Timing Attacks on NTRUEncrypt Via Variation in the Number of Hash Calls

Joseph H. Silverman and William Whyte

NTRU Cryptosystems, Inc.

Abstract. This report studies timing attacks on NTRUEncrypt based on variation in the number of hash calls made on decryption. The attacks apply to the parameter sets of [8,6]. To mount the attacker, an attacker performs a variable amount of precomputation, then submits a relatively small number of specially constructed ciphertexts for decryption and measures the decryption times. Comparison of the decryption times with the precomputed data allows the attacker to recover the key in greatly reduced time compared to standard attacks on NTRUEncrypt. The precomputed data can be used for all keys generated with a specific parameter set and tradeoffs exist that increase the amount of precomputation in order to decrease the time required to recover an individual key. For parameter sets in [3] that claim k-bit security but are vulnerable to this attack, we find that an attacker can typically recover a single key with about $k/2$ bits of effort.

Finally, we describe a simple means to prevent these attacks by ensuring that all operations take a constant number of SHA calls. The recommended countermeasure does not break interoperability with the parameter sets of [8,6] and has only a slight effect on performance.

1 NTRUEncrypt Overview

In this section we briefly review how NTRUEncrypt works in order to set notation. For further details, see [2,3,6]. Recall that NTRUEncrypt uses the ring of truncated polynomials (also sometime called the ring of convolution polynomials)

$$\mathbf{Z}[X]/(X^N - 1).$$

We denote multiplication in this ring by $*$. At various stages of encryption and decryption the coefficients of these polynomials are reduced modulo q and/or modulo p, where p and q are relatively prime integers. This reduction is always performed so that the reduced coefficients lie in the range from 0 to $p-1$ (respectively 0 to $q-1$). In particular, reduction modulo p and reduction modulo q do not commute with one another. For example,

$$(11 \bmod 7) \bmod 2 = 4 \bmod 2 = 0 \quad \text{and} \quad (11 \bmod 2) \bmod 7 = 1 \bmod 7 = 1.$$

For simplicity in this note, we restrict attention to the case $p = 2$, in which case various polynomials are chosen to be binary (i.e., all coefficients 0 or 1), and in some cases with a fixed number of zeros and ones. To ease notation, we let

M. Abe (Ed.): CT-RSA 2007, LNCS 4377, pp. 208–224, 2007.

$$\mathcal{B}_N = \{\text{binary polynomials}\},$$
$$\mathcal{B}_N(d) = \{\text{binary polynomials with exactly } d \text{ ones}\}.$$

An NTRUEncrypt private key consists of a pair of (binary) polynomials f and g. The associated public key is the polynomial

$$\mathsf{h} = p * \mathsf{f}_q^{-1} * \mathsf{g} \bmod q,$$

where f_q^{-1} denotes the inverse of f modulo q. Similarly, we let f_p^{-1} denote the inverse of f modulo p. To speed decryption, the polynomial f is often taken in the form $\mathsf{f} = 1 + p\mathsf{F}$ with $\mathsf{F} \in \mathcal{B}_N(d_F)$, in which case $\mathsf{f}_p^{-1} = 1$. See [3,6] for a discussion. The special form $1 + p\mathsf{F}$ will play an important role in our attack.

Encryption and decryption use two hash functions. We denote them by G and H as in [5]. In practice, they are built using either SHA-1 or SHA-256 in various ways, depending on the desired security level, see [8]. The attack that we describe is based on the fact that the number of SHA calls required by G depends on the input to G. Thus by measuring decryption time, an attacker may obtain information about the input to G, which in turn reveals information about the private key f.

The encryption process works as follows.

$\mathsf{M} \in \mathcal{B}_N$	*Padded plaintext*
$\mathsf{r} = G(\mathsf{M}) \in \mathcal{B}_N(d_r)$	*Randomizer*
$\mathsf{m}' = \mathsf{M} \oplus H(\mathsf{r} * \mathsf{h} \bmod q)$	*Masked message representative*
$\mathsf{e} = (\mathsf{r} * \mathsf{h} + \mathsf{m}') \bmod q$	*Ciphertext*

The decryption algorithm first recovers the (padded) message representative m' and plaintext M and then uses them to recreate the blinding value r and verify that $(\mathsf{m}', \mathsf{e})$ is a valid NTRUEncrypt pair.

$\mathsf{m}' = \big((\mathsf{f} * \mathsf{e} \bmod q) \bmod p\big) * \mathsf{f}_p^{-1} \bmod p$	*Recover candidate* m'
$\mathsf{M} = \mathsf{m}' \oplus H(\mathsf{e} - \mathsf{m}' \bmod q)$	*Unmask* m' *to get* M
$\mathsf{r} = G(\mathsf{M})$	*Recover* r *used in encryption*
Verify that e equals $\mathsf{r} * \mathsf{h} + \mathsf{m}' \bmod q$	

The basis for our timing attack lies in the way in which G uses SHA to create r from M. Note that on decryption, e and m' completely determine M, and therefore the time to calculate $\mathsf{r} = G(\mathsf{M})$. The blinding value r is required to be a binary polynomial with exactly d_r ones, and the process described in [8] for creating r from M may take a different number of SHA calls for different values of M. Later we will describe exactly how this is done, but for now we simply observe that this leads to a time variation that an attacker may be able to measure and show how these timing observations may be converted into information about the private key f. We also note that a simple countermeasure, as described in Section 7, is to perform a few extra SHA calls to ensure that every decryption takes the same amount of time.

2 The Time Trail of a Ciphertext

As we saw in Section 1, the number of hash calls required to create the blinding value r from a message representative/ciphertext pair $(\mathsf{m}', \mathsf{e})$ may be different for different pairs $(\mathsf{m}', \mathsf{e})$. Each hash call requires a nontrivial amount of time, so an adversary might be able to determine how many hash calls Bob uses in attempting to decrypt a (possibly bogus) ciphertext e.

In practice, there will be a number K so that the number of hash calls required to create r from $(\mathsf{m}', \mathsf{e})$ is usually either K or $K+1$. For each pair $(\mathsf{m}', \mathsf{e})$, regardless of whether or not it is a valid NTRUEncrypt pair, we define $\mathsf{r}(\mathsf{m}', \mathsf{e})$ to be the output from the decryption algorithm,

$$\mathsf{r}(\mathsf{m}', \mathsf{e}) = G\big(\big(\mathsf{m}' + H(\mathsf{e} - \mathsf{m}' \bmod q)\big) \bmod 2\big),$$

and we set $\beta(\mathsf{m}', \mathsf{e}) \in \{0, 1\}$ by the rule

$$\beta(\mathsf{m}', \mathsf{e}) = \begin{cases} 0 & \text{if it takes } \le K \text{ hashes to create } \mathsf{r}(\mathsf{m}', \mathsf{e}), \\ 1 & \text{if it takes } > K \text{ hashes to create } \mathsf{r}(\mathsf{m}', \mathsf{e}). \end{cases}$$

Note that for known $(\mathsf{m}', \mathsf{e})$, the computation of $\mathsf{r}(\mathsf{m}', \mathsf{e})$, and thus of $\beta(\mathsf{m}', \mathsf{e})$, requires no private knowledge.

For a given $(\mathsf{m}', \mathsf{e})$, we look also at the rotations $(X^i\mathsf{m}', X^i\mathsf{e})$ for $i = 0, 1, \ldots$. We define the *Time Trail* of $(\mathsf{m}', \mathsf{e})$ to be the binary vector

$$\begin{aligned} T(\mathsf{m}', \mathsf{e}) &= \big(\beta(\mathsf{m}', \mathsf{e}), \beta(X\mathsf{m}', X\mathsf{e}), \beta(X^2\mathsf{m}', X^2\mathsf{e}), \ldots, \beta(X^{N-1}\mathsf{m}', X^{N-1}\mathsf{e})\big) \\ &\in \{0, 1\}^N. \end{aligned}$$

The Time Trail tells us how many hashes are required for each of the rotations of the pair $(\mathsf{m}', \mathsf{e})$.

Let P be the probability that a randomly chosen $(\mathsf{m}', \mathsf{e})$ requires (at most) K hash calls and similarly $1 - P$ is the probability that a randomly chosen $(\mathsf{m}', \mathsf{e})$ requires (at least) $K + 1$ hash calls. If neither P nor $1 - P$ is too small, then the probability that two pairs $(\mathsf{m}'_1, \mathsf{e}_1)$ and $(\mathsf{m}'_2, \mathsf{e}_2)$ have the same time trails is quite small. More precisely, it is not hard to derive the formula

$$\mathrm{Prob}\big(T(\mathsf{m}'_1, \mathsf{e}_1) = T(\mathsf{m}'_2, \mathsf{e}_2)\big) = (1 - 2P + 2P^2)^N.$$

This holds under the assumption that different entries in the vectors are random and independent: this is correct so long as the main variation in running time comes from the hash calls, and so long as the output of SHA-1 is in some sense random. We otherwise defer the derivation of this formula to Section A.1.

3 A Timing Attack Based on Variable Number of Hash Calls

In this section we explain how an adversary Oscar might use time trails in order to derive information about Bob's private key.

Oscar first chooses a collection of (possibly bogus) ciphertexts $\mathcal{E}$ (i.e., $\mathcal{E}$ is a collection of polynomials modulo q). He also chooses a set of message representative values $\mathcal{M}$ (i.e., a collection of binary polynomials) with the property that $\mathcal{M}$ contains many of the polynomials in the set

$$\big\{((\mathsf{f} * \mathsf{e} \bmod q) \bmod 2) * (\mathsf{f}^{-1} \bmod 2) : \mathsf{e} \in \mathcal{E}\big\}.$$

Note that this is exactly the set of message representative that Bob would create during the process of decrypting the ciphertexts in $\mathcal{E}$. More precisely, we assume that the probability

$$p_{\mathcal{M},\mathcal{E}} := \operatorname*{Prob}_{\mathsf{e}\in\mathcal{E}}\big(((\mathsf{f} * \mathsf{e} \bmod q) \bmod 2) * (\mathsf{f}^{-1} \bmod 2) \in \mathcal{M}\big)$$

is not too small.

Before starting the active part of the attack, Oscar creates a table consisting of the time trails of every pair in $\mathcal{M} \times \mathcal{E}$. In other words, he creates a searchable list of binary vectors

$$\big(T(\mathsf{m}', \mathsf{e}) : \mathsf{m}' \in \mathcal{M} \text{ and } \mathsf{e} \in \mathcal{E}\big\}.$$

Thus the precomputation required for the attack has time and space requirements that are $O\big(\#\mathcal{M} \cdot \#\mathcal{E}\big)$.

To initiate the attack, Oscar chooses a random $\mathsf{e} \in \mathcal{E}$, sends it to Bob, and records how long it takes Bob to decipher it. Note that the use of NAEP padding [5] as described above ensures that bogus ciphertexts will be rejected. But in this case the attacker does not care that the ciphertexts are rejected, so long as he can obtain timing information. This timing information enables him to determine how many hash calls are required to create r from the ciphertext e and the message representative

$$\mathsf{m}'(\mathsf{e}) := ((\mathsf{f} * \mathsf{e} \bmod q) \bmod 2) * (\mathsf{f}^{-1} \bmod 2),$$

so Oscar finds the value of $\beta(\mathsf{m}'(\mathsf{e}), \mathsf{e})$. Of course, Oscar does not know the value of $\mathsf{m}'(\mathsf{e})$.

In a similar manner, Oscar sends each of the polynomials

$$\mathsf{e}, \quad X\mathsf{e}, \quad X^2\mathsf{e}, \quad X^3\mathsf{e}, \quad \ldots, \quad X^{N-1}\mathsf{e}$$

to Bob and obtains the values $\beta(\mathsf{m}'(X^i\mathsf{e}), X^i\mathsf{e})$ for $i = 0, 1, \ldots, N-1$. We now observe that

$$\begin{aligned}\mathsf{m}'(X^i\mathsf{e}) &= ((\mathsf{f} * X^i\mathsf{e} \bmod q) \bmod 2) * (\mathsf{f}^{-1} \bmod 2)\\ &= X^i * ((\mathsf{f} * \mathsf{e} \bmod q) \bmod 2) * (\mathsf{f}^{-1} \bmod 2)\\ &= X^i\mathsf{m}'(\mathsf{e})\end{aligned}$$

Thus Oscar has determined $\beta(X^i\mathsf{m}'(\mathsf{e}), X^i\mathsf{e})$ for $i = 0, 1, \ldots, N-1$, so he knows the time trail $T(\mathsf{m}'(\mathsf{e}), \mathsf{e})$ of the pair $(\mathsf{m}'(\mathsf{e}), \mathsf{e})$.

Oscar now searches his precomputed list and, with reasonable probability, finds a small number of possibilities for $(\mathsf{m}'(\mathsf{e}), \mathsf{e})$. In other words, Oscar now has a known polynomial e and a known polynomial m' so that when Bob decrypted e, Bob got m' as the message representative. Hence Oscar knows that there is an equation of the form

$$\mathsf{m}' * \mathsf{f} \equiv (\mathsf{f} * \mathsf{e} \bmod q) \pmod 2. \tag{1}$$

(More precisely, Oscar knows e and he has a small list of possible m', one of which satisfies (1). In Section 4.1 we discuss how Oscar can disambiguate between the possible m' in a plausible attack scenario.) Equation (1) certainly contains a significant amount of information concerning Bob's private key f, although exploiting this information will depend on the specific form of e. For example, if the elements of $\mathcal{E}$ consist of polynomials with very few nonzero coefficients, then equation (1) may give information concerning the spacing between the nonzero coefficients of f. In Section 4 we describe a specific collection $\mathcal{E}$ that leads to a practical hash timing attack when the key f has the form $\mathsf{f} = 1 + p\mathsf{F}$. (This form is sometimes used to decrease decryption time.)

4 A Practical Hash Timing Attack for $\mathsf{f} = 1 + 2F$ — Theory

For this section we consider the case where $p = 2$, so q is necessarily odd, and where private keys have the form

$$\mathsf{f} = 1 + 2\mathsf{F} \qquad \text{for some binary polynomial } \mathsf{F} \in \mathcal{B}_N(d_F).$$

The parameters recommended by NTRU Cryptosystems currently take this form [3,6,8] Note that the inverse $\mathsf{f}_2^{-1} = (\mathsf{f} \bmod 2)^{-1}$ is equal to 1, so the formula that Bob uses to recover the message representative m' from a ciphertext e simplifies to

$$\mathsf{m}'(\mathsf{e}) = (\mathsf{f} * \mathsf{e} \bmod q) \bmod 2. \tag{2}$$

For later computations, we write $\mathsf{F} = \sum_j F_i X^j$ with $F_j \in \{0, 1\}$, and for any $j \in \mathbf{Z}$, we let F_j denote the coefficient $F_{(j \bmod N)}$.

Let $\lambda = 2\lceil q/8 \rceil$ be the smallest even integer that is larger than $q/4$. To mount the attack, Oscar uses the set of (bogus) ciphertexts defined by

$$\mathcal{E} = \bigl\{\lambda + \lambda X^i : 1 \le i < N\bigr\}.$$

In other words, the $\mathsf{e} \in \mathcal{E}$ are polynomials with two coefficients equal to λ and all other coefficients equal to 0. In summary, Oscar's attack is:

1. Choose a value δ.
2. Let $\mathcal{E} = \{\mathsf{e}_i = \lambda + \lambda X^i : 0 \le i \le (N-1)/2\}$ and $\mathcal{M} = \mathcal{B}_N(0 < d \le \delta)$.
3. Precompute and store in a suitably searchable database the time trails $T(\mathsf{m}', \mathsf{e})$ for every $\mathsf{m}' \in \mathcal{M}$ and every $\mathsf{e} \in \mathcal{E}$.

4. For each i, send e_i, $X\mathsf{e}_i, \ldots, X^{N-1}\mathsf{e}_i$ to Bob and use the decryption times to determine the time trail $T(\mathsf{m}'(\mathsf{e}_i), \mathsf{e}_i)$ as described in Section 3.
5. Search the database to determine $\mathsf{m}'(\mathsf{e}_i)$, either exactly or up to a small number of choices. Once a candidate $\mathsf{m}'(\mathsf{e}_i)$ is found, validate it by the methods below.
6. Use the resulting values of $\mathsf{m}'(\mathsf{e}_i)$ to reconstruct F, either by an exact computation or by cutting down on the search space for F and performing a direct search of that subset.

We now need to figure out the possible values of $\mathsf{m}'(\mathsf{e})$ that arise in (2) when Bob decrypts the ciphertexts in $\mathcal{E}$. During decryption, Bob first computes

$$\begin{aligned} a &= \mathsf{f} * \mathsf{e} \bmod q \\ &\equiv (1+2F) * (\lambda + \lambda X^i) \pmod q \\ &\equiv \lambda + \lambda X^i + \sum_{j=0}^{N-1} 2\lambda(F_j + F_{j-i})X^j. \end{aligned}$$

Thus the j^{th} coefficient of a is given by

$$a_j = \begin{cases} \lambda(1 + 2F_0 + 2F_{-i}) \bmod q & \text{if } j = 0, \\ \lambda(1 + 2F_i + 2F_0) \bmod q & \text{if } j = i, \\ \lambda(2F_j + 2F_{j-i}) \bmod q & \text{if } j \neq 0, i \end{cases} \tag{2}$$

The key observation is that since $\lambda = 2\lceil q/8 \rceil$ is just slightly larger than $q/4$, the quantities on the righthand side of 2 are between 0 and $q-1$ unless $F_j = F_{j-i} = 1$, in which case they are greater than q. Thus there is nontrivial reduction modulo q if and only if $F_j = F_{j-i} = 1$, which implies that

$$a_j = \begin{cases} \lambda,\ 2\lambda,\ \text{or } 3\lambda & \text{if } F_j = 0 \text{ or } F_{j-i} = 0, \\ 4\lambda - q \text{ or } 5\lambda - q & \text{if } F_j = F_{j-i} = 1. \end{cases}$$

The next step is to reduce a modulo 2, which yields the message representative $\mathsf{m}'(\mathsf{e}_i)$ for the (bogus) ciphertext $\mathsf{e}_i = \lambda + \lambda X^i$. Recalling that λ is even and q is odd, we see that

$$a_j \bmod 2 = \begin{cases} 0 & \text{if } F_j = 0 \text{ or } F_{j-i} = 0, \\ 1 & \text{if } F_j = F_{j-i} = 1. \end{cases}$$

This gives the following explicit description of $\mathsf{m}'(\mathsf{e}_i)$:

$$\mathsf{m}'(\mathsf{e}_i) = \sum_{j=0}^{N-1} \left(\begin{array}{l} 1 \text{ if } F_j = F_{j-i} = 1 \\ 0 \text{ otherwise} \end{array} \right) X^j \ ,$$

which in turn yields the following partial information about F:

$$\mathsf{F}(\mathsf{e}_i) = \sum_{j=0}^{N-1} \begin{pmatrix} 1 \text{ if } \mathsf{m}'(\mathsf{e}_i)_j = 1 \\ \quad \text{or } \mathsf{m}'(\mathsf{e}_i)_{j+i} = 1 \\ 0 \text{ if } \mathsf{m}'(\mathsf{e}_i)_{j-i} = 1 \text{ and } \mathsf{m}'(\mathsf{e}_i)_j \neq 1 \\ \quad \text{or } \mathsf{m}'(\mathsf{e}_i)_{j+2i} = 1 \text{ and } \mathsf{m}'(\mathsf{e}_i)_{j+i} \neq 1 \\ ? \text{ unknown otherwise} \end{pmatrix} X^j \ ,$$

Therefore, every m' with $d_{m'}$ ones that Oscar can recover will yield $d_{m'}$ pairs of non-zero coefficients of F, allowing him to reduce the search space for F. To be precise, defining the "left-hand" member of a pair in the obvious way, we see that each of the $d_{m'}$ left-hand members must be distinct, at least one of the right-hand members must not occupy the same location as the left-hand member of another pair (because N is prime) and for the remaining $d_{m'} - 1$ right-hand members the expected number of left-hand members that they occupy the same position as is given by the expected value of the hypergeometric distribution,

$$\frac{(d_{m'} - 1)^2}{N - d_{m'} - 2}$$

The expected number of distinct coefficients of value 1, $c_1(d_{m'}, N)$, is therefore

$$c_1(d_{m'}, N) = 2d_{m'} - \frac{(d_{m'} - 1)^2}{N - d_{m'} - 2}$$

Oscar will also have learned the location of some of the zero coefficients of F: each 1 coefficient that is not known to have another 1 i places to its left or to its right must have a zero in that position (as a 1 would have been detected, and the only other option is 0). This, too, will allow him to reduce the search space for F.

Now we estimate the amount of precomputation that Oscar must carry out in order to mount the attack.

First, we note that the running time of a standard combinatorial attack on an NTRUEncrypt private key is [4]

$$\tau(d_F, N) = \frac{1}{\sqrt{N}} \binom{\lceil N/2 \rceil}{\lceil d_F/2 \rceil}$$

If Oscar knows the locations of d_1 1s and d_0 0s in F, this running time becomes

$$\tau(d_F, N; d_0, d_1) = \binom{N - (d_0 + d_1 + \lfloor (d_F - d_1)/2 \rfloor)}{\lceil \frac{d_F - d_1}{2} \rceil} \tag{3}$$

(the top line here is about N, rather than about $N/2$, because rotational symmetry has been broken, and the factor of $1/\sqrt{N}$ vanishes for the same reason. Both of these changes hinder the attacker). Oscar's aim is to balance precomputation work and key-specific combinatorial work so as to recover a key in as little total effort as possible.

Oscar will start by selecting an integer δ and precomputing the time trails for all m' such that $d_{m'} \leq \delta$. We now estimate how many coefficients of F this

will enable him to recover. For any $(d_{m'}, i)$, we want to calculate the probability that F has exactly $d_{m'}$ pairs of coefficients separated by i, or in other words the probability that the dot product $(\mathsf{F} \cdot X^i \mathsf{F}) = d_{m'}$. To estimate this, consider what would happen if F and $X^i\mathsf{F}$ were independent. Each of the d_F 1 coefficients in F will select one of the coefficients in X^iF, giving a hypergeometric distribution of the values of $(\mathsf{F} \cdot X^i\mathsf{F})$. In practice, we know that if $i \neq 0$, a given 1 in F cannot select itself in X^iF and we observe that

$$P_{N,d_F}[\mathsf{F} \cdot X^i F = d_{m'}] = \mathrm{Hyp}(d_{m'}, d_F - 1, d_F, N) = \frac{\binom{d_F}{d_{m'}}\binom{N-d_F}{d_F-1-d_{m'}}}{\binom{N}{d_F-1}}$$

where $\mathrm{Hyp}(x, d, s, N)$, the hypergeometric distribution, is the probability of x successes in d draws without replacement from a pool containing N items of which s count as successes.

For any given value of $d_{m'}$, there are $(N-1)/2$ different values of i and $(N-1)/2$ distinct e_i. The expected number of m's with $d_{m'}$ 1s is therefore

$$E_1(d_{m'}) = \frac{(N-1)}{2} * \mathrm{Hyp}(d_{m'}, d_F - 1, d_F, N)$$

and the amount of precomputation work required to generate these time trails is

$$w_N(d_{m'}) = \frac{N(N-1)}{2}\binom{N}{d_{m'}}.$$

Every successful time trail identifies c_1 distinct 1 coefficients,

$$c_1(d_{m'}, N) = 2d_{m'} - \frac{(d_{m'}-1)^2}{N - d_{m'} - 2}.$$

It also identifies c_0 distinct 0s, one to the left of every lefthand 1 and one to the right of every righthand 1 except for the 1s that are lefthand in one pair and righthand in another:

$$c_1(d_{m'}, N) = 2d_{m'} - 2\frac{(d_{m'}-1)^2}{N - d_{m'} - 2}.$$

We now consider how quickly Oscar learns the distinct coefficients of F. Say that he knows d_0 0s and d_1 1s, and as a result of finding a time trail he discovers an additional c_0 0s and c_1 1s. Then we estimate the new expected total number of distinct known coefficients as

$$\begin{aligned}
\text{(new total)} &= \text{(already known)} + \text{(new)} - \text{(collisions between old and new)} \\
d_1' &= d_1 + c_1 - \mathrm{Exp}_x(\mathrm{Hyp}(x, c_1, d_1, d_F)) \\
&= d_1 + c_1 - \frac{d_1 c_1}{d_F} \\
d_0' &= d_0 + c_0 - \mathrm{Exp}_x(\mathrm{Hyp}(x, c_0, d_0, N - d_F)) \\
&= d_0 + c_0 - \frac{d_0 c_0}{N - d_F}.
\end{aligned}$$

This allows us to calculate the expected number of distinct coefficients found for a certain amount of precomputation corresponding to a certain value of δ, and therefore estimate the amount of work left to be done to recover the key. The method is:

1. Set $d_0 = d_1 = 0$. Set the total work $w = 0$.
2. For $d_{m'} = 1$ to δ:
3. Calculate $E_1(d_{m'})$.
4. If $\lfloor E_1(d_{m'}) \rceil \geq 1$:
   (a) Calculate $c_1(d_{m'}, N)$, $c_0(d_{m'}, N)$.
   (b) For $i = 1$ to $\lfloor E_1(d_{m'}) \rceil$:
   (c) Set $d_1 = d_1 + c_1 - \frac{d_1 c_1}{d_F}$.
   (d) Set $d_0 = d_0 + c_0 - \frac{d_0 c_0}{N - d_F}$
   (e) End i loop.
5. Set $w = w + w_N(d_{m'})$
6. End $d_{m'}$ loop.
7. Calculate $\tau(d_F, N; d_0, d_1)$ by (3) and output w, τ.

We emphasise that this is simply an estimate, and in particular the use of the hypergeometric distribution is an approximation to the actual distribution. The aim is simply to motivate a choice for δ.

4.1 Validating a Choice

The initial set of (bogus) ciphertexts $\mathcal{E} = \{\mathsf{e}_i = \lambda + \lambda X^i\}$ is relatively small to reduce precomputation. Recognizing a time trail will tell Oscar that with high probability he has identified $\mathsf{m}'(\mathsf{e}_i)$ for the relevant e_i in his database. However, if there is a nontrivial chance that the time trail is nonunique, Oscar may want to check that he has in fact identified the correct e_i.

To see how to do this, we note that if

$$\mathsf{e}_i = \lambda + \lambda X^i$$

decrypts to m', then so do the alternate forms

$$\mathsf{e}_i^* = (\lambda + 2) + \lambda X^i, \quad \text{or} \quad \lambda + (\lambda + 2)X^i, \quad \text{or} \quad \ldots$$

or indeed many polynomials $\lambda_1 + \lambda_2 X^i$ with λ_1 and λ_2 even integers in the vicinity of $q/4$ and satisfying $\lambda_1 + \lambda_2 > q/2$. Oscar therefore selects one of the possible e_i^*, calculates the time trail $T(\mathsf{m}', \mathsf{e}_i^*)$ for the message representative m' that he thinks is produced by decrypting the original e_i, and then submits e_i^* to the decryption oracle to find its time trail. If the measured and the calculated time trail match, he has confirmed the guess for m'. Otherwise, he knows that the original match on the time trail was just coincidence.

4.2 Results

We present the results of our analysis in Table 1. Here we have calculated two different values of δ.

The value δ_{onekey} is the value of δ that Oscar will precompute up to if he wants to recover one key, in other words the first value of δ for which the work required to peform the precomputation up to δ, $w(\delta)$, is greater than the remaining work required to break the key, τ. It can be seen that, with the exception of parameter set `ees251ep4` (presented in [7]), the log of the amount of precomputation to be performed $\log_2 w(\delta)$ is slightly more than half the claimed bit strength of the parameter sets. We also present, for interest, the number of distinct 1s and 0s that Oscar will on average have identified in a target key before he starts the combinatorial attack on the remaining coefficients.

The value δ_{allkeys} is the value of δ at which Oscar will on average recover the locations of all d_F value-1 coefficients in F through time trail analysis alone. This is the amount of precomputation that will allow him to recover any key at the cost of simply submitting about $N(N-1)$ ciphertexts for decryption. It can be seen that in general $w(\delta_{\text{allkeys}})$ is greater than $w(\delta_{\text{onekey}})$ by about 11 bits, or a factor of about 2000.

This demonstrates that, so long as the time trails are sufficiently unique and Oscar has an amount of storage that is customarily granted to attackers in this kind of paper, this attack is practical. In the next section we analyse the probability that time trails are unique.

Table 1. precomputation effort required to recover one key with minimum work and to recover all keys for the parameter sets in [7,8]

Bit Security	Parameter Set Name	N	d_r, d_F	$\delta_{\text{one key}}$	c_1	c_0	w	τ	$\delta_{\text{all keys}}$	w
80	`ees251ep4`	251	72	14	51.31	57.35	89.75	51.66	16	97.62
80	`ees251ep6`	251	48	5	40.26	64.72	47.86	23.99	7	58.36
112	`ees347ep3`	347	66	7	50.52	72.97	62.59	46.58	9	73.24
128	`ees397ep1`	397	74	8	60.81	94.48	69.95	42.74	10	80.67
160	`ees491ep1`	491	91	10	71.49	105.17	84.38	60.54	12	95.16
192	`ees587ep1`	587	108	12	79.57	110.07	98.79	88.09	14	109.62
256	`ees787ep1`	787	140	16	112.70	169.35	127.72	88.58	18	138.67

5 A Practical Hash Timing Attack for $\mathbf{f} = 1 + 2F$ — Practice

In this section we evaluate the practicality of the attack described in Section 4 for some specific NTRUEncrypt parameter sets that appear in [6,8] (and also the parameter set `ees251ep4` described in [7], which is secure but less efficient than the corresponding parameter sets in [8]). This practicality depends, among other

things, on the probability that different inputs require a greater or lesser number of SHA calls. We begin by describing how [8] uses SHA to compute r and then we compute the probability that this process takes a varying number of SHA calls.

The blinding value r, which is a binary polynomial with exactly d_r ones, is created from a hash function via repeated calls to some version of SHA. Here is the process as described in [8]:

1. Fix a value of c satisfying $2^c > N$. This value of c is specified in [8] for each of the sample NTRUEncrypt parameter sets. Also let
$$b = \lceil c/8 \rceil \qquad \text{and} \qquad n = \lfloor 2^c/N \rfloor$$
Thus b is the smallest integer such that b bytes contains at least c bits. (In practice, b will be 1 or 2.) Similarly, nN is the smallest multiple of N that is less than 2^c.
2. Call the specified version of SHA and break the output into chunks of b bytes each. Within each b byte chunk, keep the lower order c bits and discard the upper order $8b - c$ bits. Convert the lower order c bits into (little endian) integers $i_1, i_2, \ldots, i_t$. (Here t is the integer such that the output of the specified version of SHA consist of tb bytes.) This process of splitting the output from SHA is illustrated in Figure 1.
3. Create a list of indices $j_1, j_2, \ldots$ by looping through the list of i values from (2). If $i < n$ and i mod N is not already in the list, the adjoin i mod N to the list, otherwise discard i. Continue until the list contains d_r values of j. If at any point you run out of i values, then call SHA and create additional i values as specified in (2). The complete r generation algorithm is illustrated with pseudocode in Figure 2.

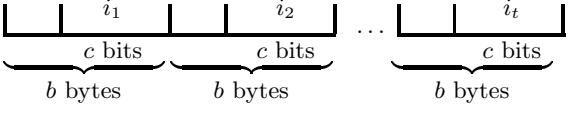

Fig. 1. Converting SHA output into c bit integers

```
(1)  jList = { }
(2)  Call SHA to get i_1, i_2, ..., i_t
(3)  Loop α = 1, 2, ..., t
(4)       If i_α < n and (i_α mod N) ∉ jList
              then adjoin i_α mod N to jList
(5)       If jList contains d_r elements, then exit
(6)  End α loop
(7)  Go to Step (2) to get more i values
```

Fig. 2. Generating r from SHA output

This description makes it clear why the number of calls to SHA may vary for different input values. If we treat the list of numbers $i_1, i_2, \ldots$ as a random sequence of integers in the range $0 \le i < 2^c$, the fundamental probabilities that we need to compute are

$$P_{C,N,n}(L,d) = \text{Prob}\left(\begin{array}{l}\text{A set of } L \text{ randomly chosen integers } i \in [0,C)\\ \text{includes exactly } d \text{ numbers satisfying both}\\ i \in [0,nN) \text{ and the values are distinct modulo } N\end{array}\right)$$

It is not hard to find a recursive formula that allows one to compute $P_{C,N,n}(L,d)$ reasonably quickly. See Appendix C for details.

In order to generate r, the algorithm described in Figure 2 needs to create a list of d_r distinct numbers satisfying $0 \le i < N$. Each time the algorithm calls SHA, it gets t numbers satisfying $0 \le i < 2^c$. Hence the probability that it suffices to call to SHA s times is equal to the probability that st random numbers in the range $[0, 2^c)$ contain at least d_r values in $[0, n)$ whose values modulo N are distinct. Hence

$$\text{Prob}(s \text{ calls to SHA suffices}) = \text{Prob}\left(\begin{array}{l}st \text{ randomly chosen integers in } [0,2^c)\\ \text{includes } \underline{\text{at least}}\ d_r \text{ values in } [0,n)\\ \text{that are distinct modulo } N\end{array}\right)$$

$$= \sum_{d_r \le d \le st} P_{2^c,N,n}(st,d).$$

In Table 2 we have assembled the NTRUEncrypt parameters from [7,8] and computed the values of s such that it is most likely to take either s or $s+1$ calls to SHA in order to generate r. The probabilities are listed in the last column of the table. The closer that the first probability is to 50%, the greater the chance that a time trail is unique, reducing the need to validate a time trail using the methods of Section 4.1. In most cases except perhaps $k = 80$ and $k = 192$, it will not be necessary to validate a time trail.

Table 2. The probability that s calls to SHA generates r

Bit Security	N	d_r	SHA bits	c	b	n	t	s : Prob(s SHA calls suffices)		$P_{\text{nonunique}}$
80	251	48	160	8	1	1	20	3 : 98.14%	4 : 100.0%	$2^{-13.5}$
112	347	66	160	14	2	47	10	7 : 15.65%	8 : 98.48%	2^{-154}
128	397	74	160	11	2	5	10	8 : 12.77%	9 : 95.10%	2^{-144}
160	491	91	160	9	2	1	10	10 : 13.87%	11 : 91.32%	2^{-193}
192	587	108	256	11	2	3	16	8 : 4.52%	9 : 82.38%	2^{-76}
256	787	140	256	12	2	5	16	10 : 53.04%	11 : 99.85%	2^{-783}

6 Practicality of Attack: Availability of Timing Information

As noted, it is possible for decryption to take a variable amount of time depending on the number of hash calls made. In this section we investigate how likely it is that this information will be leaked.

On a 1.7 GHz Pentium Pro running Windows XP, NTRUEncrypt decryption with the `ees251ep6` parameter set of [8] takes 0.09 ms. A SHA-1 call takes about 1.34μs. These are average figures. The time for these averages to settle down is obviously of interest.

We ran 100 sets of experiments, in each of which we decrypted a given `ees251ep6` ciphertext 1,000,000 times. As expected from Table 2, 98 of these ciphertexts took 3 SHA-1 calls to generate r and the other 2 took 4. We sorted the 100 experiments by running time and hoped to see that the 2 cases where there had been 4 SHA-1 calls would also have the longest running times. In fact, the noise due to other system activity overwhelms the variation in running time due to the number of hash calls on this system: the two cases where there had been 4 SHA-1 calls were in 29th and 68th position on the sorted list. Each of these runs took about 90 seconds. If the noise could be eliminated by bombarding the decryption oracle with the same ciphertext for a period of an hour, it would take the attacker $N(N-1)/2$ hours to recover all the time trails, or approximately $3\frac{1}{2}$ years. It therefore appears that this attack is unlikely to succeed against an implementation of NTRUEncryptdecryption running on a general computing platform.

At the other end of the computing scale, on an 8051-type smart card (a Philips Mifare ProX running at a 2.66 MHz internal clock, simulated on the Keil tools simulator) we observed that for `ees251ep4` the total time for a decryption was 58 ms, of which 30 ms was due to the 6 SHA-1 calls. In other words, on this platform, an additional SHA call incurs an overhead of 5 ms. It seems highly likely that in this environment the attack described in this paper is practical.

7 Conclusions and Recommendations

We have described a timing attack on the implementation of NTRUEncrypt described in [8]. The attack relies on the fact that decryption of different (possibly bogus) ciphertexts may require a different number of calls to a hash function such as SHA-1 or SHA-256. We draw some conclusions and make some recommendations.

1. The attack appears unlikely to work against NTRUEncryptrunning on a general-purpose PC platform. However, the parameter sets of [8] are claimed to be appropriate for any platform and as such it is worth investigating countermeasures that can be put in place on any platform.
2. Although we have only described an attack that relies on keys of the special form $\mathsf{f} = 1 + p\mathsf{F}$, it is reasonable to assume that similar attacks are possible for more general keys. Thus the use of general keys is not a recommended method to thwart hash timing attacks on NTRUEncrypt.
3. In order to prevent hash timing attacks, it suffices to make sure that almost all decryptions require the same number of SHA calls. This can be accomplished by fixing a parameter K_{SHA} so that almost all inputs $(\mathsf{m}', \mathsf{e})$ require at most K_{SHA} SHA calls and then performing extra SHA call(s) if necessary so that almost all inputs require exactly K_{SHA} SHA calls. Here, we can

put a more concrete meaning on "almost all" by requiring that at the k-bit security level, there is a chance of 2^{-k} that a given $(\mathsf{m}', \mathsf{e})$ has $\beta(\mathsf{m}', \mathsf{e}) = 1$. This yields the values given in Table 3 for K_{SHA}. Note that even with this number of SHA calls it is expected that decryption will take less than 0.5 s on the smartcard platform described above.

Table 3. Recommended number of SHA calls for different security levels

Bit Security	N	Expected SHA calls	K_{SHA}
80	251	3	6
112	347	8	15
128	397	9	17
160	491	11	22
192	587	9	20
256	787	10	21

Note that this recommendation will require an attacker to expend more than 2^k machine cycles to mount the attack, first because a SHA call takes more than one operation, and second because each attack involves $K_{\mathrm{SHA}} > 1$ SHA calls.

4. The method used to generate r from $(\mathsf{m}', \mathsf{e})$ in [8] is easy to implement, but it is somewhat wasteful of the pseudorandom bits produced by SHA. It might be worthwhile to look for more efficient ways to generate r which might also use a fixed number of calls to SHA, thereby eliminating the possibility of a hash timing attack. However, we note that the use of a new r-generation method would require changes to the exisiting standards, while equalization of the number of SHA calls as in (2) is a simple implementation change that maintains current standards.

   Finally, we note that NTRUEncryptshould continue to be analysed for its vulnerability to other side-channel attacks: this paper is by no means the last word on the subject.

References

1. D. Brumley, D. Boneh, Remote timing attacks are practical. Journal of Computer Networks, 2005.
2. J. Hoffstein, J. Pipher, J.H. Silverman, NTRU: A new high speed public key cryptosystem, 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
3. J. Hoffstein, J.H. Silverman, Optimizations for NTRU, Public Key Cryptography and Computational Number Theory (Warsaw, Sept. 11–15, 2000), Walter de Gruyter, Berlin–New York, 2001, 77–88.
4. 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`.

5. 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/`
6. N. Howgrave-Graham, J. H. Silverman, W. Whyte Choosing Parameter Sets for NTRUEncrypt with NAEP and SVES-3, Topics in cryptology—CT-RSA 2005, 118–135, Lecture Notes in Comput. Sci., 3376, Springer, Berlin, 2005. `www.ntru.com/cryptolab/articles.htm#2005_1`
7. Consortium for Efficient Embedded Security, *Efficient Embedded Security Standard (EESS) #1 version 2*, 2003.
8. Consortium for Efficient Embedded Security, *Efficient Embedded Security Standard (EESS) #1 version 3*, 2005.

A Probability That Two Message Representatives Have the Same Time Trail

A time trail is a binary vector of dimension N. We let P denote the probabilty that a randomly chosen coordinate is equal to 0, so $1 - P$ is the corresponding probablity that a randomly chosen coordinate is equal to 1. Then the probability that (say) the first coordinates of two random time trails agree is

$$\text{Prob(both 0)} + \text{Prob(both 1)} = P^2 + (1-P)^2 = 1 - 2P + 2P^2.$$

In order for two entire time trails to be identical, they must agree on all N of their coordinates. Hence

$$\text{Probability that two Time Trails coincide} = (1 - 2P + 2P^2)^N.$$

Therefore for any given $\mathsf{e} \in \mathcal{E}$ and $\mathsf{m}' \in \mathcal{M}$, the probability that there exists some other message representative $\mathsf{m}'' \in \mathcal{M}$ with $T(\mathsf{e}, \mathsf{m}'') = T(\mathsf{e}, \mathsf{m}')$ is approximately

$$\#\mathcal{M} \cdot (1 - 2P + 2P^2)^N.$$

B The Average Number of Ones with a Given Separation Distance

Let $\mathcal{B}_N(d)$ be the set of binary polynomials of degree less than N with exactly d ones and $N - d$ zeros. Fix i. We are interested in the average number of j such that F_j and F_{j-i} are both equal to 1, as F ranges over $\mathcal{B}_N(d)$. For a given F and i, we denote the number of such j by

$$\nu_i(\mathsf{F}) = \#\{0 \le j < N : F_j = F_{j-i} = 1\}.$$

Clearly $\nu_0(\mathsf{F}) = d$ for every $\mathsf{F} \in \mathcal{B}_N(d)$. We now fix some $1 \le i < N$ and compute the average value $\bar{\nu}_i(d)$ of $\nu_i(\mathsf{F})$ as F ranges over $\mathcal{B}_N(d)$.

$$\begin{aligned}
\bar{\nu}_i(d) = \operatorname*{Average}_{\mathsf{F}\in\mathcal{B}_N(d)} \quad \nu_i(\mathsf{F}) &= \binom{N}{d}^{-1} \sum_{\mathsf{F}\in\mathcal{B}_N(d)} \nu_i(\mathsf{F}) \\
&= \binom{N}{d}^{-1} \sum_{\mathsf{F}\in\mathcal{B}_N(d)} \sum_{j=0}^{N-1} F_j F_{j-i} \\
&= \binom{N}{d}^{-1} \sum_{j=0}^{N-1} \sum_{\mathsf{F}\in\mathcal{B}_N(d)} F_j F_{j-i} \\
&= \binom{N}{d}^{-1} \sum_{j=0}^{N-1} \#\big\{\mathsf{F}\in\mathcal{B}_N(d) : F_j = F_{j-i} = 1\big\} \\
&= \binom{N}{d}^{-1} \sum_{j=0}^{N-1} \binom{N-2}{d-2} \\
&= \binom{N}{d}^{-1} N\binom{N-2}{d-2} \\
&= \frac{d(d-1)}{N}.
\end{aligned}$$

This proves the formula cited in Section 4.

We also observe that $\nu_i(\mathsf{F})$ appears as a coefficient of the product $\mathsf{F} * \mathsf{F}^{\mathrm{rev}}$, where the reversal $\mathsf{F}^{\mathrm{rev}}$ of F is the polynomial $\mathsf{F}^{\mathrm{rev}} = \sum F_{-i}X^i$. Thus

$$\mathsf{F} * \mathsf{F}^{\mathrm{rev}} = \sum_{j=0}^{N-1}\sum_{k=0}^{N-1} F_j F_{-k} X^{j+k} = \sum_{i=0}^{N-1}\sum_{j=0}^{N-1} F_j F_{j-i} X^i = \sum_{i=0}^{N-1} \nu_i(\mathsf{F}) X^i.$$

Thus knowledge of $\nu_i(\mathsf{F})$ for $0 \le i < N$ is equivalent to knowledge of the product $\mathsf{F} * \mathsf{F}^{\mathrm{rev}}$. Using this value and the public key $\mathsf{h} = \mathsf{f}^{-1} * \mathsf{g} \bmod q$, there are practical methods for recovering F. In any case, it is certainly true that each valid $(\mathsf{r}, \mathsf{m}')$ pair that Oscar finds contains significant information about the private key f, and there are numerous ways to exploit such information in order to recover f directly (if one has enough $(\mathsf{r}, \mathsf{m}')$ pairs) or by cutting down the search space for f.

C The Probability of Choosing Distinct Values in a Given Range

In this section we describe a recursion that can be used to compute the probability

$$P_{C,N,n}(L,d) = \mathrm{Prob}\left(\begin{array}{l}\text{A set of } L \text{ randomly chosen integers } i \in [0,C) \\ \text{includes exactly } d \text{ numbers satisfying} \\ i \in [0,nN) \text{ and whose values are distinct modulo } N\end{array}\right)$$

We obtain a recursion from the observation that $P_{C,N,n}(L,d)$ equals the sum of the following two quantities:

- The probability after $L-1$ picks of having $d-1$ values in $[0,nN)$ that are distinct modulo N multiplied by the probability of picking an integer in $[0,nN)$ multiplied by the probability that it does not a repeat a previous values modulo N.
- The probability after $L-1$ picks of having d values in $[0,nN)$ that are distinct modulo N multiplied by the probability of picking an integer that either is not in $[0,nN)$ or whose value modulo N repeats a previous value.

We observe that for the first case, the probability of picking an integer in $[0,nN)$ multiplied by the probability that it does not a repeat a previous values modulo N is

$$\frac{nN}{C}\cdot\frac{N-(d-1)}{N}=\frac{n(N-d+1)}{C}.$$

For the second case, there are $C-nN$ integers in $[0,C)$ that are not in $[0,nN)$, and there are nd integers in $[0,nN)$ that are in one of the d congruence classes modulo n that have already been selected, so the probability of picking an integer that either is not in $[0,nN)$ or whose value modulo N repeats a previous value is

$$\frac{C-nN+nd}{C}=1-\frac{n(N-d)}{C}.$$

This yields the recursion formula

$$\begin{aligned}P_{C,N,n}(L,d)=P_{C,N,n}(L-1,d-1)\cdot\left(\frac{n(N-d+1)}{C}\right)\\ +P_{C,N,n}(L-1,d)\cdot\left(1-\frac{n(N-d)}{C}\right)\end{aligned}$$

Combining this recursion with the obvious initial values

$$P_{C,N,n}(L,d)=0\quad\text{if } L<d\qquad\text{and}\qquad P_{C,N,n}(L,0)=\left(1-\frac{nN}{C}\right)^{L},$$

it is an easy matter to compute $P_{C,N,n}(L,d)$ if the parameters are not too large.