On the Hardness of the Mersenne Low Hamming Ratio Assumption

Marc Beunardeau, Aisling Connolly, Rémi Géraud, and David Naccache

Département d'informatique de l'ENS, École normale supérieure,
PSL Research University, Paris, France.
given_name.family_name@ens.fr

Abstract In a recent paper [AJPS17], Aggarwal, Joux, Prakash, and Santha (AJPS) describe an ingenious public-key cryptosystem mimicking NTRU over the integers. This algorithm relies on the properties of Mersenne primes instead of polynomial rings. The security of the AJPS cryptosystem relies on the conjectured hardness of the Mersenne Low Hamming Ratio Assumption, defined in [AJPS17].

This work shows that AJPS' security estimates are too optimistic and describes an algorithm allowing to recover the secret key from the public key much faster than foreseen in [AJPS17].

In particular, our algorithm is *experimentally practical* (within the reach of the computational capabilities of a large organization), at least for the parameter choice $\{n = 1279, h = 17\}$ conjectured in [AJPS17] as corresponding to a 2^{120} security level. The algorithm is fully parallelizable.

1 Introduction

A Mersenne prime is a prime of the form $2^n - 1$, where n itself is prime.

In a recent paper [AJPS17], Aggarwal, Joux, Prakash, and Santha (AJPS) describe an ingenious public-key cryptosystem mimicking NTRU over the integers. This algorithm relies on the properties of Mersenne numbers instead of polynomial rings. This scheme is defined by the following algorithms:

- $\mathsf{Setup}(1^\lambda) \to \mathsf{pp}$, which chooses the public parameters $\mathsf{pp} = (n, h)$ so that $p = 2^n - 1$ is prime and so as to achieve a λ-bit security level. In [AJPS17] the following lower bound is derived

  $$\binom{n-1}{h-1} > 2^\lambda$$

  which for instance is satisfied by $\lambda = 120, \mathsf{pp} = (n = 1279, h = 17)$.
- $\mathsf{KeyGen}(\mathsf{pp}) \to (\mathsf{sk}, \mathsf{pk})$, which picks F, G two n-bit strings chosen independently and uniformly at random from all n-bit strings of Hamming weight h, and returns $\mathsf{sk} \leftarrow G$ and $\mathsf{pk} \leftarrow H = F/G \bmod (2^n - 1)$.

- $\mathsf{Encrypt}(\mathsf{pp}, \mathsf{pk}, b \in \{0,1\}) \to c$, which picks A, B two n-bit strings chosen independently and uniformly at random from all n-bit strings of Hamming weight h, then computes

$$c \leftarrow (-1)^b (AH + B) \bmod (2^n - 1).$$

- $\mathsf{Decrypt}(\mathsf{pp}, \mathsf{sk}, c) \to \{\perp, 0, 1\}$, which computes $D = \|Gc \bmod (2^n - 1)\|$ and returns

$$\begin{cases} 0 & \text{if } D \leq 2h^2, \\ 1 & \text{if } D \geq n - 2h^2, \\ \perp & \text{otherwise} \end{cases}$$

We refer the reader to [AJPS17] for more details on this cryptosystem which does not require further overview because we directly attack the public key to infer the secret key.

In particular, security rests upon the conjectured intractability of the following problem:

Definition 1. *The* Mersenne Low Hamming Ratio Assumption *states that given an n-bit Mersenne prime $p = 2^n - 1$ and an integer h, the advantage of any probabilistic polynomial time adversary attempting to distinguish between $F/G \bmod p$ and R is at most $\frac{\text{poly}(n)}{2^\lambda}$, where R is a uniformly random n-bit strings, and (F, G) are independently chosen n-bit strings each having Hamming weight h.*

As we will see, we argue that (F, G) can be *experimentally* computed from H, at least for the parameter choice $\{n = 1279, h = 17\}$ conjectured in [AJPS17] as corresponding to a 2^{120} security level.

The full code (Python for partition sampling and Mathematica for lattice reduction) is available from the authors upon request.

2 Outline of the Analysis

The analysis uses the Lenstra–Lenstra–Lovász lattice basis reduction algorithm (LLL, [LLL82]). We do not recall here any internal details of LLL but just the way in which it can be used to solve a linear equation with k unknowns when the total size of the unknowns is properly bounded.

2.1 Using LLL to Spread Information

Let $x_1, \ldots, x_k \in \mathbb{N}^*$ be k unknowns. Let $p \in \mathbb{N}$ be a modulus and $a_0, \ldots, a_k \in \mathbb{N}$. Consider the equation:

$$a_0 = \sum_{i=1}^{k} a_i x_i \bmod p.$$

All the reader needs to know is that the LLL algorithm will find $x_1, \ldots, x_k$ if $\prod_{i=1}^{k} x_i < p$.

In particular, LLL can be adapted to provide any uneven split of sizes between the x_i as long as the sum of those sizes does not exceed the size of p. More details on the theoretical analysis of LLL in that setting and variants are given in [NS01, Sec. 3.2] and [Jou09, Chap. 13], in the context of generalised knapsack problems.

2.2 Partition and Try

The first observation that attracted our attention is that the size[1] of F (and G) has an unusually small expectation $\sigma(n,h)$:

$$\sigma(n,h) = n\left(1 + \frac{(1-\frac{h}{n})^{n+1}-1}{\frac{h}{n}(n+1)}\right)$$

The difference in size between $n = 1279$ and $\sigma(1279, 17)$ is not huge[2] and cannot be immediately exploited. However, the same phenomenon also occurs at the least significant bits and further shortens the expected nonzero parts of F and G by 70 bits.

Similarly, assume that in the key generation procedure, both F and G happen to have bits set to 1 only in their lower halves. When this (rare event) happens, we can directly apply LLL to H to recover F and G. We call this event T.

Is that event rare? Since F and G are chosen at random, T happens with probability at least 2^{-2h}. While T's probability is not cryptographically negligible, this pre-attack only allows to target one key out of 2^{2h}. For the first suggested parameter set ($\lambda = 120$), one public key out of 67 million can be attacked in this fashion and its F and G recovered, i.e., a total break. The question is hence, can this phenomenon be extended to any key? and if so, at what cost? In particular, can we sacrifice work to increase the size of the vulnerable key space? The answers to these questions turn out to be positive, as we will explain hereafter.

Random partitions. Instead of a fixed partition of $\{0, \ldots, n-1\}$, we can sample random partitions, for instance by sampling (without replacement) m positions, which are interpreted as boundaries between regions of zeros and regions that possibly contain a 1. The total number of regions, $m+1$, determines the dimension of the lattice being reduced.

For the sake of simplicity we consider *balanced* partitions:

Definition 2. *A partition of* $\{0, \ldots, n-1\}$ *into* $m/2$ *type 1 blocks and* $m/2+1$ *type 2 blocks is* balanced *if the total size of the type 1 blocks and the total size of the type 2 blocks differ by at most one.*[3]

[1] That is, the length of a number, once its leading zeros are discarded.

[2] $1279 - \sigma(1279, 17) \approx 75$ bits.

[3] Since n is odd, we must accept a ± 1 excess.

A randomly sampled partition is not necessarily a balanced partition: we use rejection sampling to ensure the balancing property.[4] The sought-after property of these partitions is the following:

Definition 3. *Let X be a binary string of length n. A partition of X into $m/2$ type 1 blocks and $m/2+1$ type 2 blocks is* correct for X *if the type 2 blocks are completely made of zeros.*

Figure 1 illustrates the partitions that we are interested in on a simple example. Also note that the definition above does not put any constraint on type 1 blocks, which may contain zeros or not; since they are not guaranteed to be zero we refer to them as "non-zero" blocks. Accordingly, blocks of type 2 in a correct partition is referred to as "zero" blocks.

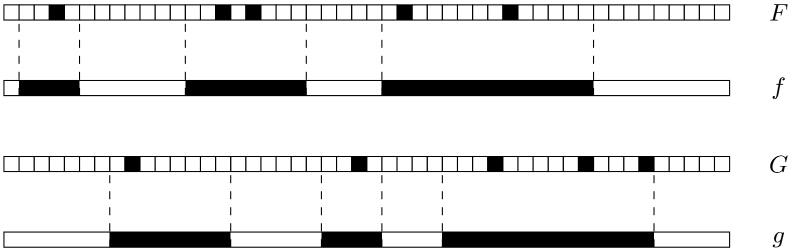

Figure 1. An illustration of the partitions that we are interested in: in these diagrams, a black square in F or G represents a 1, while white squares represent 0s. The partitions f and g are balanced and correct for F and G respectively, with "zero" blocks coloured white, and "non-zero" blocks coloured black. The vertical dashed lines show how F and G align with their respective partitions.

The observation at the beginning of this section is that using a balanced partition that is correct for F and another one that is correct for G, we can recover F and G from H.

Since F and G are unknown, we cannot construct a correct partition from them directly; but the probability that a random balanced partition is correct for F (resp. G) is lower bounded[5] by 2^{-h}. Assuming that F and G are independent, which they should be according to the key generation procedure, we found a correct partition for *both* F and G with a probability of 2^{-2h}.

Remark 1. We may also consider imbalanced partitions which allow an extra speed-up for a subtle reason: Given that the unknowns found by LLL have a low

[4] There is room for improvement here as well, since rejection sampling is a very inefficient approach. Nevertheless it will be sufficient for our discussion, and any approach to generating such partitions would work without impacting the analysis.

[5] We ignore the fact that we sample without replacement here, as $h \ll n$. Under this conservative approximation, all the bits are sampled uniformly and independently, and may fall with probably $1/2$ either in a type 1 or a type 2 block.

Hamming density, the odds that these numbers naturally begin by a sequence of zeros (and are hence shorter than expected) is high. The interesting point is that the total length of such natural gains sums up and allows to unbalance the partition in favor of type 1 blocks. Consider the analogy of a fishing boat that can carry up to 1000 kilograms of fish. The fishermen fishes with 3 nets having maximal capacities of 200, 300 and 500 kilograms each. Because waters are sparse in fish, the nets are expected to catch only 70% of their maximal capacity. Hence, we see that larger nets (285, 428, 714) can be used to optimize the boat's fishing capacity. However, unlike the boat, with LLL fish cannot be thrown back to the water and... excess weight sinks the boat (the attack fails). Hence if this speed-up strategy is used, we need to catch more than normal but not be too greedy. Note as well that if all variables end by at least ℓ trailing (LSB) zeros then these $m\ell$ zeros add-up to the gain as well (because there is no constant term in the equation a division of all variables by 2 has no effect on the solution's correctness). We did not exploit nor analyze these tricks in detail.

Trying partitions. The attack then consists in sampling a balanced partition, running LLL, and checking whether the values of F and G obtained from the reduction have the correct Hamming weight and yield H by division. Concretely, the matrix to be reduced is obtained as follows from the partitions f of F and g of G:

1. Compute the size of the each non-zero blocks in f and g, we call these sizes $\boldsymbol{u} = \{u_i\}$ and $\boldsymbol{v} = \{v_i\}$ respectively, with $i = 0, \dots, m/2 - 1$. Let $w = \max_i\{u_i, v_i\}$.
2. Construct the vector $\boldsymbol{s} = s_i$ as follows:
$$s_i = \begin{cases} 2^{w-v_i} & \text{if } i < m/2 \\ 2^{w-u_i} & \text{if } m/2 \leq i < m \end{cases}$$
3. Construct the vector $\boldsymbol{a} = \{a_j\}$ as follows: let f_i (resp. g_i) denote the starting position of the non-zero blocks in F (rep. G), and set
$$a_j = \begin{cases} H \times 2^{g_i} \bmod p & \text{if } j < m/2 \\ p - 2^{f_i} & \text{if } m/2 \leq j < m \end{cases}$$
4. Choose an integer K, and assemble the matrix $\boldsymbol{M}$ as follows:
$$\boldsymbol{M} = \begin{pmatrix} \operatorname{diag}(\boldsymbol{s}) & K\boldsymbol{a} \\ 0 & Kp \end{pmatrix}$$
where $\operatorname{diag}(\boldsymbol{x})$ is the diagonal matrix whose diagonal entries are given by $\boldsymbol{x}$. The coefficient K is a tuning parameter, which we set to 2^{1200}.
5. Finally, we use LLL on $\boldsymbol{M}$ (using the Mathematica command `LatticeReduce`) and recover the reduced matrix's row that complies with the Hamming density of F and G. This row is expected to give the values of the non-zero blocks

of F and G, and we can check its correctness by computing its Hamming weight, and checking that the ratio of the candidate values modulo p give H.

By the above analysis, a given partition is correct with probability 2^{-2h}, which for $\lambda = 120$ is only 2^{-34}; if we can run LLL reasonably fast, which is the case for $m = 16$, an efficient attack happens to be within the reach of a well-equipped organization. Experimental evidence indeed suggests the feasibility of the attack, see Section 3.

Remark 2. For larger security parameters λ, the ratio h/n deduced from the analysis in [AJPS17] asymptotically vanishes. It should be checked if this influences imbalanced partition finding to the attacker's relative advantage for larger values of λ. We did not explore this avenue left to the reader as a potential research question.

3 Putting it Together

To illustrate the attack's feasibility, we fix a random tape in a deterministically verifiable way and implement our algorithm (see Figure 2).

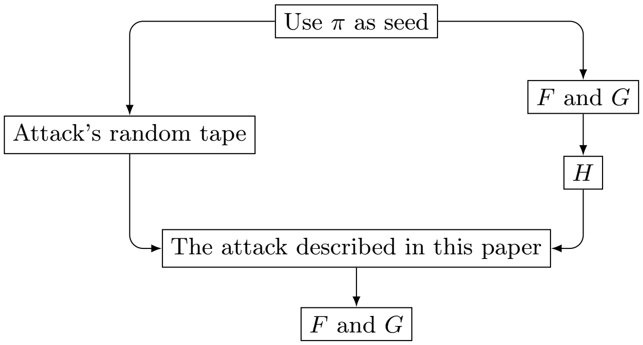

Figure 2. The feasibility demonstration consists in deriving the attack's random tape from a verifiable source in a deterministic way, as well as the keys.

We generated a nothing-up-our-sleeves key with the procedure of Figure 3. The sample(S, h) procedure selects h indices without replacement in the range S. It is implemented[6] by returning the h first entries of a deterministic Fisher–Yates shuffle of S. The randomness in sample(S, h) is simulated by iterating the

[6] Other implementations are of course possible and do not affect the analysis. For other classical sampling without replacement algorithms, the reader may consult [SW12].

```
$n, h \leftarrow \mathsf{pp}$
$I_1 = \{i_1, \ldots, i_h\} \leftarrow \text{sample}(\{0, \ldots, n-1\}, h)$
$I_2 = \{i_1, \ldots, i_h\} \leftarrow \text{sample}(\{0, \ldots, n-1\}, h)$
$F \leftarrow \sum_{i \in I_1}^{h} 2^i$
$G \leftarrow \sum_{i \in I_2}^{h} 2^i$
return $(\mathsf{sk} = G, \mathsf{pk} = F \cdot G^{-1} \bmod p)$
```

Figure 3. The $\mathsf{KeyGen}(\mathsf{pp})$ procedure.

SHA256 function, starting with the seed given by the ASCII representation of the 100 first decimals of π:

```
31415926535897932384626433832795028841971693993751
05820974944592307816406286208998628034825342117068
```

In a real attack we would simply use a fast non-cryptographic random number generator, but the above choice serves the purpose of reproducibility.

This gives the following (in hexadecimal notation, the zero MSBs have not been written):

$$I_1 = \{\mathtt{33}, \mathtt{47}, \mathtt{8e}, \mathtt{95}, \mathtt{a1}, \mathtt{134}, \mathtt{19f}, \mathtt{1ab}, \mathtt{1ac}, \mathtt{1ce}, \mathtt{25d}, \mathtt{301}, \mathtt{30a}, \mathtt{3ee}, \mathtt{444}, \mathtt{46b}, \mathtt{471}\}$$

$$I_2 = \{\mathtt{89}, \mathtt{b5}, \mathtt{de}, \mathtt{116}, \mathtt{141}, \mathtt{1dd}, \mathtt{1de}, \mathtt{2ae}, \mathtt{322}, \mathtt{37a}, \mathtt{388}, \mathtt{38a}, \mathtt{3f9}, \mathtt{48c}, \mathtt{48d}, \mathtt{4e9}, \mathtt{4f2}\}$$

```
F = 2080000000010000000000000000000004000000000000000000000000000000
    0000000000000000000000000040200000000000000000000000000000000000
    0000020000000000000000000000000000000000040000000180080000000000
    0000000000000001000000000000000000000000000000000000200204000000
    00000000000800000800000000000
```

```
G = 4020000000000000000000000300000000000000000000000000000000000020
    0000000000000000000000000050004000000000000000000000400000000000
    0000000000000000040000000000000000000000000000000000000000000000
    0000060000000000000000000000000000000000000020000000000400000000
    0000040000000002000000000020000000000000000000000000000000000
```

```
H = 1610fecf11dbd70f5d09da1244a85c3aa7aed7de75a6d1fe4e988b5f66d66e1b
    c27d46afd96800ff8b2b67316dff1046b88d205e620ba78a813c15f47ab8a7d2
    a8f7eb12fe0fcff882307d92d4c0f9296a7cf4390ce3140e11e4b7c802fa67d3
    a8517d30b00980380bdf8992ed6a2d3f74e25f14bae21786672bddae4f2bf897
    f38741cdc10b319f8272d42f738cd296d4907331518c3439621aefad5c3d1a7c
```

3.1 Recovering F and G from H

Finding a Winning Partition. At this step, we generate random balanced partitions and try LLL on the resulting decomposition. Doing so we quickly find the following partitions

$$f = \{\texttt{2a}, \texttt{bf}, \texttt{134}, \texttt{1ec}, \texttt{233}, \texttt{253}, \texttt{25a}, \texttt{270}, \texttt{2ee}, \texttt{32d}, \texttt{3e4}, \texttt{41e}, \texttt{42b}, \texttt{4a7}, \texttt{4f6}, \texttt{4fd}\}$$
$$g = \{\texttt{7c}, \texttt{142}, \texttt{1d0}, \texttt{22a}, \texttt{289}, \texttt{2c8}, \texttt{2de}, \texttt{2e7}, \texttt{2eb}, \texttt{33c}, \texttt{372}, \texttt{3a0}, \texttt{3da}, \texttt{3ff}, \texttt{48a}, \texttt{4fd}\}$$

respectively for F and G, which upon lattice reduction yield candidates of the correct Hamming weight. Their ratio indeed gives H; however one may debate our claim that this partition was found at random and argue that we constructed it from our prior knowledge of F and G.

To counter this argument and insist that finding partitions is reasonably easy, we derived them *deterministically from the same seed as the key.* To achieve this, we proceed as follows: we draw two independent sets of $m/2 - 1$ indices in the range $[0, n/2]$, which gives the sizes of the zero blocks and the non-zero blocks. This guarantees that the partitions are balanced. The randomness used for this sampling is obtained by iterating SHA256 as for key generation.

As in the example above, we constructs partitions for $m = 16$ — this choice is not dictated by probability (as the likelihood to find a correct partition is in theory independent of m), but rather by a trade-off between the cost of LLL and the number of partitions explored. It is possible for instance to start with $m = 2$ partitions, then $m = 3$, and so forth, but we settled for a random search which is easier to implement.

We found the following partition for F at run #1,152,006 (in 116 s):

$$f = \{\texttt{27}, \texttt{b2}, \texttt{10e}, \texttt{13c}, \texttt{198}, \texttt{1cf}, \texttt{24b}, \texttt{27b}, \texttt{2ac}, \texttt{30f}, \texttt{3e1}, \texttt{456}, \texttt{45a}, \texttt{4ba}, \texttt{4d6}, \texttt{4fd}\}$$

Recovering F alone took about two minutes.[7] Given that we have a totally deterministic random tape, we regard our experiment as legitimately reflecting reality. Because F and G are independent, this brings the total effort to about the square of this number, i.e. about 2^{34} attempts to get both partitions with certainty. Each of these attempts must also involve one LLL, which is the main cost factor.

Using the same sequence, #64,249 gave a partition for G too (in 7.6 s):

$$g = \{\texttt{7b}, \texttt{11c}, \texttt{13b}, \texttt{181}, \texttt{1cc}, \texttt{1e1}, \texttt{284}, \texttt{2e6}, \texttt{318}, \texttt{329}, \texttt{36f}, \texttt{3e5}, \texttt{3f1}, \texttt{404}, \texttt{476}, \texttt{4fd}\}$$

Finally, note that the task is fully parallelizable and would benefit from running on several independent computers, a remark that we will later use in our final workfactor estimates.

Computing the Secret Key Running our program as explained in Section 2, we recover F, G, and confirm that $H = F/G \bmod p$.

[7] Experiments with random partitions show that this number is quite variable and follows a Poisson distribution, with a correct partition being typically found earlier, with an average of 2^{17} tries.

3.2 Predicting the Total Execution Time

Putting all the above figures together and assuming no further algorithmic improvements, the total expected effort is:

$$\frac{(\text{LLL\_Time} + 2 \times \text{Partition\_Time}) \times \text{Average\_Partition\_Tries}^2}{\text{Number\_of\_Processors}}$$

Where, in our basic scenario Average_Partition_Tries $= 2^h$.

We performed LLL on Mathematica using the `LatticeReduce` function, which took less than a second in the worst case on a simple laptop. We safely assume that this figure can be divided by 10 using a dedicated and optimized code. We also assume that a credible attacker can, for example, very easily afford buying or renting 150 TILE-Gx72 multicore processors.

$$\frac{\frac{1}{10} \times 1{,}152{,}006 \times 64{,}249}{150 \times 72} \times \frac{1}{60 \times 60 \times 24} \approx 7 \text{ days } 22 \text{ hours}$$

Hence, according to the evidence exhibited in this paper, breaking a 1279 bit key takes a week using 150 currently available multicore processors (e.g. TILE-Gx72).

4 Conclusion

While we did not formally evaluate efficiency nor asymptotic complexities, our quick and dirty experiments clearly suffice to show that key recovery is fast and within reach. An obvious countermeasure consists in increasing parameter sizes. Hence a precise re-evaluation of parameter sizes and safety margins of the Mersenne Low Hamming Ratio Assumption seems in order.

More systemic protections may consist in modifying the definition of H (and possibly the underlying cryptosystem) which is clearly a very interesting open problem.

Nonetheless the beautiful idea of Aggarwal, Joux, Prakash, and Santha exploiting the fact that arithmetics modulo Mersenne numbers is (somewhat) Hamming-weight preserving, is very elegant and seems very rich in possibilities and potential cryptographic applications.

References

AJPS17. Divesh Aggarwal, Antoine Joux, Anupam Prakash, and Miklos Santha. A new public-key cryptosystem via Mersenne numbers. Cryptology ePrint Archive, Report 2017/481, 2017. http://eprint.iacr.org/2017/481.

Jou09. Antoine Joux. *Algorithmic cryptanalysis*. CRC Press, 2009.

LLL82. Arjen Klaas Lenstra, Hendrik Willem Lenstra, and László Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261(4):515–534, 1982.

NS01. Phong Q. Nguyen and Jacques Stern. The two faces of lattices in cryptology. In Joseph H. Silverman, editor, *Cryptography and Lattices, International Conference, CaLC 2001, Providence, RI, USA, March 29-30, 2001, Revised Papers*, volume 2146 of *Lecture Notes in Computer Science*, pages 146–180. Springer, 2001.

SW12. Dennis Stanton and Dennis White. *Constructive combinatorics*. Springer Science & Business Media, 2012.