Quantum Cryptanalysis of NTRU

Scott Fluhrer

Cisco Systems

July 4, 2015

bb

1 Abstract

This paper explores some attacks that someone with a Quantum Computer may be able to perform against NTRUEncrypt, and in particular NTRUEncrypt as implemented by the publicly available library from Security Innovation. We show four attacks that an attacker with a Quantum Computer might be able to perform against encryption performed by this library. Two of these attacks recover the private key from the public key with less effort than expected; in one case taking advantage of how the published library is implemented, and the other, an academic attack that works against four of the parameter sets defined for NTRUEncrypt. In addition, we also show two attacks that are able to recover plaintext from the ciphertext and public key with less than expected effort. This has potential implications on the use of NTRU within TOR, as suggested by Whyte and Schanck[3]

2 Introduction

NTRUEncrypt[2] is a public key encryption system designed by Jeffrey Hoffstein, Jill Pipher and Joseph Silverman. It has several attractive features, one of which is that it is immune to attacks by Shor's algorithm (as it does not rely on a factorization or discrete log hard problem). Hence, it looks to be a logical component as a part of a Quantum-Resistant cryptosystem.

NTRU does appear to be immune to Shor's algorithm (which allows the attacker to quickly factor large integers and compute discrete logarithms). However, a Quantum Computer also allows an attacker to run Grover's algorithm[1], which is able to find a n bit solution to a problem in $2^{n/2}$ time. The question we would like to look at is 'how can Grover's algorithm be used to advantage in attacking NTRU?'

There has been previous analysis of the Quantum Resistance of NTRU, such as by Wang, Ma and Ma[5], however those works studied previously defined parameter sets. This work is focusing on the parameter sets distributed with the current NTRU library.

3 NTRU Basics

NTRU works in the ring of polynomials $Z[x]/x^N - 1$, where N is a prime. Computations in this ring are actually done modulo a prime power; NTRUEncrypt actually evaluates additions and multiplications modulo a prime power (q) and a small polynomial (p) during the course of its operation. However, all the operations that we'll examine are done modulo q (where $q = 2048$ is a typically choice), hence for the purposes of this paper, we can consider the ring to be $Z[x]/(x^N - 1, q)$. In addition to the base NTRU operation, NTRUEncrypt uses a padding mechanism called NAEP[4] to protect the underlying NTRU primitive from the attacker being able to deduce information from decryption failures.

There are a number of parameter sets defined for NTRU; each parameter set includes the value of N that are used during the NTRU operations, the values of p and q, as well as the expected security level for this parameter set (that is, the value k for which we expect any attack against this parameter set to take at least $O(2^k)$ operations.

NTRUEncrypt is available as a free-for-noncommercial use library from Security Innovation; we will be analyzing the parameter sets and the key generation and padding methods as implemented by that library.

When we select a private key for NTRUEncrypt, we select two polynomials F and G with specific sets of coefficients; we also need to make sure that F is invertible. Once we have that, we can compute the public polynomial $H = F^{-1}G$. The public key decryption process uses the secret polynomial F to decrypt.

When we encrypt a message m with an NTRUEncrypt public key, the library performs the following steps:

- It first examines the public key to get the security level k of the parameter set that the public key belongs to. Currently defined parameter sets have $k \in \{112, 128, 192, 256\}$
- It then selects a random k-bit value b
- It encodes the value b, the message m and a portion of the public key into a string, and hash that string to form a value ρ. It uses SHA-1 as the hash function if $k \leq 160$ and SHA-256 if $k > 160$
- It uses ρ to seed a random number generator, and uses the output of that random number generator to select a polynomial R

- It encodes the message m and the random value b as a polynomial M which consists solely of coefficients in the set $\{0, 1, -1\}$; we run a check on the value M to make sure that it has sufficient coefficient diversity (that is, this check makes sure that each of the possible values occurs sufficiently many times); if not, we go back and select a different value for b.
- It extracts the polynomial H from the public key, and generate the ciphertext $HR+M$ (where the computation is done in the ring, calculating everything modulo q).

All but the last step is actually the NAEP padding procedure, used for generate the polynomials R and M for the actual NTRU operation.

During decryption, the decryptor recovers the values m and b; it then uses the above logic to recompute R (and checks to make sure that was the R used to generate the ciphertext; this prevents invalidly generated ciphertexts from becoming an issue). What this means is that the encryptor must use this formula to generate R from b, m and the public key.

4 Key Recovery Attack 1

The NTRU public key H is the polynomial $F^{-1}G$ (computed over $\mod q$), where F and G are sparse polynomials; the polynomial F is the private key. One way to try to recover the private key is to search for an F such the product polynomial FH is sparse (which, in this case, means consists of $dG+1$ coefficients of the value p, dG coefficients of the value $-p$, and the rest 0, where dG is a constant from the parameter set).

Now, the defined NTRU parameter sets work in one of two ways; in the straight-forward method, the key generation process selects a random polynomial whose coefficients consists of dF 1's, dF -1's, and the rest 0 (where dF is a parameter from the parameter set). The other method, known as the product form, has the key generation process select three polynomials F_1, F_2, F_3, and effectively sets $F = F_1F_2 + F_3$. Each of these polynomials F_1, F_2, F_3 is sparser than the target F polynomial; and computing $F_1(F_2H) + F_3H$ is faster than computing FH directly.

To use Grover's algorithm, we could search for the polynomial F for which FH is sparse. However, doing this in this straight-forward manner doesn't work (as for all defined parameter sets, there are more than 2^{2k} possible values of F).

If we are working with a parameter set that uses the product form, one way to rewrite the equation $FH = G$ is in the form $F_1F_2H = G - F_3H$, where we know that each coefficient of G is either $0, p$ or $-p$.

The next obvious improvement is to check for this equality $\mathrm{mod} p$; as all coefficients of G are $0 \bmod p$, this simplifies the equation to $F_1F_2 \equiv -F_3H \pmod{p}$.

Now, there is an obvious objection to this; both F_1F_2 and F_3H are computed modulo q, which is relatively prime to p; what happens if an addition within $G - F_3H$ which "wraps"; that is, $-F_3H$ has one sign, and $G - F_3H$ has the other?

Well, the obvious answer is "the search for the public key fails in that case". It is worth noting that the probability of such a wrap happening is reasonable; the parameter sets in question have $p = 3$, $q = 2048$ and G having between 267 and 495 nonzero entries; assuming that the coefficents of F_3H are random, this gives us a success probability between 0.67 and 0.48. So, the obvious way to address this failure mode is, if the initial search fails, we add a random constant to both sides of the equation, and rerun the search; this has the effect of doubling the expected search time.

So, the obvious search algorithm is to store the coefficients of $-F_3H \pmod p$ (for all possible F_3 values), and then perform the Quantum Computer search over all possible F_1, F_2 values for where the coefficients of $F_1F_2H \pmod 3$ is a match for one of the precomputed values. With reasonable probability the correct private key will be the only one that gives a plausible setting; in fact, we don't need to search over all the coordinates, instead it is sufficient to search over enough to make any false hit unlikely. If we find a match, this will allow us to rederive F in (strictly speaking) fewer than 2^k operations.

In addition, we can reduce this work effort by taking advantage of the rotational symmetry of the multiply operation within $F[x]/x^N - 1$; if we consider the product AB, and then consider the product $A'B$ (where A' has the same coefficients as A, only rotated by m positions, then the product $A'B$ will have the same coefficients as AB, only rotated by m positions. This can easily be seen as the operation of rotating by m positions is the operation of multiplying by the polynomial x^m, and $(x^mA)B = x^m(AB)$. This allows us to reduce the number of F_3 polynomials we consider by a factor of N (because if a specific value of F_3 gives a solution with small coefficients, so will any rotation of F_3 and the corresponding F_1F_2.

Ttere is also a second symmetry that we can take advantage of; if we rotate the elements of F_1 left by m positions, and the elements of F_2 right by m positions, the resulting product $F_1'F_2'$ will be unchanged. That is, $(x^mF_1)(x^{-m})F_2 = F_1F_2$. This is a second symmetry that reduces the number of the products F_1F_2 we need to consider by a factor of N.

The result of these two improvements reduce the number of $-F_3H \bmod p$ values we need to precompute by a factor of N, and the number of $F_1F_2 \bmod p$ polynomials that we need the Quantum Computer to search over by a factor of N.

When we consider the parameter set EES593EP1 (which has a design strength of 192), we find it has $N = 593$, and $dF_1 = 10$, $dF_2 = 10$ and $dF_3 = 8$. This implies that the total number of F_3 polynomials is $\binom{593}{16}\binom{16}{8}$ (because each F_3 polynomial coefficients consists of 8 ps, 8 $-p$'s, and 577 0s), and the number of F_1F_2 polynomials as $\binom{593}{20}\binom{20}{10}\binom{593}{20}\binom{20}{10}$. When we take into account the factor of N decrease (because of the rotational symmetry) , this gives us $|F_3|/N \approx 2^{107.285}$. When we consider the number of F_1F_2 polynomials, we take account of the factor N decrease (because of the second rotational symmetric), this gives us $|F_1||F_2|/N \approx 2^{271.165}$. A Quantum Computer is able to search over a set of this size in approximately 2^{136} time, and this second step would dominate. When we account for the failure probability (and the possibility we're need to rerun this procedure), this gives us an overall time of $O(2^{137})$, which is considerably smaller than the original design goal of 192 bits.

When we consider all four product form parameter sets, we find that the 112 bit product form set (EES401EP2) can be attacked with an expected $O(2^{104})$ work, the 128 bit set (EES439EP1) can be attacked with an expected $O(2^{112})$ work, and the 256 bit set (EES743EP1) can be attacked with an expected $O(2^{197})$ work. In this last case, it's actually the derivation of the possible values of $-F_3H$ which is the dominating factor (because $dF_3 = 15$ for this parameter set, which is comparitively large.

Now, this approach has managed to recover the private key using fewer NTRU multiplications than expected. On the other hand, this approach is quite impractical (even beyond the number of operations involved); it assumes that we can practically check for the existence of an entry in a table with more than 2^{65} entries in constant time.

An obvious alternative approach would be to search for equalities between F_1F_2H and $G - F_3H$ modulo 2; this would allow us to ignore the sign differences in the F_1, F_2, F_3 coefficients (and avoids the possibility of the attack failing because of a wrap, as all have parameter sets in question has q being a power of 2). However, this approach turns out not to work, because G happens to be relatively dense evaluated modulo 2 in the parameter sets in question, and so there's no obvious way to determine when we've detected the correct F_1, F_2, F_3 set.

5 Key Recovery Attack 2

When the NTRU library generates a key, it goes through this procedure:

- It selects a random bitstring that is the design strength k of the parameter set, plus 64 bits (for example, for a 128 bit parameter set, it obtains 192 bits from the internal random number generator)
- It hashes this random number (with SHA-1 if the parameter strength $k \leq 160$ and SHA-256 if $k > 160$) giving us a hash value h.
- It uses that hash value h (and nothing else) to seed a random number generator, and the output of that random number generator selects a polynomial F
- It uses a similar process to select a polynomial G

This process is similar to the process used to select the random polynomial R during encryption (largely because it reuses the same code).

This key generation procedure immediately gives us a potential key recovery attack; given a public key H, we use Grover's algorithm to guess the hash value h; our verification step would seed the random number generator, selects a polynomial F', and then compute the product $F'H \pmod{q}$; if that consists solely of elements in the set $\{-p, 0, p\}$, then we accept. This works because if $F = F'$, then we have $F'H = G$ (which has the special form listed).

6 Plaintext Recovery Attack 1

A similar approach to recover the plaintext given a ciphertext C and a public key H is to run Grover's algorithm, with the guess this time being ρ the output of the hashed string. That is, they would use the function that takes a value for ρ, run it through the random number generator to generate a guess of R, and check if the polynomial $C - HR$ consists solely of the coefficients $\{0, 1, -1\}$; the correct guess of the hash will do that (as $C - HR$ is the encoded message M, which is a polynomial with those coefficients); an incorrect guess is quite unlikely to have coefficients limited to that range.

For security levels 112, 128, we use a 160 bit hash (and 2^{160} possible values for ρ), hence this approach will take an expected $O(2^{80})$ operations. For security levels 192, 256, we use a 256 bit hash, hence this approach will take an expected $O(2^{128})$ operations; in both cases, the work required is significantly smaller than the target security levels.

This is quite similar to one of the key recovery attacks we have presented above (because both attacks work against the common logic used to generate both F and R). However, there is a distinction in that the key generation procedure could be modified to use a stronger method to select F (and G) without any interoperability issue. In contrast, we cannot modify how we generate R without modifying how decryption is done (as the decryptor will expect to be able to regenerate R as part of the post-decryption validation process. Covering this plaintext recovery attack would require modifying the NTRU padding method.

7 Plaintext Recovery Attack 2

Another way an attacker can attempt to recover the plaintext, if the plaintext is low entropy, is to notice that the ciphertext is a deterministic function of the plaintext, the public key, and the k-bit random value b. If the entropy of the plaintext is low (has only 2^n possible values, for $n \lll k$), then what an attacker with a Quantum Computer could do is model the system as one with 2^{n+k} inputs (which consists of the 2^n possible inputs for the plaintext, paired with the 2^k possible values for b), and then apply Grover's algorithm to find a solution (that is, m and b) that generates the known ciphertext in $O(2^{(n+k)/2})$ time, which is less than the security level 2^k (and if n is sufficiently small, this value may be even smaller than the previous attack).

Now, in practice, this attack would not generally appear to be a significant threat; in most cases, NTRUEncrypt will be used to pass symmetric keying data, and symmetric keying data has sufficiently high entropy to make this attack infeasible. However, in those cases where NTRUEncrypt is used to transmit low-entropy plaintexts or if the attacker might have a plausible guess for the plaintext (and it is important to make such verification infeasible), this attack is of concern.

8 Conclusions and Recommendations

We have presented four attacks where an adversary with a Quantum Computer is able to attempt against NTRUEncrpyt (as implemented by the current NTRU library). One of the key recovery attacks actually attacks the key generation process that the NTRU library uses; it would be easy to modify the library to foil this approach (for example, both to use stronger hash functions when generating the hashed value h, and by extending the entropy extracted from the random number generator from $k + 64$ bits to at least $2k$ bits to prevent someone using Grover's algorithm to guess the preimage value). Because of the ease of this modification, and because the modified library would continue to interoperate with existing NTRU implementations, we recommend that such a change be made (even if it is not clear if this attack is actually practical).

We have also presented another key recovery attack that uses fewer operations than expected to recover the private key (assuming one of the four standardized parameter sets); however this attack is thoroughly impractical. We don't recommend any change to cover this attack.

Neither of the two plaintext recovery attacks we have presented actually attack the NTRU primitive itself; instead, they attack the NAEP padding method, and take advantage of the fact that the internal primitives selected for the parameter sets are scaled to withstand attacks by a classical computer, and are not sufficient if the attacker has a Quantum Computer.

These attacks may have some practical impact. For example, in 'A quantum-safe circuit-extension handshake for Tor[3]', they suggest using the NTRUEncrypt parameter set EES439EP1 (a 128 bit parameter set) to protect Tor traffic; do these results mean that someone with a Quantum Computer could reconstruct the original sender of a message with $O(2^{80})$ work?

There are three obvious ways to address the issues here. The first strategy would be to use a stronger parameter set; if your target is 128 bits security, use an NTRU parameter set targeted towards 256 bit security; such a parameter set would provide at least 128 bits security against all known attacks, even if the attacker has a Quantum Computer. The costs here are that the ciphertext and public key sizes increases, as well as a small increase in the encryption and decryption time.

The second strategy would be to define alternative parameter sets that are designed to be Quantum-Resistant. These alternative parameter sets might use the same polynomials as the current sets; however we would modify the primitives used using the NAEP padding method; they would use wider hash functions, and larger b values. The costs here would be that the new parameter sets would be incompatible with libraries that only understood the old sets, plus a minor decrease in the amount of plaintext that NTRU could encrypt in a single message (because b is now larger).

The third way of addressing this is to question whether this is actually a threat after all. $O(2^{64})$ work (as the worse case plaintext recovery attack) may sound like a conceivable amount of work; however that notation conceals a constant of proportionality, and that constant might be of significant size. In both of these attacks, the amount of work involves is actually $O(2^{64})$ encryption operations; in addition, each operation is done on a Quantum Computer (using entangled states, and using some Quantum Error Correction logic). It would appear plausible that such an operation may be considerably more expensive than the corresponding operation on a classical computer. However, we don't have a working Quantum Computer, and so we can't be certain how much more expensive these operations would be. Because of this uncertainty, while this line of argument may sound promising, our opinion is that we shouldn't rely only on that; we recommend one of the previous two strategies (especially given their relatively low cost).

References

[1] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing - STOC '96*. ACM Press, 1996.

[2] Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. NTRU: A ring-based public key cryptosystem. In *Algorithmic Number Theory*, pages 267–288. Springer Science Business Media, 1998.

[3] Zhenfei Zhang John Schanck, William Whyte. A quantum-safe circuit-extension handshake for Tor. Eprint, 2015.

[4] Ari Singer William Whyte Nick Howgrave-Graham, Joseph H. Silverman. NAEP: Provable security in the presence of decryption failures. Eprint, 2003.

[5] Hong Wang, Zhi Ma, and ChuanGui Ma. An efficient quantum meet-in-the-middle attack against NTRU-2005. *Chin. Sci. Bull.*, 58(28-29):3514–3518, oct 2013.