Adrien Hauteville, Olivier Ruatta, Jean-Pierre Tillich, and Gilles Zémor. LOCKER. Technical report, National Institute of Standards and Technology, 2017. available at https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-1-submissions.

[ACD$^+$18] Martin R. Albrecht, Benjamin R. Curtis, Amit Deo, Alex Davidson, Rachel Player, Eamonn W. Postlethwaite, Fernando Virdia, and Thomas Wunderer. Estimate all the LWE, NTRU schemes! In Dario Catalano and Roberto De Prisco, editors, *SCN 18: 11th International Conference on Security in Communication Networks*, volume 11035 of *Lecture Notes in Computer Science*, pages 351–367. Springer, Cham, September 2018. doi:10.1007/978-3-319-98113-0_19.

[ACPS09] Benny Applebaum, David Cash, Chris Peikert, and Amit Sahai. Fast cryptographic primitives and circular-secure encryption based on hard learning problems. In Shai Halevi, editor, *Advances in Cryptology – CRYPTO 2009*, volume 5677 of *Lecture Notes in Computer Science*, pages 595–618. Springer, Berlin, Heidelberg, August 2009. doi:10.1007/978-3-642-03356-8_35.

[AD17] Martin R. Albrecht and Amit Deo. Large modulus ring-LWE $\geq$ module-LWE. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology – ASIACRYPT 2017, Part I*, volume 10624 of *Lecture Notes in Computer Science*, pages

267–296. Springer, Cham, December 2017. doi:10.1007/978-3-319-70694-8_10.

[ADPS16] Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe. NewHope without reconciliation. Cryptology ePrint Archive, Report 2016/1157, 2016. URL: https://eprint.iacr.org/2016/1157.

[AGH$^+$23] Carlos Aguilar-Melchor, Nicolas Gama, James Howe, Andreas Hülsing, David Joseph, and Dongze Yue. The return of the SDitH. In Carmit Hazay and Martijn Stam, editors, *Advances in Cryptology – EUROCRYPT 2023, Part V*, volume 14008 of *Lecture Notes in Computer Science*, pages 564–596. Springer, Cham, April 2023. doi:10.1007/978-3-031-30589-4_20.

[AHJ$^+$23] Carlos Aguilar Melchor, Andreas Hülsing, David Joseph, Christian Majenz, Eyal Ronen, and Dongze Yue. SDitH in the QROM. In Jian Guo and Ron Steinfeld, editors, *Advances in Cryptology – ASIACRYPT 2023, Part VII*, volume 14444 of *Lecture Notes in Computer Science*, pages 317–350. Springer, Singapore, December 2023. doi:10.1007/978-981-99-8739-9_11.

[APS15] Martin R. Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of learning with errors. Cryptology ePrint Archive, Report 2015/046, 2015. URL: https://eprint.iacr.org/2015/046.

[ARS$^+$15] Martin R. Albrecht, Christian Rechberger, Thomas Schneider, Tyge Tiessen, and Michael Zohner. Ciphers for MPC and FHE. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015, Part I*, volume 9056 of *Lecture Notes in Computer Science*, pages 430–454. Springer, Berlin, Heidelberg, April 2015. doi:10.1007/978-3-662-46800-5_17.

[AST24] Thomas Aulbach, Simona Samardjiska, and Monika Trimoska. Practical key-recovery attack on MQ-sign and more. In Markku-Juhani Saarinen and Daniel Smith-Tone, editors, *Post-Quantum Cryptography - 15th International Workshop, PQCrypto 2024, Part II*, pages 168–185. Springer, Cham, June 2024. doi:10.1007/978-3-031-62746-0_8.

[BBB+20] Magali Bardet, Pierre Briaud, Maxime Bros, Philippe Gaborit, Vincent Neiger, Olivier Ruatta, and Jean-Pierre Tillich. An algebraic attack on rank metric code-based cryptosystems. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020, Part III*, volume 12107 of *Lecture Notes in Computer Science*, pages 64–93. Springer, Cham, May 2020. doi:10.1007/978-3-030-45727-3_3.

[BBB+24] Daniel J. Bernstein, Karthikeyan Bhargavan, Shivam Bhasin, Anupam Chattopadhyay, Tee Kiah Chia, Matthias J. Kannwischer, Franziskus Kiefer, Thales Paiva, Prasanna Ravi, and Goutam Tamvada. KyberSlash: Exploiting secret-dependent division timings in Kyber implementations, 2024. URL: https://cr.yp.to/papers.html#kyberslash.

[BBC+20a] Magali Bardet, Maxime Bros, Daniel Cabarcas, Philippe Gaborit, Ray A. Perlner, Daniel Smith-Tone, Jean-Pierre Tillich, and Javier A. Verbel. Improvements of algebraic attacks for solving the rank decoding and MinRank problems. In Shiho Moriai and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2020, Part I*, volume 12491 of *Lecture Notes in Computer Science*, pages 507–536. Springer, Cham, December 2020. doi:10.1007/978-3-030-64837-4_17.

[BBC+20b] Daniel J. Bernstein, Billy Bob Brumley, Ming-Shing Chen, Chitchanok Chuengsatiansup, Tanja Lange, Adrian Marotzke, Bo-Yuan Peng, Nicola Tuveri, Christine van Vredendaal, and Bo-Yin Yang. NTRU Prime. Technical report, National Institute of Standards and Technology, 2020. available at https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions.

[BBD+23] Manuel Barbosa, Gilles Barthe, Christian Doczkal, Jelle Don, Serge Fehr, Benjamin Grégoire, Yu-Hsuan Huang, Andreas Hülsing, Yi Lee, and Xiaodi Wu. Fixing and mechanizing the security proof of Fiat-Shamir with aborts and Dilithium. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology – CRYPTO 2023, Part V*, volume 14085 of *Lecture Notes in Computer Science*, pages 358–389. Springer, Cham, August 2023. doi:10.1007/978-3-031-38554-4_12.

[BBE+19] Gilles Barthe, Sonia Belaïd, Thomas Espitau, Pierre-Alain Fouque, Mélissa Rossi, and Mehdi Tibouchi. GALACTICS: Gaussian sampling for lattice-based constant- time implementation of cryptographic signatures, revisited. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019: 26th Conference on Computer and Communications Security*, pages 2147–2164. ACM Press, November 2019. doi:10.1145/3319535.3363223.

[BBM00] Mihir Bellare, Alexandra Boldyreva, and Silvio Micali. Public-key encryption in a multi-user setting: Security proofs and improvements. In Bart Preneel, editor, *Advances in Cryptology – EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 259–274. Springer, Berlin, Heidelberg, May 2000. doi:10.1007/3-540-45539-6_18.

[BBM+24] Carsten Baum, Ward Beullens, Shibam Mukherjee, Emmanuela Orsini, Sebastian Ramacher, Christian Rechberger, Lawrence Roy, and Peter Scholl. One tree to rule them all: Optimizing GGM trees and OWFs for post-quantum signatures. Cryptology ePrint Archive, Paper 2024/490, 2024. https://eprint.iacr.org/2024/490.

[BC24] Daniel J. Bernstein and Tung Chou. CryptAttackTester: high-assurance attack analysis. In Leonid Reyzin and Douglas Stebila, editors, *Advances in Cryptology – CRYPTO 2024, Part VI*, volume 14925 of *Lecture Notes in Computer Science*, pages 141–182. Springer, Cham, August 2024. doi:10.1007/978-3-031-68391-6_5.

[BCC+10] Charles Bouillaguet, Hsieh-Chung Chen, Chen-Mou Cheng, Tung Chou, Ruben Niederhagen, Adi Shamir, and Bo-Yin Yang. Fast exhaustive search for polynomial systems in $\mathbb{F}_2$. In Stefan Mangard and François-Xavier Standaert, editors, *Cryptographic Hardware and Embedded Systems – CHES 2010*, volume 6225 of *Lecture Notes in Computer Science*, pages 203–218. Springer, Berlin, Heidelberg, August 2010. doi:10.1007/978-3-642-15031-9_14.

[BCH+23] Ward Beullens, Ming-Shing Chen, Shih-Hao Hung, Matthias J. Kannwischer, Bo-Yuan Peng, Cheng-Jhih Shih, and Bo-Yin Yang. Oil and vinegar: Modern parameters and implementations. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2023(3):321–365, 2023. doi:10.46586/tches.v2023.i3.321-365.

[BCLv17] Daniel J. Bernstein, Chitchanok Chuengsatiansup, Tanja Lange, and Christine van Vredendaal. NTRU prime: Reducing attack surface at low cost. In Carlisle Adams and Jan Camenisch, editors, *SAC 2017: 24th Annual International Workshop on Selected Areas in Cryptography*, volume 10719 of *Lecture Notes in Computer Science*, pages 235–260. Springer, Cham, August 2017. doi:10.1007/978-3-319-72565-9_12.

[BCS13] Daniel J. Bernstein, Tung Chou, and Peter Schwabe. McBits: Fast constant-time code-based cryptography. In Guido Bertoni and Jean-Sébastien Coron, editors, *Cryptographic Hardware and Embedded Systems – CHES 2013*, volume 8086 of *Lecture Notes in Computer Science*, pages 250–272. Springer, Berlin, Heidelberg, August 2013. doi:10.1007/978-3-642-40349-1_15.

[Ber03] Thierry P. Berger. Isometries for rank distance and permutation group of Gabidulin codes. *IEEE Trans. Inf. Theory*, 49(11):3016–3019, 2003.

[Ber16] Daniel J. Bernstein. The post-quantum Internet (slides and video), 2016. URL: https://cr.yp.to/talks.html#2016.02.24.

[Ber17] Daniel J. Bernstein. Divergence bounds for random fixed-weight vectors obtained by sorting, 2017. URL: https://cr.yp.to/papers.html#divergence.

[Ber22a] Daniel J. Bernstein. combinatorial lattice security? Email to KpqC bulletin, 2022.

[Ber22b] Daniel J. Bernstein. Multi-ciphertext security degradation for lattices. Cryptology ePrint Archive, Report 2022/1580, 2022. URL: https://eprint.iacr.org/2022/1580.

[Ber23a] Daniel J. Bernstein. Predicting performance for post-quantum encrypted-file systems, 2023. URL: https://cr.yp.to/papers.html#pppqefs.

[Ber23b] Daniel J. Bernstein. Predicting performance for post-quantum encrypted-file systems. Cryptology ePrint Archive, Report

2023/1878, 2023. URL: https://eprint.iacr.org/2023/1878.

[Ber24a] Daniel J. Bernstein. insertionseries-20240515 (software), 2024. URL: https://cr.yp.to/2024/insertionseries-20240515.py.

[Ber24b] Daniel J. Bernstein. Understanding binary-Goppa decoding. *IACR Communications in Cryptology*, 1, 2024. https://cr.yp.to/papers.html#goppadecoding.

[Beu22] Ward Beullens. Breaking Rainbow takes a weekend on a laptop. In Yevgeniy Dodis and Thomas Shrimpton, editors, *Advances in Cryptology – CRYPTO 2022, Part II*, volume 13508 of *Lecture Notes in Computer Science*, pages 464–479. Springer, Cham, August 2022. doi:10.1007/978-3-031-15979-4_16.

[BFFP11] Charles Bouillaguet, Jean-Charles Faugère, Pierre-Alain Fouque, and Ludovic Perret. Practical cryptanalysis of the identification scheme based on the isomorphism of polynomial with one secret problem. In Dario Catalano, Nelly Fazio, Rosario Gennaro, and Antonio Nicolosi, editors, *PKC 2011: 14th International Conference on Theory and Practice of Public Key Cryptography*, volume 6571 of *Lecture Notes in Computer Science*, pages 473–493. Springer, Berlin, Heidelberg, March 2011. doi:10.1007/978-3-642-19379-8_29.

[BG14] Shi Bai and Steven D. Galbraith. An improved compression technique for signatures based on learning with errors. In Josh Benaloh, editor, *Topics in Cryptology – CT-RSA 2014*, volume 8366 of *Lecture Notes in Computer Science*, pages 28–47. Springer, Cham, February 2014. doi:10.1007/978-3-319-04852-9_2.

[BHH$^+$19] Nina Bindel, Mike Hamburg, Kathrin Hövelmanns, Andreas Hülsing, and Edoardo Persichetti. Tighter proofs of CCA security in the quantum random oracle model. In Dennis Hofheinz and Alon Rosen, editors, *TCC 2019: 17th Theory of Cryptography Conference, Part II*, volume 11892 of *Lecture Notes in Computer Science*, pages 61–90. Springer, Cham, December 2019. doi:10.1007/978-3-030-36033-7_3.

[BMP13] Joan Boyar, Philip Matthews, and René Peralta. Logic minimization techniques with applications to cryptology. *Journal of Cryptology*, 26(2):280–312, April 2013. doi:10.1007/s00145-012-9124-7.

[BN19] S. V. Bezzateev and I. K. Noskov. Patterson algorithm for decoding separable binary Goppa codes. In *2019 Wave Electronics and its Application in Information and Telecommunication Systems (WECONF)*, pages 1–5, 2019. doi:10.1109/WECONF.2019.8840650.

[BP18] Daniel J. Bernstein and Edoardo Persichetti. Towards KEM unification. Cryptology ePrint Archive, Report 2018/526, 2018. URL: https://eprint.iacr.org/2018/526.

[CC20] Daniel Coggia and Alain Couvreur. On the security of a Loidreau rank metric code based encryption scheme. *Des. Codes Cryptogr.*, 88(9):1941–1957, 2020.

[CCD$^+$22] Jung Hee Cheon, Hyeongmin Choe, Julien Devevey, Tim Güneysu, Dongyeon Hong, Markus Krausz, Georg Land, Junbum Shin, and Damien Stehlé. HAETAE. Submission to KpqC Round 1, 2022.

[CCH$^+$22] Jung Hee Cheon, Hyeongmin Choe, Dongyeon Hong, Jeongdae Hong, Hyoeun Seong, Junbum Shin, and MinJune Yi. SMAUG: the Key Exchange Algorithm based on Module-LWE and Module-LWR. Submission to KpqC Round 1, 2022.

[CDG$^+$17] Melissa Chase, David Derler, Steven Goldfeder, Claudio Orlandi, Sebastian Ramacher, Christian Rechberger, Daniel Slamanig, and Greg Zaverucha. Post-quantum zero-knowledge and signatures from symmetric-key primitives. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017: 24th Conference on Computer and Communications Security*, pages 1825–1842. ACM Press, October / November 2017. doi:10.1145/3133956.3133997.

[CHH$^+$23] Jolijn Cottaar, Kathrin Hövelmanns, Andreas Hülsing, Tanja Lange, Mohammad Mahzoun, Alex Pellegrini, Alberto Ravagnani, Sven Schäge, Monika Trimoska, and Benne de Weger. Report on evaluation of KpqC candidates. Cryptology ePrint

Archive, Report 2023/1853, 2023. URL: https://eprint.iacr.org/2023/1853.

[CHK$^+$21] Chi-Ming Marvin Chung, Vincent Hwang, Matthias J. Kannwischer, Gregor Seiler, Cheng-Jhih Shih, and Bo-Yin Yang. NTT multiplication for NTT-unfriendly rings. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021(2):159–188, 2021. URL: https://tches.iacr.org/index.php/TCHES/article/view/8791, doi:10.46586/tches.v2021.i2.159-188.

[Cho17] Tung Chou. McBits revisited. In Wieland Fischer and Naofumi Homma, editors, *Cryptographic Hardware and Embedded Systems – CHES 2017*, volume 10529 of *Lecture Notes in Computer Science*, pages 213–231. Springer, Cham, September 2017. doi:10.1007/978-3-319-66787-4_11.

[CKK$^+$23] YongRyeol Choi, MinGi Kim, YoungBeom Kim, JinGyo Song, JaeHwan Jin, HeeSeok Kim, and Seog Chung Seo. KpqBench: performance and implementation security analysis of KpqC competition round 1 candidates, 2023. URL: https://eprint.iacr.org/2023/1437.

[CKP$^+$20] Shaanan Cohney, Andrew Kwong, Shahar Paz, Daniel Genkin, Nadia Heninger, Eyal Ronen, and Yuval Yarom. Pseudorandom black swans: Cache attacks on CTR_DRBG. In *2020 IEEE Symposium on Security and Privacy*, pages 1241–1258. IEEE Computer Society Press, May 2020. doi:10.1109/SP40000.2020.00046.

[CKPS00] Nicolas Courtois, Alexander Klimov, Jacques Patarin, and Adi Shamir. Efficient algorithms for solving overdefined systems of multivariate polynomial equations. In Bart Preneel, editor, *Advances in Cryptology – EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 392–407. Springer, Berlin, Heidelberg, May 2000. doi:10.1007/3-540-45539-6_27.

[CMT23] Alain Couvreur, Rocco Mora, and Jean-Pierre Tillich. A new approach based on quadratic forms to attack the McEliece cryptosystem. Cryptology ePrint Archive, Paper 2023/950, 2023. https://eprint.iacr.org/2023/950.

[COT14] Alain Couvreur, Ayoub Otmani, and Jean-Pierre Tillich. Polynomial time attack on wild McEliece over quadratic extensions. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology – EUROCRYPT 2014*, volume 8441 of *Lecture Notes in Computer Science*, pages 17–39. Springer, Berlin, Heidelberg, May 2014. doi:10.1007/978-3-642-55220-5_2.

[DB22] Jan-Pieter D'Anvers and Senne Batsleer. Multitarget decryption failure attacks and their application to Saber and Kyber. In Goichiro Hanaoka, Junji Shikata, and Yohei Watanabe, editors, *PKC 2022: 25th International Conference on Theory and Practice of Public Key Cryptography, Part I*, volume 13177 of *Lecture Notes in Computer Science*, pages 3–33. Springer, Cham, March 2022. doi:10.1007/978-3-030-97121-2_1.

[DCP$^+$20] Jintai Ding, Ming-Shing Chen, Albrecht Petzoldt, Dieter Schmidt, Bo-Yin Yang, Matthias J. Kannwischer, and Jacques Patarin. Rainbow. Technical report, National Institute of Standards and Technology, 2020. available at https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions.

[DDLL13] Léo Ducas, Alain Durmus, Tancrède Lepoint, and Vadim Lyubashevsky. Lattice signatures and bimodal Gaussians. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013, Part I*, volume 8042 of *Lecture Notes in Computer Science*, pages 40–56. Springer, Berlin, Heidelberg, August 2013. doi:10.1007/978-3-642-40041-4_3.

[Del78] Philippe Delsarte. Bilinear forms over a finite field, with applications to coding theory. *Journal of Combinatorial Theory, Series A*, 25(3):226–241, 1978.

[Den03] Alexander W. Dent. A designer's guide to KEMs. In Kenneth G. Paterson, editor, *9th IMA International Conference on Cryptography and Coding*, volume 2898 of *Lecture Notes in Computer Science*, pages 133–151. Springer, Berlin, Heidelberg, December 2003. doi:10.1007/978-3-540-40974-8_12.

[DFPS23] Julien Devevey, Pouria Fallahpour, Alain Passelègue, and Damien Stehlé. A detailed analysis of Fiat-Shamir with aborts. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances* *in Cryptology – CRYPTO 2023, Part V*, volume 14085 of *Lecture Notes in Computer Science*, pages 327–357. Springer, Cham, August 2023. doi:10.1007/978-3-031-38554-4_11.

[DGZ17] Jean-Christophe Deneuville, Philippe Gaborit, and Gilles Zémor. Ouroboros: A simple, secure and efficient key exchange protocol based on coding theory. In Tanja Lange and Tsuyoshi Takagi, editors, *Post-Quantum Cryptography - 8th International Workshop, PQCrypto 2017*, pages 18–34. Springer, Cham, 2017. doi:10.1007/978-3-319-59879-6_2.

[DHK$^+$21] Julien Duman, Kathrin Hövelmanns, Eike Kiltz, Vadim Lyubashevsky, and Gregor Seiler. Faster lattice-based KEMs via a generic fujisaki-okamoto transform using prefix hashing. In Giovanni Vigna and Elaine Shi, editors, *ACM CCS 2021: 28th Conference on Computer and Communications Security*, pages 2722–2737. ACM Press, November 2021. doi:10.1145/3460120.3484819.

[DHK$^+$23] Julien Duman, Kathrin Hövelmanns, Eike Kiltz, Vadim Lyubashevsky, Gregor Seiler, and Dominique Unruh. A thorough treatment of highly-efficient NTRU instantiations. In Alexandra Boldyreva and Vladimir Kolesnikov, editors, *PKC 2023: 26th International Conference on Theory and Practice of Public Key Cryptography, Part I*, volume 13940 of *Lecture Notes in Computer Science*, pages 65–94. Springer, Cham, May 2023. doi:10.1007/978-3-031-31368-4_3.

[DKR$^+$20] Jan-Pieter D'Anvers, Angshuman Karmakar, Sujoy Sinha Roy, Frederik Vercauteren, Jose Maria Bermudo Mera, Michiel Van Beirendonck, and Andrea Basso. SABER. Technical report, National Institute of Standards and Technology, 2020. available at https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions.

[DP23] Léo Ducas and Ludo N. Pulles. Does the dual-sieve attack on learning with errors even work? In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology – CRYPTO 2023, Part III*, volume 14083 of *Lecture Notes in Computer Science*, pages 37–69. Springer, Cham, August 2023. doi:10.1007/978-3-031-38548-3_2.

[DRV20] Jan-Pieter D'Anvers, Mélissa Rossi, and Fernando Virdia. (One) failure is not an option: Bootstrapping the search for failures in lattice-based encryption schemes. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020, Part III*, volume 12107 of *Lecture Notes in Computer Science*, pages 3–33. Springer, Cham, May 2020. doi:10.1007/978-3-030-45727-3_1.

[DS05] Jintai Ding and Dieter Schmidt. Rainbow, a new multivariable polynomial signature scheme. In John Ioannidis, Angelos Keromytis, and Moti Yung, editors, *ACNS 05: 3rd International Conference on Applied Cryptography and Network Security*, volume 3531 of *Lecture Notes in Computer Science*, pages 164–175. Springer, Berlin, Heidelberg, June 2005. doi:10.1007/11496137_12.

[DYC$^+$08] Jintai Ding, Bo-Yin Yang, Chia-Hsin Owen Chen, Ming-Shing Chen, and Chen-Mou Cheng. New differential-algebraic attacks and reparametrization of Rainbow. In Steven M. Bellovin, Rosario Gennaro, Angelos D. Keromytis, and Moti Yung, editors, *ACNS 08: 6th International Conference on Applied Cryptography and Network Security*, volume 5037 of *Lecture Notes in Computer Science*, pages 242–257. Springer, Berlin, Heidelberg, June 2008. doi:10.1007/978-3-540-68914-0_15.

[EM22] Freja Elbro and Christian Majenz. An algebraic attack against McEliece-like cryptosystems based on BCH codes. Cryptology ePrint Archive, Report 2022/1715, 2022. URL: https://eprint.iacr.org/2022/1715.

[FO99] Eiichiro Fujisaki and Tatsuaki Okamoto. How to enhance the security of public-key encryption at minimum cost. In Hideki Imai and Yuliang Zheng, editors, *PKC'99: 2nd International Workshop on Theory and Practice in Public Key Cryptography*, volume 1560 of *Lecture Notes in Computer Science*, pages 53–68. Springer, Berlin, Heidelberg, March 1999. doi:10.1007/3-540-49162-7_5.

[FO13] Eiichiro Fujisaki and Tatsuaki Okamoto. Secure integration of asymmetric and symmetric encryption schemes. *Journal of Cryptology*, 26(1):80–101, January 2013. doi:10.1007/s00145-011-9114-1.

[FOPT10] Jean-Charles Faugère, Ayoub Otmani, Ludovic Perret, and Jean-Pierre Tillich. Algebraic cryptanalysis of McEliece variants with compact keys. In Henri Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, volume 6110 of *Lecture Notes in Computer Science*, pages 279–298. Springer, Berlin, Heidelberg, May / June 2010. doi:10.1007/978-3-642-13190-5_14.

[For18] Classic McEliece Comparison Task Force. Classic McEliece vs. NTS-KEM. Posted as https://classic.mceliece.org/nist/vsntskem-20180629.pdf, 2018.

[FRL24] Décio Luiz Gazzoni Filho, Tomás Recio, and Julio López. Efficient isochronous fixed-weight sampling with applications to NTRU, 2024. URL: https://eprint.iacr.org/2024/548.

[Gab85] Ernst M. Gabidulin. Theory of codes with maximum rank distance. *Problems of Information Transmission*, 21(1):1–12, 1985.

[Gär23] Joel Gärtner. Concrete security from worst-case to average-case lattice reductions. In Nadia El Mrabet, Luca De Feo, and Sylvain Duquesne, editors, *Progress in Cryptology - AFRICACRYPT 2023 - 14th International Conference on Cryptology in Africa, Sousse, Tunisia, July 19-21, 2023, Proceedings*, volume 14064 of *Lecture Notes in Computer Science*, pages 344–369. Springer, 2023. doi:10.1007/978-3-031-37679-5_15.

[Gha22] Anirban Ghatak. Extending Coggia-Couvreur attack on Loidreau's rank-metric cryptosystem. *Des. Codes Cryptogr.*, 90(1):215–238, 2022.

[GKK$^+$17] Lucky Galvez, Jon-Lark Kim, Myeong Jae Kim, Young-Sik Kim, and Nari Lee. McNie. Technical report, National Institute of Standards and Technology, 2017. available at https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-1-submissions.

[GPT91] Ernst M. Gabidulin, A. V. Paramonov, and O. V. Tretjakov. Ideals over a non-commutative ring and thier applications in cryptology. In Donald W. Davies, editor, *Advances in Cryptology – EUROCRYPT'91*, volume 547 of *Lecture Notes in Computer Science*, pages 482–489. Springer, Berlin, Heidelberg, April 1991. doi:10.1007/3-540-46416-6_41.

[HHK17] Dennis Hofheinz, Kathrin Hövelmanns, and Eike Kiltz. A modular analysis of the Fujisaki-Okamoto transformation. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017: 15th Theory of Cryptography Conference, Part I*, volume 10677 of *Lecture Notes in Computer Science*, pages 341–371. Springer, Cham, November 2017. doi:10.1007/978-3-319-70500-2_12.

[HHM22] Kathrin Hövelmanns, Andreas Hülsing, and Christian Majenz. Failing gracefully: Decryption failures and the Fujisaki-Okamoto transform. In Shweta Agrawal and Dongdai Lin, editors, *Advances in Cryptology – ASIACRYPT 2022, Part IV*, volume 13794 of *Lecture Notes in Computer Science*, pages 414–443. Springer, Cham, December 2022. doi:10.1007/978-3-031-22972-5_15.

[HJMN24] Andreas Hülsing, David Joseph, Christian Majenz, and Anand Kumar Narayanan. On round elimination for special-sound multi-round identification and the generality of the hypercube for MPCitH. In Leonid Reyzin and Douglas Stebila, editors, *Advances in Cryptology – CRYPTO 2024, Part I*, volume 14920 of *Lecture Notes in Computer Science*, pages 373–408. Springer, Cham, August 2024. doi:10.1007/978-3-031-68376-3_12.

[HPS98] Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. NTRU: A ring-based public key cryptosystem. In *Third Algorithmic Number Theory Symposium (ANTS)*, volume 1423 of *Lecture Notes in Computer Science*, pages 267–288. Springer, June 1998.

[HS01] Jeffrey Hoffstein and Joseph H. Silverman. Optimizations for NTRU. In *Public-key cryptography and computational number theory. Proceedings of the international conference organized by the Stefan Banach International Mathematical Center, Warsaw, Poland, September 11–15, 2000*, pages 77–88. de Gruyter, 2001.

[HTMR16] Anna-Lena Horlemann-Trautmann, Kyle Marshall, and Joachim Rosenthal. Considerations for rank-based cryptosystems. In *2016 IEEE International Symposium on Information Theory (ISIT)*, pages 2544–2548, 2016. doi:10.1109/ISIT.2016.7541758.

[IJY23] Yasuhiko Ikematsu, Hyungrok Jo, and Takanori Yasuda. A security analysis on MQ-Sign. Cryptology ePrint Archive, Paper 2023/581, 2023. https://eprint.iacr.org/2023/581.

[IKOS07] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Zero-knowledge from secure multiparty computation. In David S. Johnson and Uriel Feige, editors, *39th Annual ACM Symposium on Theory of Computing*, pages 21–30. ACM Press, June 2007. doi:10.1145/1250790.1250794.

[IKOS09] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Zero-knowledge proofs from secure multiparty computation. *SIAM J. Comput.*, 39(3):1121–1152, 2009. doi:10.1137/080725398.

[JJ00] Éliane Jaulmes and Antoine Joux. A chosen-ciphertext attack against NTRU. In Mihir Bellare, editor, *Advances in Cryptology – CRYPTO 2000*, volume 1880 of *Lecture Notes in Computer Science*, pages 20–35. Springer, Berlin, Heidelberg, August 2000. doi:10.1007/3-540-44598-6_2.

[JZM21] Haodong Jiang, Zhenfeng Zhang, and Zhi Ma. On the non-tightness of measurement-based reductions for key encapsulation mechanism in the quantum random oracle model. In Mehdi Tibouchi and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2021, Part I*, volume 13090 of *Lecture Notes in Computer Science*, pages 487–517. Springer, Cham, December 2021. doi:10.1007/978-3-030-92062-3_17.

[KCC$^+$23] Seongkwang Kim, Jihoon Cho, Mingyu Cho, Jincheol Ha, Jihoon Kwon, Byeonghak Lee, Joohee Lee, Jooyoung Lee, Sangyub Lee, Dukjae Moon, Mincheol Son, and Hyojin Yoon. AIMer. Technical report, National Institute of Standards and Technology, 2023. available at https://csrc.nist.gov/Projects/pqc-dig-sig/round-1-additional-signatures.

[KHL$^+$22] Jon-Lark Kim, Jihoon Hong, Terry Shue Chien Lau, YounJae Lim, Chik How Tan, Theo Fanuela Prabowo, and Byung-Sun Won. REDOG. Submission to KpqC Round 1, 2022.

[KHL$^+$24] Jon-Lark Kim, Jihoon Hong, Terry Shue Chien Lau, YounJae Lim, Byung-Sun Won, and Bo-seung Yang. REDOG. Submission to KpqC Round 2, 2024.

[KHS$^+$22] Seongkwang Kim, Jincheol Ha, Mincheol Son, Byeonghak Lee, Dukjae Moon, Joohee Lee, Sangyup Lee, Jihoon Kwon, Jihoon Cho, Hyojin Yoon, and Jooyoung Lee. AIM: Symmetric primitive for shorter signatures with stronger security. Cryptology ePrint Archive, Report 2022/1387, 2022. URL: https://eprint.iacr.org/2022/1387.

[KHSL23] Seongkwang Kim, Jincheol Ha, Mincheol Son, and Byeonghak Lee. Mitigation on the AIM cryptanalysis. Cryptology ePrint Archive, Paper 2023/1474, 2023. https://eprint.iacr.org/2023/1474.

[KJKK22] Dong-Chan Kim, Chang-Yeol Jeon, Yeonghyo Kim, and Minji Kim. PALOMA: Binary Separable Goppa-based KEM. Submission to KpqC Round 1, 2022. https://www.kpqc.or.kr/images/pdf/PALOMA.pdf.

[KKGK21] Jon-Lark Kim, Young-Sik Kim, Lucky Erap Galvez, and Myeong Jae Kim. A modified Dual-Ouroboros public-key encryption using Gabidulin codes. *Appl. Algebra Eng. Commun. Comput.*, 32(2):147–156, 2021.

[KLS18] Eike Kiltz, Vadim Lyubashevsky, and Christian Schaffner. A concrete treatment of Fiat-Shamir signatures in the quantum random-oracle model. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018, Part III*, volume 10822 of *Lecture Notes in Computer Science*, pages 552–586. Springer, Cham, April / May 2018. doi:10.1007/978-3-319-78372-7_18.

[KP22] Jonghyun Kim and Jong Hwan Park. NTRU+: Compact Construction of NTRU Using Simple Encoding Method. Submission to KpqC Round 1, 2022.

[KPG99] Aviad Kipnis, Jacques Patarin, and Louis Goubin. Unbalanced Oil and Vinegar signature schemes. In Jacques Stern, editor, *Advances in Cryptology – EUROCRYPT'99*, volume 1592 of *Lecture Notes in Computer Science*, pages 206–222. Springer, Berlin, Heidelberg, May 1999. doi:10.1007/3-540-48910-X_15.

[KS98] Aviad Kipnis and Adi Shamir. Cryptanalysis of the Oil & Vinegar signature scheme. In Hugo Krawczyk, editor, *Advances in*

Cryptology – CRYPTO'98, volume 1462 of *Lecture Notes in Computer Science*, pages 257–266. Springer, Berlin, Heidelberg, August 1998. doi:10.1007/BFb0055733.

[KS09] Emilia Käsper and Peter Schwabe. Faster and timing-attack resistant AES-GCM. In Christophe Clavier and Kris Gaj, editors, *Cryptographic Hardware and Embedded Systems – CHES 2009*, volume 5747 of *Lecture Notes in Computer Science*, pages 1–17. Springer, Berlin, Heidelberg, September 2009. doi:10.1007/978-3-642-04138-9_1.

[KX24] Haruhisa Kosuge and Keita Xagawa. Probabilistic hash-and-sign with retry in the quantum random oracle model. In Qiang Tang and Vanessa Teague, editors, *PKC 2024: 27th International Conference on Theory and Practice of Public Key Cryptography, Part I*, volume 14601 of *Lecture Notes in Computer Science*, pages 259–288. Springer, Cham, April 2024. doi:10.1007/978-3-031-57718-5_9.

[KZ22] Daniel Kales and Greg Zaverucha. Efficient lifting for shorter zero-knowledge proofs and post-quantum signatures. Cryptology ePrint Archive, Report 2022/588, 2022. URL: https://eprint.iacr.org/2022/588.

[LDK+20] Vadim Lyubashevsky, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Peter Schwabe, Gregor Seiler, Damien Stehlé, and Shi Bai. CRYSTALS-DILITHIUM. Technical report, National Institute of Standards and Technology, 2020. available at https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions.

[LDK+22] Vadim Lyubashevsky, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Peter Schwabe, Gregor Seiler, Damien Stehlé, and Shi Bai. CRYSTALS-DILITHIUM. Technical report, National Institute of Standards and Technology, 2022. available at https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022.

[Lee23a] Joohee Lee. Analysis of NTRU+. Email to KpqC bulletin, 2023.

[Lee23b] Seungwoo Lee. Analysis of TiGER. Email to KpqC Bulletin, 2023.

[Lem12] Mark A. Lemley. The myth of the sole inventor. *Michigan Law Review*, 110:709, 2012.

[LMM24] Fukang Liu, Mohammad Mahzoun, and Willi Meier. Modelling ciphers with overdefined systems of quadratic equations: Application to Friday, Vision, RAIN and Biscuit. Cryptology ePrint Archive, Paper 2024/786, 2024. https://eprint.iacr.org/2024/786.

[LMØM23] Fukang Liu, Mohammad Mahzoun, Morten Øygarden, and Willi Meier. Algebraic attacks on RAIN and AIM using equivalent representations. Cryptology ePrint Archive, Paper 2023/1133, 2023. https://eprint.iacr.org/2023/1133.

[LPR10a] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In Henri Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, volume 6110 of *Lecture Notes in Computer Science*, pages 1–23. Springer, Berlin, Heidelberg, May / June 2010. doi:10.1007/978-3-642-13190-5_1.

[LPR10b] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings, 2010. URL: https://iacr.org/conferences/eurocrypt2010/talks/slides-ideal-lwe.pdf.

[LPR23] Tanja Lange, Alex Pellegrini, and Alberto Ravagnani. On the security of REDOG. In *ICISC (2)*, volume 14562 of *Lecture Notes in Computer Science*, pages 282–305. Springer, 2023.

[LS19] Vadim Lyubashevsky and Gregor Seiler. NTTRU: Truly fast NTRU using NTT. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2019(3):180–201, 2019. URL: https://tches.iacr.org/index.php/TCHES/article/view/8293, doi:10.13154/tches.v2019.i3.180-201.

[LS21] Vadim Lyubashevsky and Damien Stehlé. Non-applicability of the Gaborit&Aguilar-Melchor patent to Kyber and Saber. Cryptology ePrint Archive, Report 2021/1364, 2021. URL: https://eprint.iacr.org/2021/1364.

[LT18] Terry Shue Chien Lau and Chik How Tan. Key recovery attack on McNie based on low rank parity check codes and its reparation. In Atsuo Inomata and Kan Yasuda, editors, *IWSEC 18:*

13th International Workshop on Security, Advances in Information and Computer Security, volume 11049 of *Lecture Notes in Computer Science*, pages 19–34. Springer, Cham, September 2018. doi:10.1007/978-3-319-97916-8_2.

[LT19] Terry Shue Chien Lau and Chik How Tan. New rank codes based encryption scheme using partial circulant matrices. *Des. Codes Cryptogr.*, 87(12):2979–2999, 2019.

[LTP21] Terry Shue Chien Lau, Chik How Tan, and Theo Fanuela Prabowo. On the security of the modified Dual-Ouroboros PKE using Gabidulin codes. *Appl. Algebra Eng. Commun. Comput.*, 32(6):681–699, 2021.

[Lyu09] Vadim Lyubashevsky. Fiat-Shamir with aborts: Applications to lattice and factoring-based signatures. In Mitsuru Matsui, editor, *Advances in Cryptology – ASIACRYPT 2009*, volume 5912 of *Lecture Notes in Computer Science*, pages 598–616. Springer, Berlin, Heidelberg, December 2009. doi:10.1007/978-3-642-10366-7_35.

[Lyu12] Vadim Lyubashevsky. Lattice signatures without trapdoors. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 738–755. Springer, Berlin, Heidelberg, April 2012. doi:10.1007/978-3-642-29011-4_43.

[May21] Alexander May. How to meet ternary LWE keys. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology – CRYPTO 2021, Part II*, volume 12826 of *Lecture Notes in Computer Science*, pages 701–731, Virtual Event, August 2021. Springer, Cham. doi:10.1007/978-3-030-84245-1_24.

[McE78] Robert J. McEliece. A public-key cryptosystem based on algebraic coding theory. The deep space network progress report 42-44, Jet Propulsion Laboratory, California Institute of Technology, January/February 1978. https://ipnpr.jpl.nasa.gov/progress_report2/42-44/44N.PDF.

[Mer23] Jorge Correa Merlino. From Middle-Earth to the Galaxy: SMAUG vs. Kyber. Bachelor's thesis, Eindhoven University of Technology, 2023.

https://research.tue.nl/en/studentTheses/from-middle-earth-to-the-galaxy-smaug-vs-kyber.

[Ove05] Raphael Overbeck. A new structural attack for GPT and variants. In *Mycrypt*, volume 3715 of *Lecture Notes in Computer Science*, pages 50–63. Springer, 2005.

[Ove08] R. Overbeck. Structural attacks for public key cryptosystems based on Gabidulin codes. *Journal of Cryptology*, 21(2):280–301, April 2008. doi:10.1007/s00145-007-9003-9.

[Pat75] Nicholas J. Patterson. The algebraic decoding of Goppa codes. *IEEE Transactions on Information Theory*, 21:203–207, 1975.

[Pat97] Jacques Patarin. The oil and vinegar signature scheme. Dagstuhl Workshop on Cryptography, 1997.

[Péb24] Pierre Pébereau. One vector to rule them all: Key recovery from one vector in UOV schemes. In Markku-Juhani Saarinen and Daniel Smith-Tone, editors, *Post-Quantum Cryptography - 15th International Workshop, PQCrypto 2024, Part II*, pages 92–108. Springer, Cham, June 2024. doi:10.1007/978-3-031-62746-0_5.

[Pei14] Chris Peikert. Lattice cryptography for the internet. In Michele Mosca, editor, *Post-Quantum Cryptography - 6th International Workshop, PQCrypto 2014*, pages 197–219. Springer, Cham, October 2014. doi:10.1007/978-3-319-11659-4_12.

[Pei16] Chris Peikert. A decade of lattice cryptography. *Found. Trends Theor. Comput. Sci.*, 10(4):283–424, 2016. URL: https://eprint.iacr.org/2015/939, doi:10.1561/0400000074.

[Pet13] Albrecht Petzoldt. *Selecting and reducing key sizes for multivariate cryptography*. PhD thesis, Darmstadt University of Technology, Germany, 2013. URL: http://tuprints.ulb.tu-darmstadt.de/3523/.

[PJP+22] Seunghwan Park, Chi-Gon Jung, Aesun Park, Joongeun Choi, and Honggoo Kang. TiGER: Tiny bandwidth key encapsulation mechanism for easy miGration based on RLWE(R). Submission to KpqC Round 1, 2022.

[PP19] Chris Peikert and Zachary Pepin. Algebraically structured LWE, revisited. In Dennis Hofheinz and Alon Rosen, editors, *TCC 2019: 17th Theory of Cryptography Conference, Part I*, volume 11891 of *Lecture Notes in Computer Science*, pages 1–23. Springer, Cham, December 2019. doi:10.1007/978-3-030-36030-6_1.

[PS24] Amaury Pouly and Yixin Shen. Provable dual attacks on learning with errors. In Marc Joye and Gregor Leander, editors, *Advances in Cryptology – EUROCRYPT 2024, Part VII*, volume 14657 of *Lecture Notes in Computer Science*, pages 256–285. Springer, Cham, May 2024. doi:10.1007/978-3-031-58754-2_10.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *37th Annual ACM Symposium on Theory of Computing*, pages 84–93. ACM Press, May 2005. doi:10.1145/1060590.1060603.

[RT24] Lars Ran and Monika Trimoska. Shifting our knowledge of MQ-sign security. Cryptology ePrint Archive, Paper 2024/1891, 2024. URL: https://eprint.iacr.org/2024/1891.

[SAB$^+$22] Peter Schwabe, Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Gregor Seiler, Damien Stehlé, and Jintai Ding. CRYSTALS-KYBER. Technical report, National Institute of Standards and Technology, 2022. available at https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022.

[Sal19] Joseph Saltiel. In the courts: five years after Alice - five lessons learned from the treatment of software patents in litigation, 2019. URL: https://www.wipo.int/wipo_magazine/en/2019/04/article_0006.html.

[SE91] Claus-Peter Schnorr and Michael Euchner. Lattice basis reduction: Improved practical algorithms and solving subset sum problems. In Lothar Budach, editor, *FCT'91*, volume 529 of *LNCS*, pages 68–85. Springer, 1991.

[SK24] Kyung-Ah Shim and Hyeokdong Kwon. MQ-Sign: A New Post-Quantum Signature Scheme based on Multivariate Quadratic Equations: Shorter and Faster . Submission to KpqC Round 2, 2024.

[SKA22] Kyung-Ah Shim, Jeongsu Kim, and Youngjoo An. NCC-Sign: A New Lattice-based Signature Scheme using Non-Cyclotomic Polynomials. Submission to KpqC Round 1, 2022.

[SKA23] Kyung-Ah Shim, Jeongsu Kim, and Youngjoo An. NCC-Sign: A new lattice-based signature scheme using non-cyclotomic polynomials. https://github.com/jsk-NIMS/NCC-MQ-Sign/blob/main/NCC-Sign-v1.0.pdf, 2023.

[SLK22] Kyung-Ah Shim, Sangyub Lee, and Namhun Koo. Efficient implementations of Rainbow and UOV using AVX2. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2022(1):245–269, 2022. doi:10.46586/tches.v2022.i1.245-269.

[SSH11] Koichi Sakumoto, Taizo Shirai, and Harunaga Hiwatari. On provable security of UOV and HFE signature schemes against chosen-message attack. In Bo-Yin Yang, editor, *Post-Quantum Cryptography - 4th International Workshop, PQCrypto 2011*, pages 68–82. Springer, Berlin, Heidelberg, November / December 2011. doi:10.1007/978-3-642-25405-5_5.

[SSW20] Peter Schwabe, Douglas Stebila, and Thom Wiggers. Post-quantum TLS without handshake signatures. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020: 27th Conference on Computer and Communications Security*, pages 1461–1480. ACM Press, November 2020. doi:10.1145/3372297.3423350.

[Ste23] Luc G.A.M. Steenbakkers. NTRU+: Analyzing variants, attacks and security of a lattice-based post-quantum scheme. Bachelor's thesis, Eindhoven University of Technology, 2023. https://research.tue.nl/en/studentTheses/ntru-2.

[STHY23] Kyohei Sudo, Masayuki Tezuka, Keisuke Hara, and Yusuke Yoshida. Quantum search-to-decision reduction for the LWE problem. Cryptology ePrint Archive, Report 2023/344, 2023. URL: https://eprint.iacr.org/2023/344.

[SXY18] Tsunekazu Saito, Keita Xagawa, and Takashi Yamakawa. Tightly-secure key-encapsulation mechanism in the quantum random oracle model. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018, Part III*, volume 10822 of *Lecture Notes in Computer Science*, pages 520–551. Springer, Cham, April / May 2018. doi:10.1007/978-3-319-78372-7_17.

[Uni85] United States Court of Appeals for the Federal Circuit. MIT v. Ab Fortia, 1985. Note: 774 F.2d 1104. URL: https://law.justia.com/cases/federal/appellate-courts/F2/774/1104/10468/.

[Uni87] United States Court of Appeals for the Federal Circuit. Verdegaal Brothers, Inc. v. Union Oil Company, 1987. Note: 814 F.2d 628. URL: https://law.justia.com/cases/federal/appellate-courts/F2/814/628/335813/.

[Uni00] United States Court of Appeals for the Federal Circuit. Ultra-Tex Surfaces v. Chacon, 2000. Note: 204 F.3d 1360. URL: https://law.justia.com/cases/federal/appellate-courts/F3/204/1360/506445/.

[Uni02] United States Supreme Court. Festo Corp. v. Shoketsu Kinzoku Kogyo Kabushiki Co., 2002. Note: 535 U.S. 722. URL: https://tile.loc.gov/storage-services/service/ll/usrep/usrep535/usrep535722/usrep535722.pdf.

[Uni04] United States Court of Appeals for the Federal Circuit. In re Klopfenstein, 2004. Note: 380 F.3d 1345. URL: https://law.justia.com/cases/federal/appellate-courts/F3/380/1345/533650/.

[Uni07] United States Supreme Court. KSR Int'l Co. v. Teleflex Inc., 2007. Note: 550 U.S. 398. URL: https://supreme.justia.com/cases/federal/us/550/398/.

[Uni09] United States Court of Appeals for the Federal Circuit. Crown Packaging Tech., Inc. v. Rexam Beverage Can Co., 2009. Note: 559 F.3d 1308. URL: https://law.justia.com/cases/federal/appellate-courts/cafc/08-1284/08-1284-2011-03-27.html.

[Uni14] United States Supreme Court. Alice Corp. v. CLS Bank International, 2014. Note: 573 U.S. 208. URL: https://supreme.justia.com/cases/federal/us/573/208/.

[Uni23] United States District Court for the Eastern District of Virginia. Geoscope Technologies Pte. Ltd. v. Apple Inc., 2023. Note: E.D. Va. 2023. URL: https://law.justia.com/cases/federal/district-courts/virginia/vaedce/1:2022cv01373/531733/109/.

[WJW+19] Wen Wang, Bernhard Jungk, Julian Wälde, Shuwen Deng, Naina Gupta, Jakub Szefer, and Ruben Niederhagen. XMSS and embedded systems. In Kenneth G. Paterson and Douglas Stebila, editors, *SAC 2019: 26th Annual International Workshop on Selected Areas in Cryptography*, volume 11959 of *Lecture Notes in Computer Science*, pages 523–550. Springer, Cham, August 2019. doi:10.1007/978-3-030-38471-5_21.

[WXS+22] Geng Wang, Wenwen Xia, Gongyu Shi, Ming Wan, Yuncong Zhang, and Dawu Gu. Revisiting the concrete hardness of SelfTargetMSIS in CRYSTALS-Dilithium. Cryptology ePrint Archive, Report 2022/1601, 2022. URL: https://eprint.iacr.org/2022/1601.

[ZWY+23] Kaiyi Zhang, Qingju Wang, Yu Yu, Chun Guo, and Hongrui Cui. Algebraic attacks on round-reduced RAIN and full AIM;-III. Cryptology ePrint Archive, Paper 2023/1397, 2023. https://eprint.iacr.org/2023/1397.