Choosing Parameter Sets for NTRUEncrypt with NAEP and SVES-3

Nick Howgrave-Graham, Joseph H. Silverman, William Whyte

NTRU Cryptosystems,
5 Burlington Woods, MA 01803.

Abstract. We present, for the first time, an algorithm to choose parameter sets for NTRUEncrypt that give a desired level of security.

Note: This is an expanded version of a paper presented at CT-RSA 2005.

1 Introduction

Different descriptions of NTRUEncrypt, and different proposed parameter sets, have been in circulation since 1996 [9–11, 5]. However, the method for choosing parameter sets has always been something of a black art. No single paper has ever described a machine which takes as input a desired security level k and outputs a parameter set that gives k bits of security.

It is the aim of this paper to provide such a machine. This paper presents a fixed algorithm to generate all required parameters for NTRUEncrypt with the SVES-3 encryption scheme, starting from a single parameter, k. Additionally, we present a more flexible framework generalizing the fixed algorithm in order to allow for architecture and efficiency tradeoffs, while still maintaining security against all known attacks. The fixed algorithm presented earlier always produces parameters consistent with this framework. We arrive at the parameter bounds specified in this framework by reviewing the effectiveness of each known attack.

Finally, to demonstrate the flexibility of the general framework, we consider a different set of constraints and generate a parameter set subject to those constraints.

2 A Specific Algorithm

This section gives a specific instantiation of the parameter generation algorithm for NTRUEncrypt-SVES-3 with binary underlying polynomials and $p = 2$. The input to this algorithm is the security parameter k. We denote this algorithm $\mathcal{P}^1_{\mathsf{ntru}}$, and denote by $\mathcal{P}^1_{\mathsf{ntru}}(k)$ the parameter set produced by this algorithm with input k. Table 2 gives $\mathcal{P}^1_{\mathsf{ntru}}(k)$ for various common values of k. We present a more general parameter generation framework later.

1. Set N to be the first prime greater than $3k + 8$.
2. Set d to be the smallest integer such that
$$\frac{1}{\sqrt{N}}\binom{N/2}{d/2} > 2^k \ .$$
Set $d_F = d_r = d$. Set $d_g = \lfloor N/2 \rfloor$.
3. Set d_{m0} to be the largest integer such that
$$2^{N-1}\sum_{i=0}^{d_{m0}}\binom{N}{i} < 2^{-40} \ .$$
If $\frac{1}{\sqrt{N}}\binom{N/2}{d_{m0}} < 2^k$, increase N to the next largest prime and return to step 2.
4. Set q to be the first prime greater than $4d + 1$.
5. Verify that the order of $q \pmod N$ is $N - 1$ or $(N - 1)/2$. If it is not, increase q to the next prime value until a q with a sufficiently high order is found.

6. Calculate $c = \sqrt{4\pi e \sqrt{d(N-d)/N} \sqrt{d_{m0}(N-d_{m0})/N}/q}$. From Table 1, read the values A and B. If

$$AN - B - \max_r \left(\log_2 \left(1 - \left(1 - \prod_{i=0}^{d-1} \left(1 - \frac{r}{N-i} \right) \right)^N \right) + Ar/2 \right) < k \ ,$$

increase N to the next largest prime and return to step 2. Otherwise, output $\{N,\ q,\ p=2,\ d_F,\ d_r,\ d_g,\ d_{m0}\}$ and stop.

The rest of this paper derives and justifies this algorithm.

3 Bit Strength

We quantify security in terms of bit strength k, evaluating how much effort an attacker has to put in to break a scheme. All the attacks we consider here have variable running times, so we describe the strength of a parameter set using the notion of *cost*. For an algorithm $\mathcal{A}$ with running time t and probability of success ε, the cost is defined as

$$C_{\mathcal{A}} = t/\varepsilon \ .$$

This definition of cost is not the only one that could be used. For example, consider *indistinguishability against adaptive chosen-ciphertext attack*. In this attack, an attacker with access to encryption and decryption oracles chooses two messages M_0 and M_1, and is given the encryption of one of them. The attacker's output is a single bit $i \in \{0,1\}$. She wins if M_i was in fact the encrypted message. Here, the relevant measure of the attacker's power is the advantage over a random guess, defined as

$$\mathrm{adv}(\mathcal{A}(\mathsf{ind})) = 2.(\Pr[\mathsf{Succ}[\mathcal{A}]] - 1/2) \ .$$

We will use either measure as appropriate.

Our notion of cost is derived from [19] and related work. An alternate notion of cost, which is the definition above multiplied by the amount of memory used, is proposed in [28]. The use of this measure would allow significantly more efficient parameter sets, as the meet-in-the-middle attack described in Section 5.1 is essentially a time-memory tradeoff that keeps the product of time and memory constant. However, current practice is to use the measure of cost above.

We also acknowledge that the notion of comparing public-key security levels with symmetric security levels, or of reducing security to a single headline measure, is inherently problematic — see an attempt to do so in [24], and useful comments on this in [17]. In particular, extrapolation of breaking times is an inexact science, the behavior of breaking algorithms at high security levels is by definition untested, and one can never disprove the existence of an algorithm that attacks NTRUEncrypt (or any other system) more efficiently than the best currently known method. However, estimates of security have to start somewhere, and we consider this paper to provide a useful starting point for NTRUEncrypt.

4 The NTRUEncrypt one-way function

An implementation of the NTRUEncrypt encryption primitive is specified by the following parameters:

N	*Degree Parameter.* A positive integer. The associated NTRU lattice has dimension $2N$.
q	*Large Modulus.* A positive integer. The associated NTRU lattice is a convolution modular lattice of modulus q.
p	*Small Modulus.* An integer or a polynomial.
$\mathcal{D}_f, \mathcal{D}_g$	*Private Key Spaces.* Sets of polynomials from which the private keys are selected.
$\mathcal{D}_m$	*Plaintext Space.* Set of polynomials that represent encryptable messages. It is the responsibility of the encryption scheme to provide a method for encoding the message that one wishes to encrypt into a polynomial in this space.
$\mathcal{D}_r$	*Blinding Value Space.* Set of polynomials from which the temporary blinding value used during encryption is selected.

`center` *Centering Method.* A means of performing mod q reduction on decryption.

Definition 1. *The* Ring of Convolution Polynomials *is*

$$\mathcal{R} = \frac{\mathbb{Z}[X]}{(X^N - 1)}.$$

Multiplication of polynomials in this ring corresponds to the convolution product of their associated vectors. We also use the notation $\mathcal{R}_q = \frac{(\mathbb{Z}/q\mathbb{Z})[X]}{(X^N-1)}$.

Definition 2. *A polynomial* $a(X) = a_0 + a_1 X + \cdots + a_{N-1}X^{N-1}$ *is identified with its vector of coefficients* $\mathsf{a} = [a_0, a_1, \ldots, a_{N-1}]$. *The* centered norm $\|\mathsf{a}\|$ *of a polynomial or vector is defined by*

$$\|\mathsf{a}\|^2 = \sum_{i=0}^{N-1} a_i^2 - \frac{1}{N}\left(\sum_{i=0}^{N-1} a_i\right)^2 . \tag{1}$$

Definition 3. *The* width $\mathsf{Width}(\mathsf{a})$ *of a polynomial or vector is defined by*

$$\mathsf{Width}(\mathsf{a}) = \mathsf{Max}(a_0, \ldots, a_{N-1}) - \mathsf{Min}(a_0, \ldots, a_{N-1}) .$$

4.1 Key Generation

NTRUEncrypt **key generation** consists of the following operations:

1. Randomly generate “small” polynomials f and g in $\mathcal{D}_f$, $\mathcal{D}_g$ respectively.
2. Invert f in $\mathcal{R}_q$ to obtain f_q, invert f in $\mathcal{R}_p$ to obtain f_p, and check that g is invertible in $\mathcal{R}_q$ [12].
3. The public key $\mathsf{h} = p * \mathsf{g} * \mathsf{f}_q \pmod{q}$. The private key is the pair $(\mathsf{f}, \mathsf{f}_p)$.

4.2 Encryption

NTRUEncrypt **encryption** consists of the following operations:

1. Randomly select a “small”polynomial $\mathsf{r} \in \mathcal{D}_r$.
2. Calculate the ciphertext e as $\mathsf{e} \equiv \mathsf{r} * \mathsf{h} + \mathsf{m} \pmod{q}$.

4.3 Decryption

NTRUEncrypt **decryption** consists of the following operations:

1. Calculate $\mathsf{a} \equiv$ `center`$(\mathsf{f} * e)$, where the centering operation `center` reduces its input into the interval $[A, A + q - 1]$.
2. Recover m by calculating $\mathsf{m} \equiv \mathsf{f}_p * \mathsf{a} \pmod{p}$.

To see why decryption works, use $\mathsf{h} \equiv p * \mathsf{g} * \mathsf{f}_q$ and $\mathsf{e} \equiv \mathsf{r} * \mathsf{h} + \mathsf{m}$ to obtain

$$\mathsf{a} \equiv p * \mathsf{r} * \mathsf{g} + \mathsf{f} * \mathsf{m} \pmod{q} . \tag{2}$$

For appropriate choices of parameters and `center`, this is an equality over $\mathbb{Z}$, rather than just over $\mathbb{Z}_q$. Therefore step 2 recovers m: the $p * \mathsf{r} * \mathsf{g}$ term vanishes, and $\mathsf{f}_p * \mathsf{f} * \mathsf{m} = \mathsf{m} \pmod{p}$.

4.4 The NTRU Hard Problem and One-Way Function

The one-way function underlying NTRU is:

$$\begin{aligned} F : \mathcal{D}_m \times \mathcal{D}_r &\to \mathcal{R}_q \\ F(\mathsf{m}, \mathsf{r}) &= \mathsf{m} + \mathsf{r} * \mathsf{h}, \end{aligned}$$

where $q, N \in \mathbb{Z}$, $\mathsf{p} \in \mathbb{Z}[X]$, $\mathsf{h} \in \mathcal{R}_q$ are given by the output of key generation.

Definition 4. *(The $\mathcal{P}$-NTRU problem) For a parameter set $\mathcal{P}$, we denote by $\mathsf{Succ}^{\mathsf{ow}}_{\mathsf{ntru}}(\mathcal{A}, \mathcal{P})$ the success probability of any adversary $\mathcal{A}$ for finding a preimage of F,*

$$\mathsf{Succ}^{\mathsf{ow}}_{\mathsf{ntru}}(\mathcal{A}, \mathcal{P}) = \Pr\left[\begin{array}{l|l} (\mathsf{m}', \mathsf{r}') \leftarrow \mathcal{A}(\mathsf{e}, \mathsf{h}) & (\mathsf{pk} = \mathsf{h}, \mathsf{sk}) \leftarrow \mathcal{K}, \mathsf{m} \overset{R}{\leftarrow} \widetilde{R} \\ \text{s.t. } F(\mathsf{m}', \mathsf{r}') = \mathsf{e} & \mathsf{r} \leftarrow \mathtt{genr}(\rho), \rho \overset{R}{\leftarrow} \mathcal{R}_r, \mathsf{e} = F(\mathsf{m}, \mathsf{r}) \end{array}\right].$$

Assumption 1 *(The $\mathcal{P}^1_{\mathsf{ntru}}$-NTRU assumption) For every probabalistic polynomial (in k) time algorithm $\mathcal{A}$ there exists a negligible function $\nu_{\mathcal{A}}$ such that for sufficiently large k, we have*

$$\mathsf{Succ}^{\mathsf{ow}}_{\mathsf{ntru}}(\mathcal{A}, \mathcal{P}^1_{\mathsf{ntru}}(k)) \leq \nu_{\mathcal{A}}(k)$$

.

5 Security of the NTRU one-way function

Most public key cryptosystems, such as RSA [26] or ECC [18, 22], are based on a one-way function for which there is one best-known method of attack: factoring in the case of RSA, Pollard-rho in the case of ECC. In the case of NTRU, there are *two* primary methods of approaching the one-way function, both of which must be considered when selecting a parameter set.

5.1 Combinatorial Security

Polynomials are drawn from a known space $\mathcal{S}$. This space can best be searched by using a combinatorial technique originally due to Odlyzko [16], which can be used to recover f or g from h or r and m from e. We denote the combinatorial security of polynomials drawn from $\mathcal{S}$ by Comb$[\mathcal{S}]$, and the set of binary polynomials of degree $N - 1$ with exactly d coefficients equal to 1 by $\mathcal{B}_N(d)$. Then

$$\text{Comb}[\mathcal{B}_N(d)] \geq \frac{\binom{N/2}{d/2}}{\sqrt{N}} \ . \tag{3}$$

5.2 Lattice Security

The *NTRU Lattice* $\mathcal{L}_h$ associated to a polynomial $\mathsf{h} \in \mathcal{R}$ is the lattice

$$\mathcal{L}_h = \{(\mathsf{u}, \mathsf{v}) \in \mathcal{R}^2 : \mathsf{v} \equiv \mathsf{h} * \mathsf{u}/p \pmod{q}\}, \quad \text{satisfying}$$
$$\dim(\mathcal{L}_h) = 2N \qquad \text{and} \qquad \text{Disc}(\mathcal{L}_h) = q^N.$$

Lattice-based attacks may be mounted against a ciphertext e to recover the plaintext, or against a public key h to recover the private key. This section treats lattice-based attacks on the public key; the analysis for attacks on a ciphertext is almost identical. More details can be found in [13].

An NTRUEncrypt public key h describes a $2N$-dimensional NTRU lattice containing the private key (f, g). When f is of the form $\mathsf{f} = 1 + \mathsf{pF}$. the best lattice attack on the private key involves solving a Close Vector Problem (CVP).[1] Experimentally, it has been found that an NTRU lattice of this form can usefully be characterized by two quantities

$$a = N/q, \quad c = \sqrt{4\pi e \|\mathsf{F}\| \|\mathsf{g}\|/q} \ .$$

[1] Coppersmith and Shamir [6] propose related approaches which turn out not to materially affect security.

This is to say that for constant (a, c), the experimentally observed running times for lattice reduction behave roughly as

$$\log(T) = AN + B \ ,$$

for some experimentally-determined constants A and B.

Table 1 summarizes results for breaking times from [9, 13], and more recent experiments, giving breaking times for inhomogenous NTRU lattices with different (a, c) values. We represent the security by the constants A and B. The breaking time in terms of bit security is $AN + B$. It may be converted to time in MIPS-years using the equality 80 bits $\sim 10^{12}$ MIPS-years.

c	a	A	B
1.73	0.53	0.3563	−2.263
2.6	0.8	0.4245	−3.440
3.7	2.7	0.4512	+0.218
5.3	1.4	0.6492	−5.436

Table 1. Extrapolated bit security constants depending on (c, a).

For constant (a, c), increasing N increases the breaking time exponentially. For constant (a, N), increasing c increases the breaking time. For constant (c, N), increasing a decreases the breaking time, although the effect is slight. More details on this table are given in [13]. We write

$$\text{Lattice Bit Security } b_{\text{latt}} \equiv \alpha N + \beta \ .$$

The technique known as *zero-forcing* [13, 20] can be used to reduce the dimension of an NTRU lattice problem. The precise amount of the expected performance gain is heavily dependent on the details of the parameter set; we refer the reader to [13, 20] for more details. In this paper we use the formula[2]

$$\text{Gain} \sim \left(1 - \left(1 - \prod_{i=0}^{d-1}\left(1 - \frac{r}{N-i}\right)\right)^N\right) 2^{\alpha r/2} \tag{4}$$

to determine the expected gain due to picking a pattern of r zeroes, if f has d non-zero entries, and the lattice breaking bit security goes as $\alpha N + \beta$. This will typically overestimate the gain, but we use this formula for reasons of prudence.

5.3 Decryption Failure Security

NTRU decryption can fail on validly encrypted messages if the `center` method returns the wrong value of A, or if the coefficients of $\mathsf{prg} + \mathsf{fm}$ do not lie in an interval of width q. Decryption failures leak information about the decrypter's private key [14, 25], so a `center` method must make the chance of a decryption failure vanishingly small.

The parameter sets recommended in [5] allow a decryption failure probability of about 2^{-104} for 80-bit security. In this paper, we will pick parameter sets such that there will be no decryption failure, by selecting q to be greater than the maximum possible value of $\mathsf{prg} + \mathsf{fm}$. Centering then becomes simply a matter of reducing into the interval $[0, q-1]$.

5.4 Other Security Considerations

The following parameter selection criteria must also be taken into account, although encryption and decryption will work even if they are violated.

[2] Note that this formula, used in [13], corrects the equivalent formula given in [20].

Choosing N — The degree parameter N must be prime. (See [7].)

N, q and p — The small and large moduli p and q must be relatively prime in the ring $\mathcal{R}$. Equivalently, the three quantities

$$p, \quad q, \quad X^N - 1$$

must generate the unit ideal in the ring $\mathbb{Z}[X]$. (As an example of why this is necessary, in the extreme case that p divides q, the plaintext is equal to the ciphertext reduced modulo p.)

Factorization of $X^N - 1 \pmod{q}$ — If $\mathsf{F}(X)$ is a factor of $X^N - 1 \pmod{q}$, and if $\mathsf{h}(X)$ is a multiple of $\mathsf{F}(X)$, i.e., if $\mathsf{h}(X)$ is zero in the field $K = (\mathbb{Z}/q\mathbb{Z})[X]/\mathsf{F}(X)$, then an attacker can recover the value of $\mathsf{m}(X)$ in the field K.

If q has order $t \pmod{N}$, then

$$X^N - 1 \equiv (X-1)\mathsf{F}_1(X)\mathsf{F}_2(X)\cdots\mathsf{F}_{(N-1)/t}(X) \quad \text{in } (\mathbb{Z}/q\mathbb{Z})[X] \ ,$$

where each $\mathsf{F}_i(X)$ has degree t and is irreducible mod q. If $\mathsf{F}_i(X)$ has degree t, the probability that $\mathsf{h}(X)$ or $\mathsf{r}(X)$ is divisible by $\mathsf{F}_i(X)$ is presumably $1/q^t$. To avoid attacks based on the factorization of h or r, we will require that for each prime divisor P of q, the order of $P \pmod{N}$ must be $N-1$ or $(N-1)/2$. This requirement has the useful side-effect of increasing the probability that randomly chosen f will be invertible in $\mathcal{R}_q$ [27].

Information leakage from encrypted messages — The transformation $\mathsf{a} \to \mathsf{a}(1)$ is a ring homomorphism, and so the ciphertext e has the property that

$$\mathsf{e}(1) = \mathsf{r}(1)\mathsf{h}(1) + \mathsf{m}(1) \ .$$

An attacker will know $\mathsf{h}(1)$, and for many choices of parameter set $\mathsf{r}(1)$ will also be known. Therefore, the attacker can calculate $\mathsf{m}(1)$. The larger $|\mathsf{m}(1) - N/2|$ is, the easier it is to mount a combinatorial or lattice attack to recover the msssage, so the sender should always ensure that $\|\mathsf{m}\|$ is sufficiently large. This will double the encryption time, but does not appear to lead to any attacks. One of our inputs into the parameter generation algorithm will be a lower bound for the probability that a randomly generated m will be too small.

6 Encryption schemes: NAEP

In order to protect against adaptive chosen ciphertext attacks, we must use an appropriately defined *encryption scheme*. The scheme described in [15] gives provable security in the random oracle model [2, 3]. We briefly outline it here.

NAEP uses two hash functions:

$$G : \{0,1\}^{N-l} \times \{0,1\}^l \to \mathcal{D}_r \quad H : \{0,1\}^N \to \{0,1\}^N$$

In terms of the security parameter, we wish $l = \Theta(k)$, and also $N - l = \Theta(k)$.

To encrypt a message $M \in \{0,1\}^{N-l}$ using NAEP one uses the functions

$$\texttt{compress}(x) = (x \pmod{q}) \pmod{2},$$
$$\texttt{B2P} : \{0,1\}^N \to \mathcal{D}_m \cup \text{“error”}, \quad \texttt{P2B} : \mathcal{D}_m \to \{0,1\}^N$$

The function `compress` puts the coefficients of the modular quantity $x \pmod{q}$ in to the interval $[0, q)$, and then this quantity is reduced modulo 2. The role of `compress` is simply to reduce the size of the input to the hash function H for gains in practical efficiency.The function `B2P` converts a bit string into a binary polynomial, or returns “error” if the bit string does not fulfil the appropriate criteria – for example, if it does not have the appropriate level of combinatorial security. The function `P2B` converts a binary polynomial to a bit string.

The encryption algorithm is then specified by:

1. Pick $b \overset{R}{\leftarrow} \{0,1\}^l$.
2. Let $\mathsf{r} = G(M, b)$, $\mathsf{m} = \mathtt{B2P}(\ (M||b) \oplus H(\mathtt{compress}(\mathsf{r} * \mathsf{h}))\ )$.
3. If B2P returns "error", go to step 1.
4. Let $\mathsf{e} = \mathsf{r} * \mathsf{h} + \mathsf{m} \in \mathcal{R}_q$.

Step 3 ensures that only messages of the appropriate form will be encrypted.
To decrypt a message $\mathsf{e} \in \mathcal{R}_q$ one does the following:

1. Let $\mathsf{a} = \mathtt{center}(\mathsf{f} * \mathsf{e} \pmod q)$.
2. Let $\mathsf{m} = \mathsf{f}_p^{-1} * \mathsf{a} \pmod p$.
3. Let $\mathsf{s} = \mathsf{e} - \mathsf{m}$.
4. Let $M||b = \mathtt{P2B}(\mathsf{m}) \oplus H(\mathtt{compress}(\mathtt{P2B}(\mathsf{s})))$.
5. Let $\mathsf{r} = G(M, b)$.
6. If $\mathsf{r} * \mathsf{h} = \mathsf{s} \pmod q$, and $\mathsf{m} \in \mathcal{D}_m$, then return the message M, else return the string "invalid ciphertext".

The use of the scheme NAEP introduces a single additional parameter:

l *Random Padding Length.* The length of the random padding b concatenated with M in step 1.

The ind game requires an attacker to identify the message encrypted in a single, specific ciphertext. Therefore, the random padding does not require collision resistance, but it does require preimage resistance. We therefore set $l = k$ to ensure that attacks based on guessing the random padding have a k-bit cost (where cost is defined relative to the attacker's advantage).

6.1 Instantiating NAEP: SVES-3

The EESS#1 v2 standard [5] specifies an instantiation of NAEP known as SVES-3. In SVES-3, the following specific design choices are made:

- To allow variable-length messages, a one-byte encoding of the message length in bytes is prepended to the message. The message is padded with zeroes to fill out the message block.
- The hash function G which is used to produce r takes as input M; b; an OID identifying the encryption scheme and parameter set; and a string h_{trunc} derived by truncating the public key to length l_h bits.

SVES-3 includes h_{trunc} in G so that r depends on the specific public key. Even if an attacker were to find an (M, b) that gave an r with an increased chance of a decryption failure, that (M, b) would apply only to a single public key and could not be used to attack other public keys. In the case of the parameter sets proposed in this document, there are no decryption failures and so no need to input h_{trunc} to G. We will therefore use SVES-3 but set $l_h = 0$.

7 Selecting Parameter Sets for SVES-3: Framework

Having completed our review of security considerations for NTRU parameter sets, we can now specify an algorithm that generates a parameter set for NTRUEncrypt-NAEP with a desired bit security level k. First, we specify our overall framework. Then we apply it to specific sets of constraints on the parameters.

1. Determine μ, the number of bits that must be transported in m. Pick an initial candidate N, a prime number that allows μ bits to be transported.
2. For this value of N, find values of d_F, d_g, d_r that give the required level of combinatorial security.
3. Using the bound P_{reject} given in Constraint 6, calculate the minimum integer d_{m0} and the maximum integer d_{m1} such that $N/2 - d_{m0} = d_{m1} - N/2$ and the probability that a randomly chosen binary vector will have between d_{m0} and d_{m1} 1s is greater than $1 - P_{\text{reject}}$. If d_{m0} does not give sufficient combinatorial security, increase N to the next prime and repeat this step.
4. Calculate the maximum possible width of $\mathsf{prg} + \mathsf{m} + \mathsf{pFm}$. Set q to be the first prime greater than this number.

5. Verify that the order of $q \pmod N$ is $N-1$ or $(N-1)/2$. If it is not, increase q to the next prime value until a q with a sufficiently high order is found.
6. Verify whether the lattice strength is greater than 2^k for the selected N, q, $\mathcal{D}_f$, $\mathcal{D}_g$, $\mathcal{D}_r$, $\mathcal{D}_m$. (In the case of $\mathcal{D}_m$, the check is performed for the m with $d_m = d_{m0}$, or in other words the weakest m that will occur). If the strength is greater than 2^k, terminate. Otherwise, increase N to the next highest prime number and return to step 2.

The analysis below will explain why this process is likely to terminate after a very small number of iterations.

7.1 Binary polynomials

We illustrate the method using binary polynomials. In this case, we use the following constraints.

1. Take $\mathsf{p} = 2$. Require q to be prime.
2. f will be of the form $1 + \mathsf{pF}$.
3. The polynomials F, g, r, m will be binary. Product form polynomials will not be used.
4. F, g, r will have d_F, d_g, d_r 1s respectively.
5. The system must be capable of transporting $2k$ bits of message.
6. The chance that a message representative m will be rejected due to having insufficient security, P_{reject}, will be less than 2^{-40}.
7. Subject to the constraints above, minimize bandwidth.
8. Subject to the constraints above, maximize lattice security.

Select N — For k-bit security, we require $l \geq k$, as stated in the discussion of the security of NAEP. We also want to transport $2k$ bits of message, as stated in constraint 5, and to use 8 bits to encode the length of the transported message. The total number of bits to be transported in m is therefore $3k + 8$. We set N to be the first prime greater than $3k + 8$.

Select polynomial spaces — We select values for d_F, d_g, d_r so that $\text{Comb}[\mathcal{B}_N(d_F)]$, $\text{Comb}[\mathcal{B}_N(d_g)]$, $\text{Comb}[\mathcal{B}_N(d_r)] > 2^k$. The smaller d_F, d_r are, the faster operations will be. We select d_F, d_r such that $d_F = d_r = d$, d the smallest value for which $\text{Comb}[\mathcal{B}_N(d)] \geq 2^k$. The results are shown in table 2. For all values of N in the given range, $d \sim 0.19N$; in other words d increases (slightly slower than) linearly with N. Therefore, NTRUEncrypt encryption and decryption times scale roughly as N^2 for our parameter sets.

There is no particular advantage, in performance or bandwidth, to taking g to be small, so long as it is binary. Following constraint 8, we therefore take $d_g = \lfloor N/2 \rfloor$. This is a change from practice in previous parameter sets, where d_g has typically been taken to be the same as d_f.

Select $\mathcal{D}_m$ — Table 2 gives the value of d_{m0} (and $d_{m1} = N - d_{m0}$) for each N that gives a chance of 2^{-40} of having to re-encrypt. In all cases, d_{m0} is comfortably above d_f, and so m will have sufficient combinatorial security. If d_{m0} had been below d_f, increasing N will both (a) reduce the value of d_f that gives combinatorial security and (b) increase the d_{m0} that gives the desired probability of having to re-encrypt. The process of increasing N in this step will therefore eventually terminate.

Select q — We select q subject to the requirements

$$q > \text{Max}(\text{Width}(\mathsf{prg} + \mathsf{fm})) \ ,$$
$$\text{Order}(q \pmod N) \geq (N-1)/2 \ .$$

We now consider how to calculate the width of $\mathsf{prg} + \mathsf{fm}$.

Each term in the polynomial obtained by multiplying a polynomial a by a binary polynomial b with d_b 1s can be thought of as the result of selecting d_b terms from a and summing them. If b is also binary, with d_b 1s, clearly the minimum possible value of any term in $\mathsf{a} * \mathsf{b}$ is $\max(0, d_a + d_b - N)$, and the maximum is $\min(d_a, d_b)$[3].

[3] The maximum width of the product of two binary polynomials is therefore $N/2$.

In this case $d_F = d_g = d_r = d$. The number of 1s in m, d_m, is variable, but if it is less than d_F then $\max(\mathsf{Width}(F * \mathsf{m})) < d_F$, and if it is greater than d_F then $\max(\mathsf{Width}(F * \mathsf{m})) = d_F$. So

$$\begin{aligned}\max(\mathsf{Width}(\mathsf{prg} + \mathsf{m} + \mathsf{pFm})) &= 1 + 2\mathsf{p}d = 1 + 4d \\ &\Rightarrow q > 1 + 4d \sim 0.76N.\end{aligned}$$

Taking q to be the first prime greater than $1 + 4d$ gives a q with a large enough order $\pmod N$ for almost all the (d, N) pairs under consideration. The exception is $k = 256$, $N = 787$, $d = 140$, for which the first q implied, 563, has order only 131 (mod N), and the lowest q that satisfies the order requirement turns out to be 587. The values of q obtained are given in Table 2.

Check lattice strength — Having calculated d and q, we can now calculate the lattice characteristics (a, c). For a binary polynomial b with d 1s, the centered norm is given by $|\mathsf{b}| = \sqrt{d(N - d)/N}$. and ranges from $\sqrt{d}$, when d is small, to $\sqrt{d/2}$, when $d = N/2$. The thicker f, g, r, m are, the harder the lattice problem is. We therefore calculate c for lattice attacks on (r, m) when the number of 1s in m is d_{m0} to give a lower bound on the lattice security. All the parameter sets under consideration give $c \geq 2.77$, so we can use the $c = 2.6, a = 0.8$ experimental lattice strengths in Table 1 to extrapolate the strength of (r, m) and (F, g).

For interest, we briefly consider extreme cases. If $d = 0.001N$, then we have

$$q \sim 1 + 0.004N, \quad c \sim 11.6, \quad a \sim 250 \;. \tag{5}$$

If $d = N/2$, then we have

$$q \sim 2N, \quad c = 2.066, \quad a = 0.5 \;. \tag{6}$$

This shows that as d/N increases, c will decrease to a minimum of 2.066.

As table 2 below shows, the suggested parameters clearly give a sufficient level of lattice security, even taking zero-forcing into account.

Increase N if necessary — If the strength against lattice attacks is insufficient, we increase N. This will decrease (or at worst not increase) the value of d necessary to give combinatorial security, reducing d/N. As noted in equations 5 and 6 above, as d/N decreases, c will increase. Even if c were to stay constant, increasing N would increase the lattice strength; since we increase both c and N, lattice strength will certainly increase, eventually reaching the desire strength. The process of increasing N will therefore eventually terminate.

Summary — This has rederived and justified the algorithm $\mathcal{P}^1_{\mathsf{ntru}}$ presented at the start of this paper. Table 2 summarizes the results. For different values of k, we give the corresponding N, d, and q values. These, along with $\mathsf{p} = 2$, the definition of `center` as reduction into $[0, q-1]$, and the specification $l = k$, fully parameterize the system. For interoperability, other design decisions must be made, such as the exact instantiations of the random oracles; we do not address that question in this paper.

We also present the lattice bit security b_{latt}, the number of zeroes an adversary should guess when zero-forcing r, the lattice bit security including zero-forcing $b^{\mathrm{zf}}_{\mathrm{latt}}$, the number of additions required for a convolution by f or r, the public key size $N\lceil\log_2 q\rceil$ and, for comparison, the sizes of RSA and ECC keys that give a similar level of security. We also include the number of Adds With Carry required for an ECC point operation at the same security level: details of how this figure was calculated, and discussion of the appropriate figures to compare, can be found in Appendix A. The bandwidth given is the minimum bandwidth. In the case where q is a nine-bit quantity, for example, an implementation may decide to encode each coefficient in 16 bits rather than 9.

7.2 Product-form polynomials

Next we use the method above to generate parameter sets that make use of product form polynomials for efficiency advantages. We take $\mathsf{F} = \mathsf{f}_1 * \mathsf{f}_2 + \mathsf{f}_3$, with $\mathsf{f}_1, \mathsf{f}_2, \mathsf{f}_3$ all random binary with d 1s, r to have the same form, and g to be binary with $d_g = \lfloor N/2 \rfloor$. Full details of the process are given in Appendix B. Table 3 summarizes the results, including the speedup relative to the parameters for binary polynomials investigated above, the public key size $N\lceil\log_2 q\rceil$ and the RSA and ECC figures as above.

k	N	d	d_{m0}	q	$c(\mathsf{f},\mathsf{g})$	$c(\mathsf{r},\mathsf{m})$	b_{latt}	r	$b^{\text{zf}}_{\text{latt}}$	adds	size	RSA	ECC	ECC AWC	Speedup wrt ECC
80	251	48	70	197	2.93	2.77	103.1	29	97.98	12048	2008	1024	163	112210	9.31
112	347	66	108	269	2.94	2.83	143.9	31	138.26	22902	3033	$\sim$ 2048	224	170356	7.44
128	397	74	128	307	2.93	2.84	165.1	33	159.17	29378	3501	3072	256	277280	9.44
160	491	91	167	367	2.98	2.90	205.0	35	198.75	44681	4383	4096	320	645642	14.45
192	587	108	208	439	2.97	2.91	245.7	37	239.21	63396	5193	7680	384	936618	14.77
256	787	140	294	587	2.95	2.91	330.6	41	323.45	110180	7690	15360	512	1595434	14.48

Table 2. Final Parameter Sets for different values of k using binary polynomials.

k	N	d	q	adds	Speedup wrt binary	size	RSA	ECC	ECC AWC	Speedup wrt ECC
80	251	8	293	6024	2.00	2259	1024	163	112210	18.63
112	347	11	541	11451	2.00	3370	$\sim$ 2048	224	170356	14.88
128	397	12	659	14292	2.06	3890	3072	256	277280	19.40
160	491	15	967	22095	2.02	4870	4096	320	645642	29.22
192	587	17	1229	29937	2.12	6347	7680	384	936618	31.29
256	787	22	2027	51942	2.12	8459	15360	512	1595434	30.72

Table 3. Final Parameter Sets for different values of k using product form polynomials.

8 Parameter sets generated under different constraints

The algorithm presented in the previous section is designed to ensure the minimum value of N. In this section, we consider the parameter sets that would be obtained if the constraint was to obtain the minimum value of N consistent with keeping q below 256. In practice, this means ensuring that $q = 251$ and, since $d < (q-1)/4$, that $d = 62$.

By (3), to have $\text{Comb}[\mathcal{B}_N(d)] > 2^k$ for constant d, we have

$$\log_2(N) > 2k/d + 1$$

(using the approximation $\binom{a}{b} \sim a^b$ for $b \ll a$). We therefore expect N to increase exponentially with k, and this proves to be the case as is demonstrated in Table 4. Details of the parameter generation procedure are omitted.

k	N	d	q	adds	size	Speedup wrt ECC
112	367	62	251	22754	2936	7.49
128	521	62	251	32302	4168	8.58
160	1031	62	251	63911	8248	10.10

Table 4. Final Parameter Sets for different values of k under the constraint $q = 251$.

Interestingly, the $k = 112$ parameter sets generated here are faster and result in smaller keys than the parameter sets generated by requiring the smallest N. This indicates that the parameter generation algorithm can be refined, which would be an interesting direction for future research.

9 Conclusions

We presented a framework for generation of NTRUEncrypt parameter sets and used it to generate parameter sets for different levels of bits security. The framework is robust and adaptable: if future developments in lattice analysis significantly affect breaking times, it will be possible to calculate new parameter sets that give an appropriate level of security. With different inputs to the framework, different parameter sets would be possible. For example, one might take $\mathsf{p} = 2+X$ and q a power of 2 for efficiency in performing reductions; one might require $q < 256$, increasing N as necessary, for use on 8-bit processors; one might consider an alternate encryption scheme that transported fewer bits to save bandwidth. We have also demonstrated that NTRUEncrypt remains more efficient than other well-studied cryptosystems, and shown that for increasing security levels the bandwidth required for NTRUEncrypt is less than for RSA.

This paper is merely a contribution to the systematic study of how to generate NTRUEncrypt parameter sets, but we hope a useful one.

10 Acknowledgements

We would like to thank the anonymous referees of the conference version of this paper for their comments, and Philip Hirschhorn for his help with lattice reduction experiments.

References

1. ANSI X9.62, Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA), 1999.
2. M. Bellare and P. Rogaway. Optimal asymmetric encryption. In Proc. of Eurocrypt '94, volume 950 of LNCS, pages 92–111. IACR, SpringerVerlag, 1995.
3. D. Boneh, Simplified OAEP for the RSA and Rabin functions, In proceedings of Crypto '2001, Lecture Notes in Computer Science, Vol. 2139, Springer-Verlag, pp. 275-291, 2001
4. M. Brown, D. Hankerson, J. López, and A. Menezes, Software Implementation of the NIST Elliptic Curves Over Prime Fields, *CT-RSA 2001*, D. Naccache (Ed.), LNCS 2020, 250–265, Springer-Verlag, 2001.
5. Consortium for Efficient Embedded Security, *Efficient Embedded Security Standard #1 version 2*, available from `http://www.ceesstandards.org`.
6. D. Coppersmith and A. Shamir, *Lattice Attack on NTRU*, Advances in Cryptology - Eurocrypt'97, Springer-Verlag
7. C. Gentry, Key recovery and message attacks on NTRU-composite, *Advances in Cryptology —Eurocrypt '01*, LNCS 2045. Springer-Verlag, 2001
8. D. Hankerson, J. Hernandez, A. Menezes, *Software implementation of elliptic curve cryptography over binary fields*, Proceedings of CHES 2000, Lecture Notes in Computer Science, 1965 (2000), 1-24
9. J. Hoffstein, J. Pipher, J.H. Silverman, *NTRU: A new high speed public key cryptosystem*, in Algorithmic Number Theory (ANTS III), Portland, OR, June 1998, Lecture Notes in Computer Science 1423 (J.P. Buhler, ed.), Springer-Verlag, Berlin, 1998, 267–288. See also `http://www.ntru.com`.
10. J. Hoffstein and J. H. Silverman. Optimizations for NTRU. In Publickey Cryptography and Computational Number Theory. DeGruyter, 2000. Available at [4].
11. J. Hoffstein and J. H. Silverman, Random Small Hamming Weight Products With Applications To Cryptography, Discrete Applied Mathematics, to appear, Available from `http://www.ntru.com`.
12. J. Hoffstein and J. H. Silverman. Invertibility in truncated polynomial rings. Technical report, NTRU Cryptosystems, October 1998. Report #009, version 1, available at `http://www.ntru.com`.
13. J. Hoffstein, J. H. Silverman, W. Whyte, Estimated Breaking Times for NTRU Lattices, Technical report, NTRU Cryptosystems, June 2003 Report #012, version 2, available at `http://www.ntru.com`.
14. N. A. Howgrave-Graham, P. Nguyen, D. Pointcheval, J. Proos, J. H. Silverman, A. Singer, W. Whyte, *The Impact of Decryption Failures on the Security of NTRU Encryption*, Advances in Cryptology—Crypto 2003, Lecture Notes in Compputer Science 2729, Springer-Verlag, 2003, 226-246.
15. N. Howgrave-Graham, J. H. Silverman, A. Singer and W. Whyte. NAEP: Provable Security in the Presence of Decryption Failures IACR ePrint Archive, Report 2003-172, `http://eprint.iacr.org/2003/172/`
16. N. A. Howgrave-Graham, J. H. Silverman, W. Whyte, A Meet-in-the-Middle Attack on an NTRU Private key, Technical report, NTRU Cryptosystems, June 2003. Report #004, version 2, available at `http://www.ntru.com`.
17. B. Kaliski, Comments on SP 800-57, Recommendation for Key Management, Part 1: General Guidelines. Available from `http://csrc.nist.gov/CryptoToolkit/kms/CommentsSP800-57Part1.pdf`.

18. N. Koblitz. *Elliptic curve cryptosystems.* Mathematics of Computation, 48, pages 203–209, 1987.

19. A. K. Lenstra, E. R. Verheul, *Selecting cryptographic key sizes*, Journal of Cryptology vol. 14, no. 4, 2001, 255-293. Available from `http://www.cryptosavvy.com`.

20. A. May, J.H. Silverman, *Dimension reduction methods for convolution modular lattices*, in Cryptography and Lattices Conference (CaLC 2001), J.H. Silverman (ed.), Lecture Notes in Computer Science 2146, Springer-Verlag, 2001

21. T. Meskanen and A. Renvall. Wrap Error Attack Against NTRUEncrypt. Proc. of WCC '03.

22. V. Miller. *Uses of elliptic curves in cryptography.* In Advances in Cryptology: Crypto '85, pages 417–426, 1985.

23. NIST, Digital Signature Standard, FIPS Publication 186-2, February 2000.

24. NIST Special Publication 800-57, Recommendation for Key Management, Part 1: General Guideline, January 2003. Available from `http://csrc.nist.gov/CryptoToolkit/kms/guideline-1-Jan03.pdf`.

25. J. Proos *Imperfect Decryption and an Attack on the NTRU Encryption Scheme*, IACR ePrint Archive, report 02/2003. `http://eprint.iacr.org/2003/002/`.

26. R. Rivest, A. Shamir, L. M. Adleman, *A method for obtaining digital signatures and public-key cryptosystems*, Communications of the ACM, 21 (1978), 120-126.

27. J. H. Silverman, Invertibility in Truncated Polynomial Rings, Technical report, NTRU Cryptosystems, October 1998 Report #009, version 1, available at `http://www.ntru.com`.

28. Robert D. Silverman, A Cost-Based Security Analysis of Symmetric and Asymmetric Key Lengths. *RSA Labs Bulletin 13*, April 2000. available from `http://www.rsasecurity.com/rsalabs`.

A Comparing ECC times to NTRUEncrypt times

In this section we give a comparison, in terms of basic operations, of elliptic curve point multiplication and NTRUEncrypt polynomial operations. The speed of elliptic curve point multiplications (in software) for the prime fields given in [1, 23] is analyzed in [4]. The analysis in [4] is most complete for the smallest NIST prime field F_{p192}, where $p_{192} = 2^{192} - 2^{64} - 1$. The prime p_{192} is slightly less than a power of 2^{32} (a typical machine word size) and has a sparse bit representation, yielding added efficiencies for various operations such as modular reduction.

The first three columns of Table 5 appear in [4, Table 10] and give the number of basic operations required for an average point multiplication on the NIST p_{192} elliptic curve. The authors of [4] note that "95.4% of the total execution time was spent on these basic operations."

Operations in the first three columns of Table 5 are modulo p. We denote by AWC. the amount of time it takes to perform a single 32 bit addition-with-carry, and estimate the time for an elliptic curve point multiplication in terms of AWC. For example, a single addition modulo p_{192} takes 6 AWC.[4]

To estimate AWC numbers for other security levels, we follow [4, Table 9], which gives running times for point multiplications for all the recommended fields. The estimated number of AWC for field p_i is simply the number of AWC for p_{192} times the ratio of the running times for p_i and p_{192}. The results are shown in Table 6. Our figures are based on the figures of [4] that do not use precomputation, as ECC encryption and decryption typically involve one point multiplication on an arbitrary base point for which precomputation cannot have been performed.[5]

The NTRUEncrypt performance figures given are the time for a single convolution, calculated as

$$\text{Time} = dN \quad \text{(Binary polynomials)} \ ,$$
$$\text{Time} = 3dN \quad \text{(Product-form polynomials)} \ .$$

This slightly underestimates the time for a convolution, in which each of the N coefficients is produced by a combination of d additions (without carry) and 1 to d reductions mod q. However, the overhead due to the reductions is not great; on a 32-bit machine, for example, each reduction can be accomplished in $\log_2(d)$ subtractions. We therefore consider the figures presented to be a good first-order approximation to the actual running times.

[4] Plus potentially one subtraction of p, which we consider free due to the form of p_{192}.

[5] Signing is more likely to use precomputation, increasing speeds about 3.5-fold.

Field operation	# of calls	Percentage of total time	AWC per call	Number of AWC
Addition (Alg 1)	1137	5.8%	6.00	6822.00
Subtraction (Alg 2)	1385	7.4%	6.28	8703.93
Integer multiplication (Alg 3)	1213	38.3%	37.14	45048.72
Integer squaring (Alg 4)	934	28.20%	35.5	33169.03
Fast reduction (Alg 7)	2147	14.8%	8.11	17407.86
Modular inversion (Alg 8)	1	0.9%	1058.59	1058.59
Total		95.4%		112210.14

Table 5. Average number of function calls and percentage of time spent on the basic field operations in executions of [4, Algorithm 10] for elliptic curve point multiplication for the p_{192} curve. Data in first three columns is from [4, Table 10]. The algorithm numbers in Table 5 refer to the algorithms described in [4].

Field	Time	AWC
p_{192}	2144	112210
p_{224}	3255	170356
p_{256}	5298	277280
p_{384}	17896	936618
p_{521}	30484	1595433

Table 6. Estimated number of AWC for each of the NIST recommended finite fields, derived from [4, Table 9].

In summary, NTRUEncrypt convolution operations with binary polynomials are $7.5 - 15$ times faster, and NTRUEncrypt convolutions with product-form polynomials are $15 - 30$ times faster than ECC point multiplications, at the same security level. This figure leaves out the time required for any hash function operations. For encryption, ECC requires an additional point multiplication to a known base point, which increases encryption times by a factor of $1.3 - 2$. For decryption, NTRUEncrypt-NAEP requires an additional encryption operation for the consistency, increasing decryption times by a factor of 2.

B Details of Product Form Calculations

B.1 Combinatorial Security of Product-form Polynomials

Product-form polynomials [10, 11] are polynomials of the form $\mathsf{a}_1 * \mathsf{a}_2$ or $\mathsf{a}_1 * \mathsf{a}_2 + \mathsf{a}_3$. The advantage of polynomials of this form is that they can be specified more compactly, and multiplied by more quickly, than binary polynomials with the same level of combinatorial security, though at the cost of requiring more RAM.

In this paper we will only consider the combinatorial security of polynomials of the form $\mathsf{a} = \mathsf{a}_1 * \mathsf{a}_2 + \mathsf{a}_3$, where $\mathsf{a}_1, \mathsf{a}_2, \mathsf{a}_3$ are all binary with $d_{a_1}, d_{a_2}, d_{a_3}$ 1s respectively, $d_{a1} = d_{a2} = d_{a3} = d_a$, and there are no further constraints on a. If $\mathcal{P}_N(d)$ is the set of all polynomials of this form, then $\mathrm{Comb}[\mathcal{P}_N(d)] \geq \min\Bigg(\binom{N-\lceil N/d \rceil}{d-1}^2, \max\left(\binom{N-\lceil \frac{N}{d} \rceil}{d-1}\binom{N-\lceil \frac{N}{d-)} \rceil}{d-2}, \binom{N}{2d}\right), \max\left(\binom{N}{d}\binom{N}{d-1}, \binom{N-\lceil \frac{N}{2d} \rceil}{2d-1}\right) \Bigg)$.

Previous parameter sets [5] have suggested using product-form polynomials $\mathsf{a} = \mathsf{a}_1 * \mathsf{a}_2 + \mathsf{a}_3$, where the product polynomial a is constrained to be binary. However, this increases the time to generate those polynomials, and more so as the security parameter k increases. For reasons of efficiency the parameter generation algorithm in this paper does not require binary output polynomials.

B.2 Parameter Set Generation

We work from the same constraints as in section 7.1, except that:

1. F will be of the form $\mathsf{f}_1 * \mathsf{f}_2 + \mathsf{f}_3$, with $\mathsf{f}_1, \mathsf{f}_2, \mathsf{f}_3$ all random binary.
2. r will be of the form $\mathsf{r}_1 * \mathsf{r}_2 + \mathsf{r}_3$, with $\mathsf{r}_1, \mathsf{r}_2, \mathsf{r}_3$ all random binary.
3. $\mathsf{f}_1, \mathsf{f}_2, \mathsf{f}_3, \mathsf{r}_1, \mathsf{r}_2, \mathsf{r}_3$ will have d 1s; g will have d_g 1s.

As before, we set N to be the first prime greater than $3k+8$. The parameter set generation then proceeds as follows.

Select polynomial spaces — Select the smallest d that gives the desired level of combinatorial security, and take $d_g = N/2$ to give the greatest possible lattice security. Table 7 shows the resulting values for d and the corresponding Hamming weight of F, $\mathsf{Hw(F)}(= \mathsf{Hw(r)})$. For all values of N, $d_F = d_r \sim 0.03N$. As in the previous section, d increases slightly slower than linearly with N, so NTRUEncrypt encryption and decryption times scale approximately as N^2. The value obtained for d_{m0} only depends on N and does not change.

Select q — Select the smallest prime q such that $\mathrm{Order}(q \pmod N)) \geq (N-1)/2$ and $q > \mathrm{Max}(\mathrm{Width}(\mathsf{prg} + \mathsf{fm}))$. For both $\mathsf{f} * \mathsf{m}$ and $\mathsf{r} * \mathsf{g}$, one of the operands is binary but the other is product-form, so the width of the product $\mathsf{a} * \mathsf{b}$ is no longer bounded by $\min(\mathsf{Hw}(\mathsf{a}, \mathsf{b}))$. However, since one of the operands is binary, the width is certainly bounded by $\mathsf{Hw(a)}$, where a is the non-binary input polynomial. Therefore,

$$\begin{aligned} \max(\mathsf{Width}(\mathsf{prg} + \mathsf{m} + \mathsf{pfm})) &= 1 + 2\mathsf{p}d(d+1) = 1 + 4d(d+1) \\ &\Rightarrow q > 1 + 4d(d+1) \Rightarrow q \sim cN^2. \end{aligned}$$

Applying the requirement that the order of $q \pmod N$ be large, we increase $q(N = 397)$ and $q(N = 491)$. The other values of q are unaffected.

Check lattice strength — We now calculate the lattice characteristics (a, c). For g we use the standard centered norm. For f, the situation is more complicated. Roughly speaking, centered norms obey the *pseudo-multiplicative* and *pseudo-additive rules*

$$|\mathsf{a} * \mathsf{b}| \sim |\mathsf{a}| * |\mathsf{b}|, \quad |\mathsf{a} + \mathsf{b}| \sim \sqrt{|\mathsf{a}|^2 + |\mathsf{b}|^2}\ .$$

The centered norm $|\mathsf{F}|$ will in general be $|\mathsf{F}| \sim \sqrt{d^2(N-d)^2/N^2 + d(N-d)/N}$. However, in the case where F is binary, $|\mathsf{F}|$ will take the considerably lower value $|\mathsf{F}| = \sqrt{D(N-D)/N}, \quad D = d^2 + d$. Although it will be extremely rare for randomly generated product-form F to be binary, we will use this lower value in calculating c. For all parameter sets under consideration we obtain the result $c > 1.73$, so we can use the extrapolation line obtained at $c = 1.73$, presented in table 1 above, to estimate lattice strength. Estimating the effects of zero-forcing is also harder in this case, because the number of zeroes in f, r is now variable. We will assume that the product-form polynomials can be approximated by dropping $d(d+1)$ balls into N boxes. The expected number of empty boxes is

$$\mathsf{E}(\text{zeroes}) = N(1 - 1/N)^{d(d+1)}\ .$$

We use this expected number of zeroes in our zero-forcing calculations.[6][7]

We can also estimate $||\mathsf{F}||$ by estimating the expected number of 0s, 1s, 2s, and so on, and calculating the centered norm using Equation 1. This third estimate of $||\mathsf{F}||$ gives a higher c value than the other methods described above. We denote it by $c_{0,1,2,3}$ in Table 7.

Table 7 summarizes the results for lattice strength for product form polynomials. The value r is the number of zeroes an adversary should guess when zero-forcing. We also give $c_{\mathsf{f}_1,\mathsf{f}_2,\mathsf{f}_3}$, the expected value of c as calculated from the norms of $\mathsf{f}_1, \mathsf{f}_2, \mathsf{f}_3$, and $c_{0,1,2,3}$, the expected value of c as calculated from the expected numbers of 0s, 1s, 2s and 3s in F. Both of these measures give a higher value for c than the one we use, demonstrating that in general the lattice security will be considerably above the extrapolation line based on $c = 1.73$. The final parameter sets are given in Table 3.

[6] If a given polynomial has more than the expected number of zeroes, this will help the attacker by improving their chances of guessing a pattern, but also harm them because the fewer entries a polynomial has the greater its norm, and the harder the associated lattice problem, will in general be.

[7] An attacker could also attempt zero-forcing by inverting h and looking for patterns of zeroes in g. This approach would be worthwhile if there were fewer zeroes in F than in g, but for the parameter sets under consideration, this is not the case and zero-forcing on F will always be more effective.

k	N	d	d/N	$\mathsf{Hw}(\mathsf{F})$	q	$c(\mathsf{f},\mathsf{g})$	$c(\mathsf{r},\mathsf{m})$	b_{latt}	r	$b_{\mathrm{latt}}^{\mathrm{zf}}$	$c_{\mathsf{f}_1,\mathsf{f}_2,\mathsf{f}_3}$	E_0	E_1	E_2	E_3	E_4	$c_{0,1,2,3}$
80	251	8	0.032	72	293	2.57	2.43	87.2	20	80.1	2.76	188	55	7	1	0	2.79
112	347	11	0.032	132	541	2.21	2.13	117.8	16	118.7	2.59	237	90	18	2	0	2.56
128	397	12	0.030	156	659	2.24	2.17	136.3	17	136.6	2.50	268	105	21	4	0	2.53
160	491	15	0.031	210	967	2.08	2.02	171.3	16	170.1	2.42	301	146	38	4	0	2.52
192	587	17	0.029	306	1229	2.02	1.97	203.3	14	204.6	2.39	348	180	51	8	0	2.41
256	787	22	0.028	462	2027	1.78	1.75	278.1	14	276.7	2.27	414	261	93	17	1	2.28

Table 7. Lattice constant c for different values of k