Attacks on the AJPS Mersenne-based cryptosystem

Koen de Boer[1], Léo Ducas[1], Stacey Jeffery[1,2], and Ronald de Wolf[1,2,3]

[1]CWI, Amsterdam
[2]QuSoft
[3]University of Amsterdam

December 1, 2017

Abstract

Aggarwal, Joux, Prakash and Santha recently introduced a new potentially quantum-safe public-key cryptosystem, and suggested that a brute-force attack is essentially optimal against it. They consider but then dismiss both Meet-in-the-Middle attacks and LLL-based attacks. Very soon after their paper appeared, Beunardeau et al. proposed a practical LLL-based technique that seemed to significantly reduce the security of the AJPS system. In this paper we do two things. First, we show that a Meet-in-the-Middle attack can also be made to work against the AJPS system, using locality-sensitive hashing to overcome the difficulty that Aggarwal et al. saw for such attacks. We also present a quantum version of this attack. Second, we give a more precise analysis of the attack of Beunardeau et al., confirming and refining their results.

1 Introduction

Aggarwal, Joux, Prakash and Santha [1] recently proposed a variant of the NTRU public-key encryption scheme [13]. This variant uses integers with sparse binary representation as a secret key, rather than polynomials with small coefficients. In particular, their cryptosystem is suspected to be resistant to quantum attacks.

Their system works as follows. Consider a *Mersenne* number $N = 2^n - 1$, with n prime. Then we can identify the ring $R = \mathbb{Z}/N\mathbb{Z}$ with the set of n-bit strings, where 1^n is identified with 0^n. To set up the keys of the cryptosystem, choose $f, g \in R$ of fixed Hamming weight $w < \sqrt{n}/2$ uniformly at random, subject to g having a multiplicative inverse in R. Set the *public* key to $h := f/g$ (this corresponds to an n-bit string of arbitrary Hamming weight) and the *private* key to g. In the next section we show how Aggarwal et al. use these keys for encryption and decryption.

The security of this system relies on the assumption that it is hard to solve the following *Mersenne Low Hamming Ratio Search Problem*: given $n, w \in \mathbb{N}$ and $h \in R$, find $f, g \in R$ of weight w such that $h = f/g$, assuming such f and g exist. A brute-force attack on this system would just try out all $\binom{n-1}{w-1}$ possible g's of weight w that start with a 1 (the latter is without loss of generality) and check whether hg has weight w. Aggarwal et al. [1] suggest that this brute-force attack is close to optimal. This would correspond to roughly $\lambda = \log \binom{n-1}{w-1} \approx \frac{1}{2} \cdot w \log n$ bits of security. On a quantum computer, this brute-force attack could be implemented using Grover's quantum search algorithm [11] in time roughly $\sqrt{\binom{n-1}{w-1}}$, corresponding to roughly $\frac{1}{4} \cdot w \log n$ bits of security.

In particular, Aggarwal et al. [1] consider and then dismiss two possible lines of attack that could be better than brute force. First, they suggest that a combinatorial Meet-in-the-Middle attack would fail due to a problem of "approximate collisions". Second, they argue that their variation makes an adaptation of known lattice attacks against NTRU ineffective. The latter claim was rapidly challenged, when a faster experimental attack using LLL reduction was found by Beunardeau et al. [6]. This attack exploits the low weight of f and g, and is able to find f, g using partition-search in the interval $2^n - 1$. The authors argue that their attack reduces the bit security to about $\lambda \approx 2w$. Beunardeau et al. [6] warn that their attack is only practically feasible, and might not work with, for example, increasing parameters. They further expect that slightly changing the cryptosystem protects against this attack [6, §4].

1.1 This work

In this work we revisit the security of the AJPS cryptosystem. We first propose a Meet-in-the-Middle attack that circumvents the issues raised by [1] and gives a polynomial speed-up over a brute-force attack. It runs in classical time $\tilde{O}\left(\sqrt{\binom{n-1}{w-1}}\right)$, and can be accelerated on a quantum computer to $\tilde{O}\left(\sqrt[3]{\binom{n-1}{w-1}}\right)$. Our analysis requires several minor heuristics, which we have confirmed experimentally. Secondly, we formally analyze the attack of Beunardeau et al. [6]. Our analysis suggest that the attack is slightly less efficient, asymptotically, than suggested in [6]. However, this small difference in complexity makes little difference in practice.

Meet-in-the-Middle attack. Aggarwal et al. [1, § 5.1] described a failed attempt at a Meet-in-the-Middle (MITM) attack on their cryptosystem. It fails because the "collisions" in the "middle" are not exact, and they view this failure as evidence for the optimality of the brute-force attack. In contrast, we show how a MITM attack on their system can nonetheless be executed, using locality-sensitive hashing to overcome the issue of inexact, approximate collisions.

The idea is still, given $h \in \{0,1\}^n$, to find an n-bit string $g \in R$ of weight $\leq w$, such that hg has low weight. Split the n-bit string $g = g_1 \oplus g_2$ into an n-bit string g_1 with roughly αw 1s in the first αn bits and 0s elsewhere, and a g_2 with roughly $(1-\alpha)w$ 1s in the last $(1-\alpha)n$ bits and 0s elsewhere. Now $hg = hg_1 \oplus hg_2$ having low weight corresponds to hg_1 and hg_2 being approximately equal (i.e., having low Hamming distance), so our goal becomes to find an "approximate collision" between the two sets $\{hg_1\}$ and $\{hg_2\}$. We can do this by first computing all elements of the first set, together with their hashes, and storing these in an appropriate data structure. After that we search in the second set to find an approximate collision with the elements in the data structure (if such an approximate collision exists). This attack turns out to be substantially cheaper than a brute-force search over all g's of weight w. In the classical case, setting the split at $\alpha = 1/2$, the runtime of the attack is roughly $\binom{n/2}{w/2} \approx \binom{n}{w}^{1/2}$, which corresponds to roughly $\frac{1}{4} \cdot w \log n$ bits of security. In the quantum case, setting $\alpha = 1/3$, yields an algorithm similar to [5], which has runtime roughly $\binom{n/3}{w/3} \approx \binom{n}{w}^{1/3}$, corresponding to $\frac{1}{6} \cdot w \log n$ bits of security. For our quantum algorithm, we make use of a technique for quantum search among items with variable checking costs, due to Ambainis [3]. To complement our theoretical analysis we also implemented this attack on a classical computer and ran a simulation for quantum computers. Our source code is available at `https://github.com/lducas/MiTM-Mersenne`.

Analysis of the lattice attack of Beunardeau et al. Although Beunardeau et al. [6] provide experimental evidence for the efficiency of their attack, they leave open the task of providing a theoretical analysis to support the correctness of their approach. This leaves some uncertainty for a concrete security estimate of the cryptosystem of Aggarwal et al. We attempt to fill this gap with a more in-depth analysis of their attack. We conclude that the cost of their attack is in fact of the form $(2+\delta+o(1))^{2w}$ for some very small constant $\delta > 0$. Besides clarifying the heuristic asymptotic complexity of the attack of Beunardeau et al. [6], it also essentially confirms their practical claim that their attack reduces the security to roughly $2w$ bits.

1.2 Impact

The impact of this work is mostly of a conceptual nature. Our Meet-in-the-Middle attack is a reminder that inexact collisions can sometimes be circumvented, depending on the metric at hand. While a similar near-collision MITM attack was well known against NTRU (attributed to Odlyzko in [13]), it was rather easy due to how close the near-collisions were. The setting of Aggarwal et al. is more demanding. Our work also shows another application of Nearest-Neighbor Search (NNS) techniques to cryptanalysis, which have already found important application to lattice problems [17, 4, 16].

Our analysis of the attack of Beunardeau et al. [6] also provides better confidence in the revised security estimate of the treated cryptosystem [1]. Moreover, we hope that it provides clear tools and heuristics to understand the behavior of LLL in more general scenarios.

Open questions. Our work highlights several interesting open questions. Concerning the cryptosystem of [1], an interesting idea would be to see whether the lattice attack and the MITM attack could be combined into an even faster attack, as was already done against NTRU by Howgrave-Graham [14]. At first sight, it seems that this approach would not lead to an exponential acceleration, yet it may make it possible to amortize the polynomial cost of each call to the LLL algorithm.

More generally, our work highlights the question of Nearest-Neighbor Search using quantum computers. This question was already approached in [16, 18], which considered generic application of Grover's algorithm over classical NNS techniques. It seems an important question to determine whether less generic approaches could perform better.

1.3 Organization

The remainder of this paper is organized as follows. In Section 2, we give the necessary preliminaries, including a description of the AJPS cryptosystem of [1], and a description of a variant of the quantum search algorithm due to [3], for settings where the cost of checking if an element is marked varies. In Section 3, we present and analyze our classical Meet-in-the-Middle attack, and in Section 4, we present and analyze our quantum Meet-in-the-Middle attack. In Section 5, we present our formal analysis of the Beunardeau et al. attack [6].

2 Preliminaries

2.1 The AJPS Cryptosystem

In this section, we will describe the cryptosystem of Aggarwal et al. [1] (the AJPS cryptosystem). Let $N = 2^n - 1$, where n is a prime number[1], and let $R = \mathbb{Z}/N\mathbb{Z}$ be the integer ring modulo N. We define $w = \lfloor \sqrt{n}/2 \rfloor$ to be the upper bound on what we will consider "low weight".

[1]Numbers of the form $N = 2^n - 1$ with $n \in \mathbb{N}$ are called *Mersenne numbers*. If, additionally, $N = 2^n - 1$ is prime, it is called a *Mersenne prime*. For the purposes of the AJPS cryptosystem, N doesn't need to be prime, but n does.

We will identify a number in R with its binary representation. In this way, we can represent the elements of R by the elements of $\mathbb{F}_2^n$, with 1^n and 0^n both representing $0 \in R$, and all other elements of R having unique representatives in $\mathbb{F}_2^n$. For nonzero $a \in R$, denote by $|a|$ the Hamming weight of the unique binary representation of a, and define $|0| = 0$. Similarly, denote by $\Delta(a, b)$ the Hamming distance between the binary representations of a and b (using the representation 0^n for $0 \in R$). Note that it is not necessarily the case that $|ab|$ and $|a| \cdot |b|$ are equal nor that $|a + b|$ and $|a| + |b|$ are equal. However, we have the following.

Lemma 2.1 ([1])**.** *Let $a, b \in R$. Then*

(i) $|a + b| \leq |a| + |b|$,

(ii) $|ab| \leq |a| \cdot |b|$, *and*

(iii) if $a \neq 0$, *then* $|-a| = n - |a|$.

Elements of the ring R have the special property that for any $i \in \{0, \ldots, n-1\}$ and $a \in R$, the binary representation of $a \cdot 2^i \mod N$ is just a cyclic shift of the binary representation of a by i.

We now describe the AJPS cryptosystem [1] with public parameter n.

Key Generation Randomly choose two elements $f, g \in R$ of Hamming weight w, where g is invertible in R. Set $h = f/g$. The public key is h, and the secret key is g.

Encryption To encrypt a bit s, pick random $p, q \in R$ of Hamming weight at most w. Output the ciphertext $c = (-1)^s(ph + q) \in R$.

Decryption To decrypt c, compute $cg = (-1)^s(phg + qg) = (-1)^s(pf + qg)$. Since p, q, f and g all have Hamming weight $\leq w$, the n-bit string $pf + qg$ has Hamming weight $\leq 2w^2 < n/2$ by Lemma 2.1. Thus if $s = 0$, then $|cg| < n/2$. On the other hand, if $s = 1$, then $|-cg| < n/2$, so by Lemma 2.1, $|cg| > n - n/2 = n/2$. Thus, to decrypt c, output 0 if $|cg| < n/2$, and 1 otherwise.

To attack this cryptosystem, it suffices to solve the following problem:

Given $h \in R$, find $g \in R$ of Hamming weight w such that $|hg| = w$, assuming such a g exists.

Since multiplication by 2^i just shifts the binary representation of an element of R by i, if g is a solution to the above problem, then so is $2^i g$. Thus, if a solution exists, then a solution with the first bit set to 1 exists, and so we can restrict our attention to such solutions. Since a brute-force attack can find such a solution g in time $\binom{n-1}{w-1}$, to achieve security parameter λ, n and w must satisfy $\binom{n-1}{w-1} > 2^\lambda$ and $w < \sqrt{n}/2$. Our results, however, imply that a stronger condition is required to achieve λ-bit security.

2.2 Quantum search with variable costs

In this section we will introduce the quantum search algorithm, originally due to Grover [11] and later generalized [7, 8]. This algorithm searches a universe of size N for a particular *marked* item, given access to some procedure for checking if a given item is marked, using $O(\sqrt{N})$ calls to the checking procedure. We will also make use of an elegant variant of the quantum search algorithm due to Ambainis [3], that has better complexity when the cost of checking if a given item is marked varies by item.

Let U be some set of N objects, and let $\mathcal{C} : U \to \{0,1\}$ be some procedure, called the *checking* procedure, that outputs 1 when given a marked item, and suppose the complexity of the procedure $\mathcal{C}$ is C. Then there exists a quantum algorithm with complexity $\tilde{O}(\mathsf{C}\sqrt{N})$ that outputs $u \in U$ such that $\mathcal{C}(u) = 1$ with probability at least $2/3$, assuming such a u exists.

We may also consider the scenario in which the complexity of computing $\mathcal{C}(u)$ varies with u. Call this complexity $\mathsf{C}(u)$. Using the previously mentioned standard quantum search algorithm, we can search for $u \in U$ such that $\mathcal{C}(u) = 1$ in $\tilde{O}(\max_{u \in U} \mathsf{C}(u)\sqrt{N})$ steps. However, we can do better:

Theorem 2.2 (Quantum search with variable costs (Ambainis [3]))**.** *Let $\mathcal{C} : U \to \{0,1\}$ be any checking procedure. There exists a quantum algorithm that outputs $u \in U$ such that $\mathcal{C}(u) = 1$ with probability at least $2/3$ (assuming such a u exists), and has complexity*

$$\tilde{O}\left(\sqrt{\sum_{u \in U} \mathsf{C}(u)^2}\right),$$

where $\mathsf{C}(u)$ is the cost of computing $\mathcal{C}(u)$, and needn't be known in advance. We call this algorithm quantum search for $u \in U$ that satisfies $\mathcal{C}$*.*

In the case where $\mathsf{C}(u) = \mathsf{C}$ is constant, the algorithm from Theorem 2.2 has complexity $\tilde{O}(\mathsf{C}\sqrt{N})$, as in the standard quantum search algorithm.

3 Classical Meet-in-the-Middle Attack

3.1 Introduction

The "Meet-in-the-Middle attack" (MITM attack) is a well-known generic cryptographic attack that can be deployed against a variety of cryptosystems, often achieving an improved time complexity in breaking the system, at the cost of greater space complexity. It may have originated in [12].

To illustrate this attack, we give an example in the context of the knapsack problem, which can be described as follows. Given numbers $h_1, \ldots, h_n \in \mathbb{Z}$, find $g \in \mathbb{F}_2^n$ such that

$$\sum_{i=1}^{n} h_i g[i] = 0.$$

The MITMA idea is to split $\mathbb{F}_2^n = G_1 \oplus G_2$ into two equally-large subspaces of dimension $n/2$, where $G_1 = \{(g, 0^{\lceil n/2 \rceil}) : g \in \mathbb{F}_2^{\lfloor n/2 \rfloor}\}$ and $G_2 = \{(0^{\lfloor n/2 \rfloor}, g) : g \in \mathbb{F}_2^{\lceil n/2 \rceil}\}$. We calculate all numbers $H(g_1) = -\sum_i h_i g_1[i]$ for $g_1 \in G_1$ and store them in a database $\mathcal{D}$. This costs $2^{n/2}$ time and space, up to a $\mathrm{poly}(n)$ factor.

Hereafter, we calculate $H(g_2) = \sum_i h_i g_2[i]$ for $g_2 \in G_2$, and check whether the element $-H(g_2)$ is somewhere in $\mathcal{D}$, using a single database lookup. If so, then we have found a $g_1 \in G_1$ such that $H(g_2) = -H(g_1)$. Then $g_1 + g_2 \in \mathbb{F}_2^n$ is a solution. This search costs about $2^{n/2}$ database lookups, and $2^{n/2} \cdot \mathrm{poly}(n)$ time. This has much better time complexity than trying all combinations, which costs roughly 2^n time (but $\mathrm{poly}(n)$ space).

Description of our MITM attack on the AJPS system Given $h \in R$, we want to find $f, g \in R$, each of Hamming weight w, such that $h = f/g$ — or equivalently, such that $gh = f$. In other words, our task is, informally, to find $g \in \mathbb{F}_2^n$ of weight w such that $|gh|$ is small.

For $\alpha \in [0,1]$ to be specified later, we define

$$G_1^{(\alpha)} = \{(g, 0^{\lceil (1-\alpha)n \rceil}) : g \in \mathbb{F}_2^{\lfloor \alpha n \rfloor}, |g| = \lfloor \alpha w \rfloor\}$$
$$\text{and } G_2^{(\alpha)} = \{(0^{\lfloor \alpha n \rfloor}, g) : g \in \mathbb{F}_2^{\lceil (1-\alpha)n \rceil}, |g| = \lceil (1-\alpha)w \rceil\}.$$

We note that while $G_1^{(\alpha)} \oplus G_2^{(\alpha)}$ does not include all $g \in \mathbb{F}_2^n$ such that $|g| = w$, restricting to this set is without loss of generality, since, if $g = g_1 + g_2$ is a solution, meaning that both g and gh have weight w, then for any $z \in \{0, \ldots, n-1\}$, $2^z g$ is also a solution. This is because $2^z g$ just shifts the binary representation of g by z in a cyclic manner, so $2^z g$ and $2^z gh$ also have weight w. Thus, if there exists a solution, then there exists a solution in which g_1 and g_2 have weights $\lfloor \alpha w \rfloor$ and $\lceil (1-\alpha)w \rceil$, respectively.

The attack will begin by enumerating $(g_1, g_1 h)$ for all $g_1 \in G_1^{(\alpha)}$, after which we will search over $G_2^{(\alpha)}$ for some g_2 that is in collision with some $g_1 \in G_1^{(\alpha)}$, where, intuitively, we want to define g_1, g_2 to be in collision whenever the Hamming distance $\Delta(g_1 h, -g_2 h)$ is not much bigger than $2w$. The difficulty is that, given some value $-g_2 h$, while it would be easy to find a stored value of $g_1 h$ that is *equal* to $-g_2 h$, it is not immediately clear how to find such a stored value that is *close in Hamming distance* to $-g_2 h$.

Locality-sensitive hash functions Our solution is to use a simple form of locality-sensitive hashing [15]. Intuitively, a locality-sensitive hash should take the same value, with high probability, on two elements that are *close* with respect to some desired distance. In our case, for $\mathcal{B} = \{i_1, \ldots, i_B\} \subset [n]$ with $i_1 < \cdots < i_B$, define $\mathcal{H}_\mathcal{B} : \mathbb{F}_2^n \to \mathbb{F}_2^{|\mathcal{B}|}$ by $\mathcal{H}_\mathcal{B}(s_1, \ldots, s_n) = (s_{i_1}, \ldots, s_{i_B})$. We will use the function family $\mathcal{F}_B = \{\mathcal{H}_\mathcal{B} : |\mathcal{B}| = B\}$ for some B to be specified later. This works for our purposes, because if two strings are close in Hamming distance, then on a random small subset $\mathcal{B}$ of their bits, they are likely to agree.

Detailed description of algorithm Our Meet-in-the-Middle attack proceeds as follows:

1. Choose a uniformly random $\mathcal{H} \in \mathcal{F}_B$.
2. Initialize an empty hash table $\mathcal{D}$, with 2^B (initially empty) linked lists, one for each element in the range of $\mathcal{H}$.
3. For each $g_1 \in G_1^{(\alpha)}$:
   (a) Insert $(g_1, \mathcal{H}(g_1 h))$ into $\mathcal{D}$.
4. For each $g_2 \in G_2^{(\alpha)}$:
   (a) Look up $\mathcal{H}(-g_2 h)$ in $\mathcal{D}$, and let L be the resulting list of values g_1 such that $\mathcal{H}(g_1 h) = \mathcal{H}(-g_2 h)$.
   (b) For each g_1 in L:
      i. If $|g_1 + g_2| = w$ and $|(g_1 + g_2)h| = w$, output $g_1 + g_2$.

Analysis of algorithm We first argue that our algorithm succeeds in finding a solution if one exists. The next lemma shows that if $-g_2 h$ is uniformly random from R, and b is an arbitrary element of R with Hamming weight w, then (with high probability) $-g_2 h$ and $g_1 h = -g_2 h + b$ do not differ in much more than $2w$ bits of their binary representations — i.e., $\Delta(-g_2 h, g_1 h)$ is not much larger than $2w$.

c	0111 1110
a	0010 1111
b	0001 0001
$b \wedge c$	0001 0000

Table 1: Having a nonzero bit in $b \wedge c$ leads necessarily to an extra carry.

Lemma 3.1. *Let $a \in R = \mathbb{Z}/(2^n - 1)\mathbb{Z}$ be chosen uniformly at random, let $w \in \mathbb{N}$ and let $b \in R$ be any element such that $|b| = w$. Then, for $s > 0$, we have:*

$$\mathbb{P}\big[\Delta(a, a+b) > 2w + s\sqrt{w}\big] \leq \exp\left(-\frac{s^2}{8 + 4sw^{-1/2}}\right) + 2^{-n}. \tag{1}$$

Proof. We assume a to be chosen uniformly at random from $\mathbb{F}_2^n$, instead of from R. These two distributions $\mathcal{P}, \mathcal{Q}$ on the set of n-bit strings only differ by 2^{-n} with respect to the total variation distance

$$\frac{1}{2}\sum_{b \in \mathbb{F}_2^n} |\mathcal{P}[b] - \mathcal{Q}[b]| = \frac{1}{2}\left(\frac{1}{2^n} + (2^n - 1)\left(\frac{1}{2^n - 1} - \frac{1}{2^n}\right)\right) = 2^{-n},$$

which accounts for the 2^{-n}-term in the right-hand side of Equation (1).

Given a, b, we define the carry element $c_n c_{n-1} \ldots c_1 = c \in R$ by $c = (a + b) \oplus (a \oplus b)$. One can show that c equals the 'carry vector' that one puts above the sum of a and b when doing addition on a blackboard (see Table 1). Note that $a \oplus b \oplus c = a + b$, implying that $\Delta(a, a + b) = |b \oplus c| = |b| + |c| - |b \wedge c| = w + |c| - |b \wedge c|$.

Let $G \sim \text{Geo}(1/2)$ be a random variable with the geometric distribution with parameter $p = 1/2$, having values in $\{0, 1, \ldots\}$. Then the random variable $C = |c|$ is majorized by $S + |b \wedge c|$, where $S = G + \ldots + G$ (w times) is the sum of w independent and identically geometrically distributed variables. Indeed, every added nonzero bit from b gives rise to a geometric distribution, and, might increase the number of carries of one of the next nonzero bit of b by one. If the latter happens — say $c_i = b_i = 1$ — then $|b \wedge c|$ is increased by one, see Table 1.

Therefore, $\Delta(a, a + b) = w + |c| - |b \wedge c| \leq w + \underbrace{G + \ldots + G}_{w \text{ times}} = w + S$. This in turn means that

$$\begin{aligned}\mathbb{P}\big[\Delta(a, a+b) > 2w + s\sqrt{w}\big] &\leq \mathbb{P}\big[S > w + s\sqrt{w}\big] = \mathbb{P}\big[\text{Bin}(2w + s\sqrt{w}, 1/2) < w\big] \\ &\leq \exp\left(-\frac{(s\sqrt{w})^2}{8w + 4s\sqrt{w}}\right) = \exp\left(-\frac{s^2}{8 + 4sw^{-1/2}}\right).\end{aligned}$$

The first equality holds because the event '$S > w + s\sqrt{w}$' is the same as the event that in a sequence of $2w + s\sqrt{w}$ fair coin flips, there are fewer than w successes. We upper bound the probability of the latter by Chernoff's inequality. □

Heuristic 3.2. *The above lemma still holds when setting $a = hg_1$ and $b = f$ for f, g, h distributed as in the AJPS cryptosystem.*

Remark. The above heuristic is corroborated by experiments, see Appendix A. More concretely, for primes $n \leq 2000$, it holds that $\Delta(-g_2 h, g_1 h) \leq 2w - 1$ for more than half of the keys, and that $\Delta(-g_2 h, g_1 h) \leq 2w + 7$ for about 90% of the keys.

We are now ready to analyze the space and time complexity of the algorithm. We will set $\alpha = 1/2$. We can see immediately that the algorithm requires $\tilde{O}(|G_1^{(\alpha)}|) = \tilde{O}\left(\binom{n/2}{w/2}\right)$ space.

To analyze the time complexity, we first note that Step 3 of the algorithm costs $|G_1^{(\alpha)}|$ insertions into the data structure, so the cost is $\tilde{O}(|G_1^{(\alpha)}|) = \tilde{O}\left(\binom{n/2}{w/2}\right)$. The loop in Step 4 runs $|G_2^{(\alpha)}|$ times, and the iteration corresponding to some $g_2 \in G_2^{(\alpha)}$ costs approximately $1 + \ell(g_2)$, where $\ell(g_2) = |\{g_1 \in G_1^{(\alpha)} : \mathcal{H}(g_1 h) = \mathcal{H}(-g_2 h)\}|$. The total cost of Step 4 is thus at most:

$$|G_2^{(\alpha)}| + \sum_{g_2 \in G_2^{(\alpha)}} \ell(g_2).$$

We can rewrite the above as

$$\begin{aligned} |G_2^{(\alpha)}| + & \sum_{v \in \mathbb{F}_2^B} |\{(g_1, g_2) \in G_1^{(\alpha)} \times G_2^{(\alpha)} : \mathcal{H}(g_1 h) = \mathcal{H}(-g_2 h) = v\}| \\ &= |G_2^{(\alpha)}| + |\{(g_1, g_2) \in G_1^{(\alpha)} \times G_2^{(\alpha)} : \mathcal{H}(g_1 h) = \mathcal{H}(-g_2 h)\}|. \end{aligned}$$

Heuristic 3.3. *For every fixed $\mathcal{H} \in \mathcal{F}_B$, with high probability over g and f as chosen in the AJPS system, we have $|\{(g_1, g_2) \in G_1^{(\alpha)} \times G_2^{(\alpha)} : \mathcal{H}(g_1 h) = \mathcal{H}(-g_2 h)\}| \approx |G_1^{(\alpha)}| \cdot |G_2^{(\alpha)}| 2^{-B}$.*

Remark. The above heuristic is obtained by considering all $\mathcal{H}(g_1 h)$ and $\mathcal{H}(g_2 h)$ values as independent random uniform strings of B bits. The validity of this heuristic is confirmed by the experiments presented in Appendix A.1.

Let $g = g_1^* + g_2^*$ be a solution. The algorithm will only find this g if $\mathcal{H}(g_1^* h) = \mathcal{H}(-g_2^* h)$, in which case we say $\mathcal{H}$ is *good* for g. By Lemma 3.1 and assuming Heuristic 3.2, $\Delta(g_1^* h, -g_2^* h) \leq 2w + s\sqrt{w}$ happens with high probability, for a fixed constant s. So, assuming $\Delta(g_1^* h, -g_2^* h) \leq 2w + s\sqrt{w}$, the hash function $\mathcal{H}$ is good with probability at least $p(B) = \frac{\binom{n-2w-s\sqrt{w}}{B}}{\binom{n}{B}}$ where the probability is over the function family $\mathcal{F}_B = \{\mathcal{H}_{\mathcal{B}} : |\mathcal{B}| = B\}$.

Lemma 3.4. *Under the above heuristics, setting $\alpha = 1/2$ and $B = \lceil \log_2 \binom{n/2}{w/2} \rceil$ ($\approx \frac{w}{2} \log(n/w) + O(w)$), the time complexity of the algorithm is $\tilde{O}\left(\sqrt{\binom{n}{w}}\right)$.*

Proof. Ignoring polylogarithmic factors, the complexity of the algorithm equals $|G_1^{(\alpha)}| + |G_2^{(\alpha)}| + |G_1^{(\alpha)}| \cdot |G_2^{(\alpha)}| 2^{-B}$. Note that $|G_2^{(\alpha)}| 2^{-B} \leq 1$, by the choice of B and the fact that $|G_2^{(\alpha)}| = \binom{n/2}{w/2}$. Therefore the complexity of steps 1-4 equals $2|G_1^{(\alpha)}| + |G_2^{(\alpha)}| = 3\binom{n/2}{w/2} = \tilde{O}\left(\sqrt{\binom{n}{w}}\right)$. To achieve constant success probability, we repeat the algorithm $1/p(B)$ times, which is, as we will show, polynomial in n. We use the identity $\ln \binom{m}{\ell} = \ell \ln(m/\ell) + \ell + O(\ln m)$ whenever $\ell = \tilde{O}(\sqrt{m})$, and the fact that $w^2 \approx n/4$. We have:

$$\begin{aligned} \ln \frac{1}{p(B)} &= \ln \binom{n}{B} - \ln \binom{n - 2w - s\sqrt{w}}{B} = -B \ln \left(\frac{n - 2w - s\sqrt{w}}{n} \right) + O(\ln n) \\ &= (1 + o(1)) \frac{2wB}{n} + O(\ln n) = (1 + o(1)) \frac{w^2}{n} \ln(n/w) + O(\ln n) = O(\ln n). \end{aligned}$$

□

4 Quantum Meet-in-the-Middle Attack

We now present our quantum meet-in-the-middle attack. The first example of a quantum meet-in-the-middle algorithm was the collision finding algorithm of Brassard, Høyer and Tapp [9]. Similar ideas were later used in a quantum algorithm for the subset sum problem [5], which has a similar structure to the algorithm presented here. One difference in our new algorithm is the use of Ambainis's variable-cost quantum search algorithm [3], described in Section 2.2.

The algorithm presented in this section requires time and space $\tilde{O}\left(\binom{n}{w}^{1/3}\right)$. The bulk of the memory required for this quantum algorithm must be *quantum accessible*, meaning it does not need to be able to store a quantum state, but must be accessible in superposition. Only $O(n)$ of the space used by the algorithm must be fully quantum memory, capable of being in an arbitrary superposition.

The quantum algorithm presented and analyzed in this section is then very similar to the classical MITM attack, except we use quantum search to search over all $g_2 \in G_2^{(\alpha)}$, and then since the complexity of this step of the algorithm decreases in the quantum case, it is optimal to use $\alpha = 1/3$ rather than $\alpha = 1/2$.

Detailed description of algorithm Our quantum MITM attack proceeds as follows:

1. Choose a uniformly random $\mathcal{H} \in \mathcal{F}_B$.
2. Initialize an empty hash table $\mathcal{D}$.
3. For each $g_1 \in G_1^{(\alpha)}$:
   (a) Insert $(g_1, \mathcal{H}(g_1 h))$ into $\mathcal{D}$.
4. Quantum search (using Theorem 2.2) for $g_2 \in G_2^{(\alpha)}$ that satisfies the following checking procedure:
   (a) Look up $\mathcal{H}(-g_2 h)$ in $\mathcal{D}$, and let $L(g_2)$ be the resulting list of values g_1 such that $\mathcal{H}(g_1 h) = \mathcal{H}(-g_2 h)$.
   (b) Quantum search for g_1 in L that satisfies the following checking procedure:
      i. If $|g_1 + g_2| = w$ and $|(g_1 + g_2)h| = w$, output 1.

Analysis of algorithm We will use $\alpha = 1/3$. The algorithm requires $|G_1^{(\alpha)}| = \binom{\alpha n}{\alpha w} = \binom{n/3}{w/3}$ quantum accessible memory, and $O(\log |G_2^{(\alpha)}|) = O(n)$ quantum memory.

In order to upper bound the time complexity, we will make use of Heuristic 3.3 with $\alpha = 1/3$. Then we have the following.

Lemma 4.1. *Assuming Heuristic 3.2 and Heuristic 3.3 with $\alpha = 1/3$, setting $B = \lceil \log_2 \binom{n/3}{w/3} \rceil$, the time complexity of the algorithm is $\tilde{O}\left(\binom{n}{w}^{1/3}\right)$.*

Proof. As in the classical algorithm, the time complexity of Steps 1 to 3 of the quantum algorithm is $\tilde{O}(|G_1^{(\alpha)}|) = \tilde{O}\left(\binom{\alpha n}{\alpha w}\right)$, which, in this case, is $\tilde{O}\left(\binom{n/3}{w/3}\right)$. For a particular $g_2 \in G_2^{(\alpha)}$, Steps 4a and

4b together cost (neglecting negligible factors) $1+\sqrt{\ell(g_2)}$, so using variable cost quantum search, as described in Section 2.2, the total cost of Step 4 is

$$\sqrt{\sum_{g_2\in G_2^{(\alpha)}}(1+\sqrt{\ell(g_2)})^2} = O\left(\sqrt{|G_2^{(\alpha)}|}+\sqrt{\sum_{g_2\in G_2^{(\alpha)}}\ell(g_2)}\right).$$

By Heuristic 3.3 and the choice of B, we have $\sum_{g_2\in G_2^{(\alpha)}}\ell(g_2)\approx |G_1^{(\alpha)}|\cdot|G_2^{(\alpha)}|2^{-B}\leq|G_2^{(\alpha)}|$. Thus, the total complexity of steps 1-4 of the attack is $O\left(\sqrt{|G_2^{(\alpha)}|}\right)=O\left(\sqrt{\binom{2n/3}{2w/3}}\right)=\tilde{O}\left(\binom{n/3}{w/3}\right)$. Finally, as in Section 3, $\mathcal{H}$ is *good* with probability $p(B)=\frac{\binom{n-2w-s\sqrt{w}}{B}}{\binom{n}{B}}$. To achieve constant success probability, we repeat $1/p(B)$ times, which is polynomial in n by a similar reasoning as in Lemma 3.4. $\square$

5 Analysis of the Beunardeau et al. attack

Within a week of the publication of the AJPS cryptosystem [1], an experimental attack was proposed by Beunardeau et al. [6]. This attack exploits the fact that a certain lattice, derived from the public key of the AJPS cryptosystem and two well-chosen partitions, has very short vectors. One of these short vectors, which can be found by means of the LLL lattice reduction algorithm [19], represents the private key.

Although Beunardeau et al. do not give a clear asymptotic estimate of the complexity of their attack, they do suggest tentatively that it might run in time $2^{2w}n^{O(1)}$, where $w=\lfloor\sqrt{n}/2\rfloor$ is the Hamming weight of secret key $g\in R$ [6, §2.2]. More specifically, once a partition is chosen, the attack runs in polynomial time $n^{O(1)}$, and the probability that it is successful should be about 2^{-2w}.

Remark. Note that this probability is taken only over the randomness of the secret key. It is not obvious that one can amplify the success probability for a fixed key up to a constant by repeating the attack with 2^{2w} different partitions. Indeed, there could be certain keys that are caught by a fraction of partitions significantly smaller than 2^{-2w}.

In this section, we propose an analysis of a simplified version of their attack. Using standard lattice heuristics we can argue that, for each pair of partitions, the probability that a secret key will be found by applying LLL on the derived lattice equals $(\frac{1}{2}-c\left(\frac{d}{w}\right)^2+o(1))^{2w}$, where d is the lattice dimension, and c is a very small constant, say $1/140$. The lattice dimension d corresponds to the number of blocks in a partition of the bits of f and g. While in theory we can choose d between 2 and $O(w)$, in order to find f and g for a particular h, we will generally need to choose d as large as $\Omega(w)$. We discuss this more at the end of Section 5.3. While asymptotically slightly different from the tentative conclusion of [6], this analysis certainly does not contradict the fact that this attack is quite efficient in practice, and remains the best known attack (better than our MITM attack).

We remark that one could also replace LLL with a perfect SVP-oracle to raise the success probability to $(\frac{1}{2}+o(1))^{2w}$, but this would increase the running time of the lattice reduction step to $2^{\Theta(d)}$. Namely, for partitions of size $d=\Theta(w)$ the ratio of the cost over success probability remains at least $2^{(2+\delta)w+o(w)}$ for a fixed $\delta>0$.

Finally, we remark that this attack can also be sped up with a quantum computer. If, for a particular fixed key g, the probability that a sampled partition allows the LLL subroutine to find the secret key is p, then there is a quantum algorithm that finds the key in only $\sqrt{1/p}$ calls to the subroutine, compared to the $1/p$ calls required by a classical algorithm. So under the heuristic assumption that $p\approx 2^{-2w}$, there is a quantum algorithm that recovers the key in time $\approx 2^w$.

Unfortunately, despite some effort, we have not been able to answer the question left open by Beunardeau et al.: "Are there classes of public keys that are harder to recover using this lattice attack, and if so, which ones?"

5.1 Partitions

In this section, we show how partitioning of $[n] = \{0, \ldots, n-1\}$ can lead to a short representation of the secret key $g \in R = \mathbb{Z}/N\mathbb{Z}$. The overall idea is to write g as a binary string in $\mathbb{F}_2^n$, as before. Since g has a low Hamming weight, one can imagine the one-valued bits scattered sparsely among the n possible positions. One then chooses interval-like subsets of $[n]$ such that, with any luck, each one-valued bit falls in the right-half of one of these subsets. In that case, each subset of $[n]$ in the partition corresponds to a binary substring of g representing a "small" number. Consequently, the array of these numbers can be considered as a short representation of g. An example is depicted in Figure 1.

Remark. Because of the bit-wise arithmetic in R, it is natural to consider interval-like partitions only. An interval-like partition P of $[n]$ consists of subsets that are of the form $\{a, a+1, a+2, \ldots, b-1, b\}$ for $a, b \in [n]$, i.e., subsets without 'gaps'. Due to the fact that multiplication by 2^i in R simply shifts all binary representations by i, we also allow subsets of the form $\{a, a+1, \ldots, n-2, n-1, 0, 1, \ldots, b\}$.

Remark. Formally, our approach is slightly different from the one of Beunardeau et al. [6]. Namely, they define partitions with black and white blocks, hoping that all 1-valued bits of the secret key fall into the white blocks of the partitions. It turns out, however, that the black blocks do not play any role in the construction of the lattice related to this partition. Therefore, we prefer to omit the black partitions in our approach. This alteration has no algorithmic impact and is merely an editorial choice simplifying the analysis.

g	00100000000001000010
Partition	$\{19, 18, 17, 16\}, \{15, 14, 13, 12\}, \{11, \ldots, 6\}, \{5, \ldots, 0\}$
g partitioned in a "good" way	0010 0000 000001 000010
Array of decimal numbers representing g	$[2, 0, 1, 2]$, $g = 2 \cdot 2^{16} + 0 \cdot 2^{12} + 1 \cdot 2^6 + 2 \cdot 2^0$

Figure 1: Example partition of g with Hamming weight 3.

5.2 Lattice Reduction

Lattice construction Given any two interval-like partitions $P = \{P_1, \ldots, P_k\}$, $Q = \{Q_1, \ldots, Q_\ell\}$ of $[n]$ and a public key $h \in R$. Let p_i, q_i be the least elements of P_i, Q_i respectively. Then, one can consider the following lattice.

$$\mathcal{L}_{P,Q,h} = \left\{ (x_1, \ldots, x_k, y_1, \ldots, y_\ell) \;\middle|\; h \cdot \sum_{i=1}^{k} 2^{p_i} \cdot x_i - \sum_{j=1}^{\ell} 2^{q_i} \cdot y_i \equiv 0 \bmod N \right\}.$$

This lattice $\mathcal{L}_{P,Q,h}$ has determinant $\Delta = N$ and dimension $d = k + \ell$. Namely, as $\mathcal{L}_{P,Q,h}$ is a sublattice of $\mathbb{Z}^d$, we have $\det(\mathcal{L}_{P,Q,h}) = \det(\mathbb{Z}^d) \cdot [\mathbb{Z}^d : \mathcal{L}_{P,Q,h}] = N$, since $\det(\mathbb{Z}^d) = 1$ and the group index equals N.

This lattice contains vectors of the form $(0, \ldots, 0, 2^m, -1, 0, \ldots, 0)$, for some m, which we will call 'structural' vectors. These structural vectors have length $\sqrt{4^{|P_i|} + 1}$ and $\sqrt{4^{|Q_i|} + 1}$. For example, $(2^{p_2 - p_1}, -1, 0, \ldots, 0) \in \mathcal{L}_{P,Q,h}$ is a structural vector which is easily seen to have the described length, observing that $p_2 - p_1 = |P_1|$. Applying this example for every two subsequent variables of the same kind, one arrives at all structural vectors.

Definition 5.1 (Secret vector)**.** Let $h = f/g \in R$ be as in the AJPS-cryptosystem, suppose $P = \{P_1, \ldots, P_k\}$ and $Q = \{Q_1, \ldots, Q_\ell\}$ are interval-like partitions of $[n]$ and denote $p_i = \min P_i$ and $q_j = \min Q_j$. We define the *secret vector*

$$s := (g_1, \ldots, g_k, f_1, \ldots, f_\ell) \in \mathcal{L}_{P,Q,h},$$

where $0 \leq g_i < 2^{|P_i|}$ and $0 \leq f_j < 2^{|Q_j|}$ are the unique natural numbers such that $\sum_{i=1}^{k} g_i \cdot 2^{p_i} = g$ and $\sum_{j=1}^{\ell} f_j \cdot 2^{q_j} = f$.

Remark. The vector s is actually just the concatenation of the vectors $(g_1, \ldots, g_k)$ and $(f_1, \ldots, f_\ell)$, which are constructed from g, P and f, Q respectively as in Figure 1.

Applying LLL Let us recall the guarantees provided by the LLL algorithm.

Lemma 5.1 ([19, 20])**.** *For any $\gamma > \sqrt{4/3}$, the LLL$_\gamma$-algorithm applied to a d-dimensional lattice L returns, within polynomial time, a basis $(b_1, \ldots, b_d)$ of L satisfying*

- $\|b_1\| \leq \mathrm{HF}(L) := \gamma^{(d-1)/2} \cdot \Delta_L^{1/d}$ *(Hermite factor bound);*
- $\|b_1\| \leq \mathrm{AF}(L) := \gamma^{d-1} \cdot \lambda_1(L)$ *(Approximation factor bound).*

where $\lambda_1(L)$ is the length of a shortest nonzero vector of L, and Δ_L is the determinant of the lattice L.

In practice, LLL performs much better. For cryptanalytic purposes, one often assumes $\gamma = 1.04$, which is corroborated by many experiments [21]. In the current analysis, this practical value of γ will be used.

The inequalities in Lemma 5.1 give rise to two so-called *regimes* of LLL$_\gamma$, the *Hermite regime* and the *Approximation regime.* A lattice L lies in the Hermite regime when $\mathrm{HF}(L) \leq \mathrm{AF}(L)$, and lies in the Approximation regime whenever $\mathrm{AF}(L) < \mathrm{HF}(L)$. One distinguishes these two cases because the output of LLL differs significantly between the regimes. This effect is most prominent when a single, unique short vector causes a lattice to be in the Approximation regime; in that case LLL typically outputs this particular short vector [10, §3.3].

One would like to have that this last scenario holds for the lattice $\mathcal{L}_{P,Q,h}$ and the secret vector s. So, informally, one wishes to have no vectors in $\mathcal{L}_{P,Q,h}$ that are shorter than usual except for the secret vector s. One obstacle could be that the structural vectors are too short, causing s not to be unique. However, we will rule out this possibility by comparing the lengths of these structural vectors to the *Gaussian heuristic* of $\mathcal{L}_{P,Q,h}$.

The Gaussian heuristic uses a geometric argument to estimate the length of the shortest vector of a lattice [10]. For d-dimensional lattices L one expects $\lambda_1(L) \approx \sqrt{d/(2\pi e)} \cdot \Delta_L^{1/d}$, according to this heuristic. Applying this to the lattice of interest, one obtains $\lambda_1(\mathcal{L}_{P,Q,h}) \approx \sqrt{n/(2\pi e)} \cdot 2^{\frac{n}{d}}$. Recall that the structural vectors have approximate length $2^{|P_i|}$ and $2^{|Q_i|}$. So, whenever $|P_i|, |Q_i| > n/d + \Theta(\log n)$, we have $2^{|P_i|}, 2^{|Q_i|} > \sqrt{d/(2\pi e)} \cdot 2^{\frac{n}{d}}$. So, in this case, the structural vectors are not shorter than the estimate of the Gaussian heuristic and hence longer than the secret vector s. Note that the average size of $|P_i|, |Q_i|$ is $2n/d$, meaning that this constraint is not so restrictive.

Therefore, we assume the following heuristic.

Heuristic 5.2. *The attack of Beunardeau et al. is successful in recovering the secret vector s if s (as in Definition 5.1) is the shortest vector and causes the lattice $\mathcal{L}_{P,Q,h}$ to fall into the Approximation regime, i.e.,* $\mathrm{AF}(\mathcal{L}_{P,Q,h}) < \mathrm{HF}(\mathcal{L}_{P,Q,h})$.

From the above heuristic we can deduce that the lattice attack succeeds if

$$\|s\| \cdot \gamma^{d-1} < \gamma^{(d-1)/2} \cdot 2^{n/d} = \mathrm{HF}(\mathcal{L}_{P,Q,h}).$$

Moreover, according to the study of Albrecht et al. [2] on the behavior of LLL for unique-SVP instances, this condition should be essentially tight. More precisely, we expect the attack to fail with overwhelming probability when $\mathrm{AF}(\mathcal{L}_{P,Q,h}) > O(\sqrt{d}) \cdot \mathrm{HF}(\mathcal{L}_{P,Q,h})$.

The metric bounds $\|s\|_\infty \le \|s\| \le \sqrt{d} \cdot \|s\|_\infty$ imply that the attack passes when $\sqrt{d} \cdot \|s\|_\infty < \gamma^{-(d-1)/2} \cdot 2^{\frac{n}{d}}$ and is expected to fail when $\|s\|_\infty > O(\sqrt{d}) \cdot \gamma^{-(d-1)/2} \cdot 2^{\frac{n}{d}}$. Since $\|s\|_\infty = \max\{g_i, f_i\}$, we can write $\|s\|_\infty = 2^r$, where r is the bit size of the maximum of the g_i and f_i. Putting this in the inequalities and taking base-two logarithms, yields the following. The attack succeeds whenever $r < \frac{n}{d}(1 - \delta_1 - \delta_2)$ and is expected to fail when $r > \frac{n}{d}(1 - \delta_1 + \delta_2 + O(d/n))$, where

$$\delta_1 = \frac{d(d-1) \cdot \log_2(\gamma)}{2n} \quad \text{and} \quad \delta_2 = \frac{d \cdot \log_2(d)}{2n}.$$

5.3 Success probability analysis

Let P and Q be partitions with block sizes at least $n/d + \Theta(\log n)$, where $d = k + \ell$ with $k = |P|$ and $\ell = |Q|$. We analyze the success probability of the lattice attack with respect to random f and $g \in R$ both of Hamming weight $w = \lfloor \sqrt{n}/2 \rfloor$.

From the previous section, we found that it suffices that the non-zero bits of f and g fall in the rightmost r bits of each block, in order to make LLL find the secret vector. So, for g, the total number of bits that are allowed to be one equals $\ell \cdot r$. Therefore, we can approximate the probability of the bits of f and g all falling in the good region by

$$\left(\frac{\ell \cdot r}{n}\right)^w \left(\frac{k \cdot r}{n}\right)^w.$$

Putting in the upper and lower bound for r, we obtain an upper and lower bound for the success probability p of the attack.

$$\left(\frac{\ell k(1 - \delta_1 - \delta_2)}{d^2}\right)^w < p < \left(\frac{\ell k(1 - \delta_1 + \delta_2 + O(d/n))}{d^2}\right)^w.$$

In order to maximize the above probability, we will assume that $k = \ell = d/2$ and $d = O(w)$. Namely, the fraction $\frac{\ell k}{(\ell+k)^2}$ attains its maximum at $\ell = k = d/2$. Recalling $w^2 \approx n/4$, we obtain $\delta_1 = \frac{d(d-1)}{2n} \cdot \log_2(\gamma) \approx \frac{1}{8}\left(\frac{d}{w}\right)^2 \cdot \log_2(\gamma)$ and $\delta_2 = o(1)$ as $n \to \infty$. Therefore

$$\left(\frac{1 - \delta_1 - o(1)}{2}\right)^{2w} < p < \left(\frac{1 - \delta_1 + o(1)}{2}\right)^{2w}.$$

Thus, assuming Heuristic 5.2, the success probability of LLL recovering a randomly chosen AJPS secret key pair $(f, g) \in R^2$ from the lattice $\mathcal{L}_{P,Q,h}$ (where $h = f/g$), is roughly $\left(\frac{1}{2} - c\left(\frac{d}{w}\right)^2 + o(1)\right)^{2w}$, where $c = \log_2(\gamma)/8 = \log_2(1.04)/8 \approx 1/140$. This probability value suggests that one should start with partitions with a small number of blocks, exploiting both the low dimension m of the lattice $\mathcal{L}_{P,Q,h}$ and a slightly larger success probability. Note, however, that it is not likely that the secret key $s = (g, f)$ will be recovered in this stage; the smaller d is, the fewer possible partitions there are, so the need to sample new partitions will require us to increase d to $\Omega(w)$ for most keys.

Replacing LLL by an SVP-oracle. If one replaces LLL by an SVP-oracle, the success condition from Heuristic 5.2 needs to be amended. Instead, the attack would be successful when s is the shortest vector of L. Heuristically this is the case if and only if s is shorter than what is predicted by the Gaussian Heuristic $\lambda_1(L) \approx \sqrt{d/2\pi e} \cdot \Delta_L^{1/d}$. Using a similar analysis, this leads to a success probability of $2^{-2w+o(1)}$. Note however that the best SVP-solvers [4] need time $(3/2)^{d/2}$, which would increase the overall complexity of the attack significantly.

5.4 Generalization to scaled partitions

The attack that is treated above is a simplification of the attack of Beunardeau et al.; essentially we omitted a 'scaling' technique [6, §2.2, 'Trying partitions']. This particular technique allows the variation of partition sizes and the fraction of each partition block that must consist of leading 0s.

The lattice $\mathcal{L}_{P,Q,h}$ scaled by the vector $\sigma = (\sigma_1, \ldots, \sigma_k, \sigma_1', \ldots, \sigma_\ell') \in \mathbb{R}^d$ can be defined explicitly as follows.

$$\mathcal{L}_{P,Q,h}^{\sigma} = \left\{ (\sigma_1 x_1, \ldots, \sigma_k x_k, \sigma_1' y_1, \ldots, \sigma_\ell' y_\ell) \;\middle|\; h \cdot \sum_{i=1}^{k} 2^{p_i} \cdot x_i - \sum_{j=1}^{\ell} 2^{q_i} \cdot y_i \equiv 0 \bmod N \right\}.$$

Allocating less weight σ_i to the content x_i of a certain partition P_i lets a lattice reduction algorithm tolerate larger values x_i; this means that the required fraction of leading 0s in this partition is diminished. This technique implies more freedom in choosing block sizes and required fractions of leading 0s.

Note, however, that scaling the entire lattice $L \mapsto cL$ by a constant won't affect the attack at all. Therefore, one might require, without loss of generality, that $\prod_{i=1}^{k} \sigma_i \prod_{j=1}^{\ell} \sigma_j' = 1$. This implies that the increase and decrease of the fractions of leading 0s of the blocks are in an equilibrium, not affecting the total region where non-zero bits are allowed.

So, this extension possibly increases the number of public keys that can be broken but does not affect the running time nor the success probability of the attack. Even considering this generalization, we were not able to prove that this improved attack could recover *every* key with constant probability in time $2^{(2+\delta)w+o(1)}$ for some small constant $\delta > 0$.

Acknowledgments

The authors wish to thank David Naccache, Antoine Joux and Marc Beunardeau for helpful discussions. LD is supported by a NWO Veni Innovational Research Grant under project number 639.021.645. SJ is supported by an NWO WISE Grant and an NWO Veni Innovational Research Grant under project number 639.021.752. RdW is partially supported by ERC Consolidator Grant 61530-QPROGRESS.

References

[1] D. Aggarwal, A. Joux, A. Prakash, and M. Santha. A new public-key cryptosystem via Mersenne numbers. Cryptology ePrint Archive, Report 2017/481, 2017. `http://eprint.iacr.org/2017/481`.

[2] M. R. Albrecht, F. Göpfert, F. Virdia, and T. Wunderer. Revisiting the Expected Cost of Solving uSVP and Applications to LWE. Cryptology ePrint Archive, Report 2017/815, 2017. `https://eprint.iacr.org/2017/815`.

[3] A. Ambainis. Quantum search with variable times. *Theory of Computing Systems*, 47(3):786–807, Oct 2010.

[4] A. Becker, L. Ducas, N. Gama, and T. Laarhoven. New directions in nearest neighbor searching with applications to lattice sieving. In *Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2016)*, pages 10–24, 2016.

[5] D. J. Bernstein, S. Jeffery, T. Lange, and A. Meurer. Quantum algorithms for the subset sum problem. In *Proceedings of the 5th International Conference on Post-Quantum Cryptography (PQCrypto 2013)*, pages 16–33, 2013.

[6] M. Beunardeau, A. Connolly, R. Géraud, and D. Naccache. On the hardness of the Mersenne low Hamming ratio assumption. In *Progress in Cryptology – LATINCRYPT 2017*, 2017. Available at `http://eprint.iacr.org/2017/522`.

[7] M. Boyer, G. Brassard, P. Høyer, and A. Tapp. Tight bounds on quantum searching. *Fortschritte der Physik*, 46(4-5):493–505, 1998.

[8] G. Brassard, P. Høyer, M. Mosca, and A. Tapp. Quantum amplitude amplification and estimation. In *Quantum Computation and Quantum Information: A Millennium Volume*, volume 305 of *AMS Contemporary Mathematics Series Millennium Volume*, pages 53–74. AMS, 2002.

[9] G. Brassard, P. Høyer, and A. Tapp. Quantum algorithm for the collision problem. *ACM SIGACT News*, 28:14–19, 1997. `arXiv:quant-ph/9705002`.

[10] N. Gama and P. Q. Nguyen. Predicting lattice reduction. *Advances in Cryptology – EUROCRYPT 2008*, pages 31–51, 2008.

[11] L. K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the 28th Annual ACM Symposium on the Theory of Computing (STOC 1996)*, pages 212–219, 1996.

[12] M. Hellman. A cryptanalytic time-memory trade-off. *IEEE Transactions on Information Theory*, 26(4):401–406, July 1980.

[13] J. Hoffstein, J. Pipher, and J. H. Silverman. NTRU: A ring-based public key cryptosystem. In *International Algorithmic Number Theory Symposium*, pages 267–288. Springer, 1998.

[14] N. Howgrave-Graham. A hybrid lattice-reduction and meet-in-the-middle attack against NTRU. *Advances in Cryptology – CRYPTO 2007*, pages 150–169, 2007.

[15] P. Indyk and R. Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proceedings of 30th Symposium on Theory of Computing (STOC 1998)*, 1998.

[16] T. Laarhoven. *Search problems in cryptography.* PhD thesis, Eindhoven University of Technology, `http://www.thijs.com/docs/phd-final.pdf`, 2015.

[17] T. Laarhoven. Sieving for shortest vectors in lattices using angular locality-sensitive hashing. In *Annual Cryptology Conference*, pages 3–22. Springer, 2015.

[18] T. Laarhoven, M. Mosca, and J. van de Pol. Finding shortest lattice vectors faster using quantum search. *Designs, Codes and Cryptography*, 77(2-3):375–400, 2015.

[19] A. K. Lenstra, H. W. Lenstra, and L. Lovasz. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261:515–534, 1982.

[20] P. Q. Nguyen. Hermite's constant and lattice algorithms. In *The LLL Algorithm*, pages 19–69. Springer, 2009.

[21] P. Q. Nguyen and D. Stehlé. LLL on the average. *Algorithmic Number Theory*, pages 238–256, 2006.

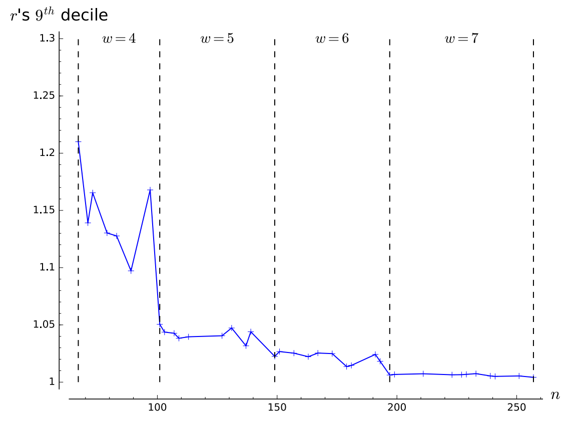
A Experiments

Since our MITM attack is not fully provable due to the presence of Heuristics 3.2 and 3.3, we provide some experimental verifications. The `python` scripts of those experiments are available at `https://github.com/lducas/MiTM-Mersenne`.

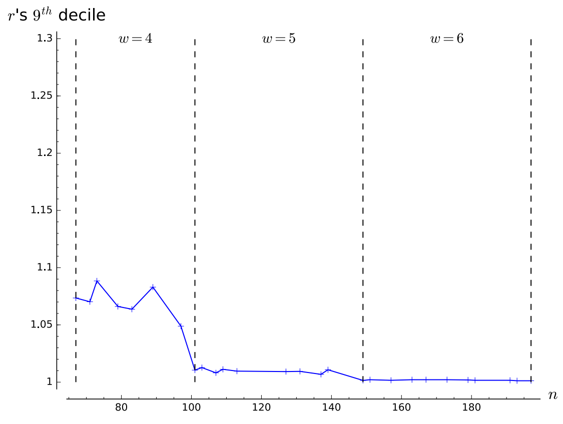
One tweak in our implementation is that when w is odd, we do not split our space exactly into two equal parts. Instead we choose $w_1 = \lfloor w/2 \rfloor$, $w_2 = w - w_1$, and then choose n_1, n_2, such that $\binom{n_1}{w_1} \approx \binom{n_2}{w_2}$. We will also simulate the quantum case, and choose $w_1 = \lceil w/3 \rceil$, $w_2 = w - w_1$, and then choose n_1, n_2, such that $\binom{n_1}{w_1}^2 \approx \binom{n_2}{w_2}$. In both the classical and quantum case, we set $B = \lfloor \log_2 \binom{n_1}{w_1} \rfloor$.

A.1 Verification of Heuristic 3.3

We recall that Heuristic 3.3 states that the count c of collisions $c = |\{(g_1, g_2) \in S_1 \times S_2 : \mathcal{H}(g_1 h) = \mathcal{H}(-g_2 h)\}|$ is approximately given by $c' = |S_1| \cdot |S_2| 2^{-B}$. We measure the ratio $r = c/c'$ experimentally, over 100 samples for each dimension n. Sporadically, this ratio may get as large as 3, yet for 90% of the experiments, it was very close to 1. Figure 2 below shows the 9th decile of r as n grows.



Classical setting



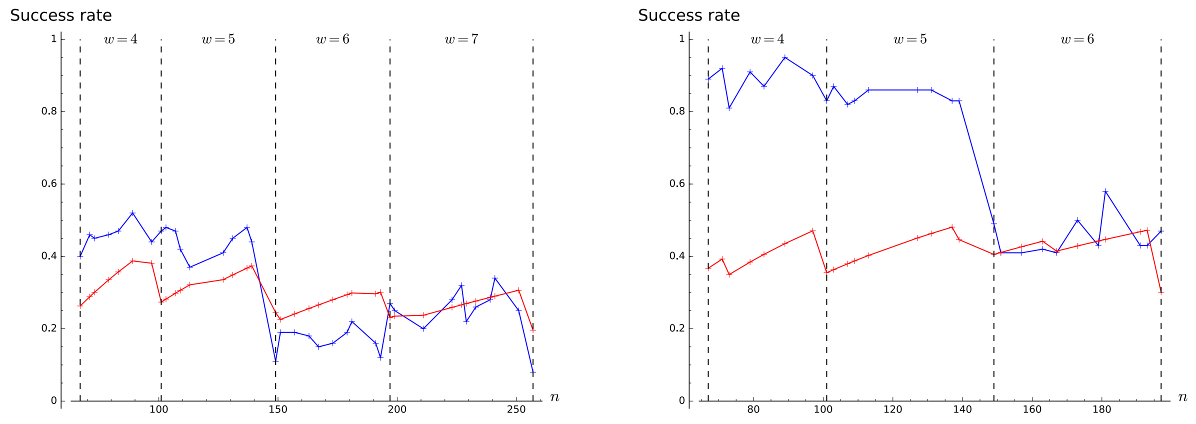
Quantum setting

Figure 2: 9th decile of the ratio between the measured number of collisions c and expected number of collisions c' according to Heuristic 3.3, over 100 experiments per dimension.

A.2 Running time and success probability

We now report on the practical efficiency of our attack and compare it to our heuristic prediction.

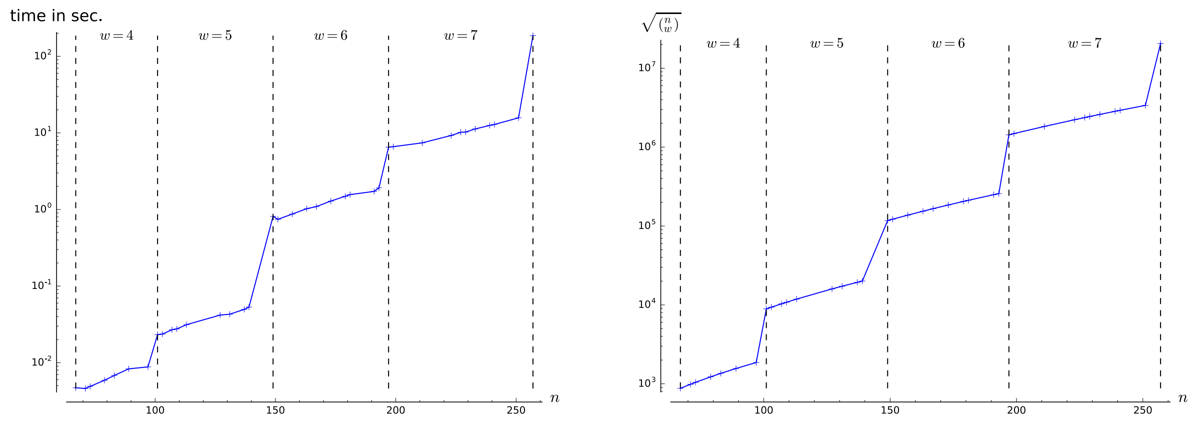
Remark. Note that in the quantum regime, the success probability of this MITM attack in practice is sometimes significantly larger than the theoretical prediction. This is most likely due to the fact that our analysis is done for one particular solution, while certain rotations of the same key may be found as well if its bits are properly balanced with respect to the split $\mathbb{F}_2^n = G_1 \oplus G_2$.



Classical setting

Quantum setting

Figure 3: Success rate of the attack over 100 trials (in blue), compared to the theoretical success rate $(1-2w/(n-B))^B$ (in red). The rather discontinuous shape of the red curve is due to the rounding of $w = \lfloor\sqrt{n}/2\rfloor$ and $B = \lfloor \log_2 \binom{n_1}{w_1} \rfloor$.



Experimental running time

Prediction

Figure 4: Average running time of the classical attack over 100 trials in comparison with the function $\sqrt{\binom{n}{w}}$, which is the dominant factor in our asymptotic complexity.