Imperfect Decryption and an Attack on the NTRU Encryption Scheme

John A. Proos

University of Waterloo,
Waterloo, Canada
japroos@math.uwaterloo.ca

January 7, 2003

Abstract

A property of the NTRU public-key cryptosystem is that it does not provide perfect decryption. That is, given an instance of the cryptosystem, there exist ciphertexts which can be validly created using the public key but which can't be decrypted using the private key. The valid ciphertexts which an NTRU secret key will not correctly decipher determine, up to a cyclic shift, the secret key. In this paper we present attacks based on this property against the NTRU primitive and many of the suggested NTRU padding schemes [15, 10, 11]. These attacks use an oracle for determining if valid ciphertexts can be correctly deciphered, and recover the user's secret key. The attacks are quite practical. For example, the attack against the NTRU-REACT padding scheme proposed in [15] with the $N = 503$ parameter set [21] requires on average fewer than 30,000 oracle calls and can be performed on a PC in a few minutes. As the traditional definition of a public-key encryption scheme requires perfect decryption, we also define a new type of encryption scheme which encompasses both NTRU and an attack model for the attacks presented against it.

1 Introduction

The NTRU cryptosystem [9] is based on polynomial algebra modulo two distinct moduli. The problem of recovering the secret key from the public key has a very natural formation as a lattice basis reduction problem (see [5]). The security parameters suggested for the cryptosystem by NTRU

Cryptosystems Inc. are designed to make the related lattice basis reduction problem intractable (see [21]).

In [12] Jaulmes and Joux presented a chosen ciphertext attack against the NTRU cryptosystem. The basis of the attack was to construct invalid ciphertexts and ask for their decryptions. Knowledge of these decryptions could then be used to recover information about the user's secret key. The attack could be launched against the NTRU primitive as well as the OAEP-based padded version of NTRU found in [19]. Since the publication of this attack there have been many new padding schemes introduced [15, 10, 11] to protect the NTRU cryptosystem against invalid ciphertext attacks. All but one of the new padding schemes has the property that the decrypted value gives sufficient information to recalculate the unique valid ciphertext for the padded message. The one padding scheme without this property can be broken by an attack similar to the Jaulmes and Joux attack against the OAEP-based NTRU padding scheme (see section 5.3). When the valid ciphertext can be computed from the decryption value, attacks based on invalid ciphertexts can then be thwarted by altering the decryption process to compute the valid ciphertext from the recovered plaintext and return 'invalid' whenever the received ciphertext does not match the valid ciphertext. The modified decryption algorithm will return 'invalid' whenever it is given an invalid ciphertext. Thus no information about the secret key can be revealed through ciphertexts which the attacker already knows are invalid.

The NTRU cryptosystem has the interesting property that it does not provide perfect decryption. That is, given a key pair (pk, sk) there exist messages m for which some of the ciphertexts created using pk and m will not correctly decrypt to m. When the ability to correctly decipher valid ciphertexts is dependent on the secret key, the act of decrypting can leak information about the secret key. This information can be leaked both when messages are incorrectly and correctly decrypted. Thus imperfect decryption can be a very dangerous property for a cryptosystem to have. The NTRU cryptosystem has the undesirable property that the set of ciphertexts which do not correctly decrypt determine, up to a cyclic shift, the secret key.

The fact that the NTRU cryptosystem has imperfect decryption means that it does not satisfy the traditional definition of a public-key encryption scheme (see section 2.1). In this paper we present a new attack model for schemes with imperfect decryption and present attacks within this new model against the NTRU primitive and many of the suggested NTRU padding schemes. All of these attacks use indecipherable valid ciphertexts to recover the user's secret key. In the new attack model the attacker is given access to an oracle for determining when a validly created ciphertext correctly decrypts. Suppose an NTRU padding scheme which is secure against invalid ciphertext attacks receives a validly created ciphertext c of m which incorrectly decrypts to m'. The decryption algorithm, as modified above to resist the Jaulmes-Joux attacks, will return 'invalid' since the re-encryption of m' will not equal c. Thus decryption will return 'invalid' for a valid ciphertext exactly when the ciphertext can not be correctly deciphered. This implies that if the attacker can determine whether a user decrypts a ciphertext to a message or 'invalid' then the attacker can use the user as the oracle.

The new attack is quite practical and realistic. For example, the attack against the NTRU-REACT padding scheme proposed in [15] with the $N = 503$ parameter set [21] requires on average fewer than 30,000 oracle calls and can be performed on a PC in a few minutes. The attacks on some of the other padding scheme and parameter set combinations require more than 30,000 oracle calls. The success rates of the attacks depend on the number of oracle calls they are allowed to perform, but all the attacks have been successful with fewer than 250,000 oracle calls.

The rest of this paper is organized as follows. Section 2 recalls the definition of and security notions for public-key encryption schemes, and provides definitions for the new attack model for encryption schemes without perfect decryption. Section 3 reviews the REACT transformation [16] and analyzes its security for schemes without perfect decryption. Sections 4 and 5 review the NTRU cryptosystem and its suggested padding schemes. Section 6 presents the new attacks and some implementation results.

2 Encryption Schemes and Security

2.1 Public-Key Encryption Schemes

The standard definition of a *public-key encryption scheme* (PKE) [1, 3, 8] is a triple of algorithms, $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$, where

- $\mathcal{K}$, the *key generation algorithm*, is a probabilistic algorithm which takes as input a security parameter $k \in \mathbb{N}$ and returns a pair (pk, sk) of matching public and secret keys.

- $\mathcal{E}$, the *encryption algorithm*, is a probabilistic algorithm that takes as input a public key pk and a message $x \in \mathcal{M} \subseteq \{0,1\}^*$ and returns a ciphertext $y \in \mathcal{C} \subseteq \{0,1\}^*$, denoted $\mathcal{E}_{pk}(x) = y$. $\mathcal{M}$ is referred to as the message space and $\mathcal{C}$ is referred to as the ciphertext space. We will on occasion denote encryption as a deterministic algorithm, $\mathcal{E}_{pk}(x; r) = y$, where r is a bit string recording the coin tosses used by $\mathcal{E}_{pk}$ to produce y on input x. The random bit string r used in an encryption will be referred to as a nonce.

- $\mathcal{D}$, the *decryption algorithm*, is a deterministic algorithm that takes as input a secret key sk and a ciphertext y and returns either a message $x \in \mathcal{M}$ or a special symbol $\perp$ to indicate that the ciphertext was invalid. This is denoted $\mathcal{D}_{sk}(y) = x, \perp$.

It is further required that for all (pk, sk) which can be output from $\mathcal{K}$ and all $x \in \mathcal{M}$ that if $\mathcal{E}_{pk}(x; r) = y$ for any r then $\mathcal{D}_{sk}(y) = x$. That is the decryption algorithm will correctly decrypt any validly encrypted message. Given a key pair (pk, sk) a ciphertext y is called *valid* if there exists a message x and a random string r such that $\mathcal{E}(x; r) = y$, otherwise y is called *invalid*.

There are some security results [16, 7] for which the definition of a PKE also requires the decryption algorithm to have the property that it returns $\perp$ for all invalid ciphertexts. The encryption schemes which meet this more restrictive definition are simply a subset of those meeting the above definition and will be referred to as *restricted PKEs*. The results for the REACT transformation [16] are actually stated with no reference as to how a decryption algorithm should behave on invalid ciphertexts, but in fact only hold for the more restrictive definition of a PKE (see section 3).

2.2 Imperfect Public-Key Encryption Schemes

As mentioned above, the NTRU cryptosystem, $\Pi^N = (\mathcal{K}^N, \mathcal{E}^N, \mathcal{D}^N)$, does not provide perfect decryption since there exist x and r combinations for which $\mathcal{E}^N_{pk}(x; r)$ produces a ciphertext y such that $\mathcal{D}^N_{sk}(y) \neq x$. Thus the NTRU encryption scheme does not meet the definition of a public-key encryption scheme. This means that the security results proved for PKEs can not be assumed to hold true for NTRU.

In order to analyze the security of systems, like NTRU, without perfect decryption we define an *imperfect public-key encryption scheme (IPKE)* as a PKE without the requirement of perfect decryption (i.e. there can exist (pk, sk), x and r for which $\mathcal{D}_{sk}(\mathcal{E}_{pk}(x; r)) \neq x$). It would certainly be more natural to refer to IPKEs as public-key encryption schemes and the subset of these with perfect decryption (i.e. PKEs) as perfect public-key encryption schemes. However, we are restricted by the fact that there already exists a definition for PKEs.

Just as for PKEs, we can create a subset of all IPKEs by adding the requirement that invalid ciphertexts always decrypt to $\perp$. IPKEs which meet this requirement will be referred to as *restricted IPKEs*. From the definitions it is clear that every PKE is also a IPKE and that every restricted PKE is also a restricted IPKE. Note that the requirement of perfect decryption forces PKE encryption algorithms to be injective, however the encryption algorithm of an IPKE need not be injective.

We shall refer to a valid ciphertext $y = \mathcal{E}_{pk}(x; r)$ as *decipherable* or *indecipherable* with respect to x depending on whether or not $\mathcal{D}_{sk}(y) = x$. There are two points worth mentioning with regard to the definitions of validity and decipherability. The first is that the validity of a ciphertext depends only on its ability to be generated by $\mathcal{E}_{pk}$ and is independent of $\mathcal{D}_{sk}$. The second is that since encryption is no longer required to be injective there can exist ciphertexts y which can be validly created from two different messages. The decipherability of such ciphertexts can then be different with respect to the different messages. For example if $y = \mathcal{E}_{pk}(x; r) = \mathcal{E}_{pk}(x'; r')$, $\mathcal{D}_{sk}(y) = x$ and $x \neq x'$ then y is decipherable with respect to x but is indecipherable with respect to x'.

2.3 Security Notions

The three popular notions of security for encryption schemes are one-way (OW), indistinguishability (IND) and non-malleability (NM). An encryption scheme is said to be one-way if given a ciphertext $y = \mathcal{E}(x; r)$ an attacker can not recover the entire message x. Suppose an attacker chooses two messages m_1 and m_2 and someone encrypts one of these messages to form the ciphertext y. Indistinguishability (also known as semantic security) implies that given y the attacker can not guess which of m_1 and m_2 was encrypted with probability significantly better than one half. The last notion of security, non-malleability, roughly speaking requires that given a ciphertext y of a message x an attacker can not form a new ciphertext y' whose message is related to x in any meaningful way. There are two different definitions of non-malleability, but they were shown to be equivalent for PKEs (see [3]).

There are several types of attacks against encryption schemes which vary in the amount of information and abilities which are given to an attacker. For public-key encryption schemes the weakest type of attack is a *chosen plaintext attack* (CPA) which simply assumes that the attacker has access to the public key and the encryption function. Other types of attacks are created by giving the attacker access to various oracles. These attacks include:

- *Reaction attacks (RA)*: Here the attacker is given access to an oracle that given a ciphertext y will return whether or not y decrypts to invalid (i.e. returns whether $\mathcal{D}_{sk}(y) = \perp$).

- *Plaintext-checking attacks (PCA)*: Here the attacker is given access to an oracle that given a bit string y and a message x returns whether or not y is an encryption of x. (i.e. returns whether there exists an r such that $\mathcal{E}_{pk}(x; r) = y$)

- *Chosen-ciphertext attacks (CCA1 and CCA2)*: Here the attacker is given access to a decryption oracle that on input y returns $D_{sk}(y)$. There are two versions of these attacks, *non-adaptive chosen-ciphertext attacks* (CCA1) and *adaptive chosen-ciphertext attacks* (CCA2). For the non-adaptive version of the attack the attacker is only given access to the decryption oracle prior to being given the challenge ciphertext. For the adaptive version the attacker can query the decryption oracle even after receiving the challenge ciphertext, but for obvious reasons is not allowed to ask for the decryption of the challenge ciphertext.

If an encryption scheme is secure in the sense of SEC $\in$ {OW,IND,NM} against ATT $\in$ {RA,CPA,CCA1,CCA2} we will denote this by SEC-ATT (e.g. IND-CPA implies indistinguishability against chosen plaintext attacks). Note that the oracle for a CCA2 attack can be used to simulate the oracles of the other three attacks and thus if a scheme is secure against CCA2 attacks it is also secure against RA, CPA and CCA1 attacks. In [1] many relations among the various security levels were determined for PKEs, including that for PKEs IND-CCA2 is equivalent to NM-CCA2. Since the results are only proven for PKEs, care must be exercised when applying these results to IPKEs without perfect decryption. Section 3.1 gives an example of a result which holds for PKEs but does not hold for IPKEs.

2.4 Decipherable Ciphertext Attacks

When an encryption scheme has perfect decryption the act of correctly decrypting a ciphertext which is known to be valid does not reveal any information about the secret key. However, when an encryption scheme has imperfect decryption an attacker may be able to determine information about the secret key from knowledge of whether or not a valid ciphertext decrypted correctly. It is important to note that information about the secret key can be leaked even when a valid ciphertext is correctly decrypted. We shall now define a new type of oracle and attack based on the imperfect decryption property of IPKEs.

Let $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be an IPKE. Given an instance of Π with key pair (pk, sk), a *decipherable ciphertext oracle*, $DC_{(pk,sk)}$, is an oracle which on input x, r and y such that $\mathcal{E}_{pk}(x; r) = y$ returns whether or not $\mathcal{D}_{sk}(y) = x$. That is, a DC oracle can be used to determine if a valid ciphertext encrypted using pk can be correctly decrypted using sk.

An attack which makes use of the public information and a DC oracle shall be referred to as a *decipherable ciphertext attack* (DCA). Because a DC oracle provides some information about decryption, the strength of a DCA attack is greater than a CPA attack. On the other hand, a decryption oracle can be used to simulate a DC oracle which implies that a DCA attack is weaker than a CCA2 attack. Note that for a PKE, every valid ciphertext is decipherable which means a DC oracle for a PKE provides no information. Thus, for PKEs, DCA attacks are equivalent to CPA attacks.

3 The REACT Transformation

In 2000 Okamoto and Pointcheval [16] presented their Rapid Enhanced-security Asymmetric Cryptosystem Transform (REACT). REACT uses two hash functions G and H which output k_1-bit and k_2-bit strings respectively and transforms a PKE $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ into a new PKE $\Pi^R = (\mathcal{K}^R, \mathcal{E}^R, \mathcal{D}^R)$ as follows:

- $\mathcal{K}^R = \mathcal{K}$,
- $\mathcal{E}^R_{pk}(m; S, R)$: To encrypt a k_1-bit message m select a random message S and nonce R for Π and calculate $c_1 = \mathcal{E}_{pk}(S; R)$. The ciphertext then consists of (c_1, c_2, c_3), where $c_2 = G(S) \oplus m$ and $c_3 = H(S, m, c_1, c_2)$.
- $\mathcal{D}^R_{sk}(c_1, c_2, c_3)$: To decrypt calculate $S' = \mathcal{D}_{sk}(c_1)$, $m' = G(S') \oplus c_2$ and $c_3' = H(S', m', c_1, c_2)$. If S' is a Π message (thus not $\perp$) and $c_3' = c_3$ then output m', otherwise output $\perp$.

It was shown by Okamoto and Pointcheval [16] that in the random oracle model if Π is OW-PCA then Π^R will be IND-CCA2. However the proof only holds for restricted PKEs since its validity requires that $\mathcal{D}(y) =\perp$ for all invalid ciphertexts y. It turns out that when REACT is applied to a OW-PCA PKE which does not return $\perp$ for all invalid ciphertexts then the resulting PKE may not be IND-CCA2. To see this, consider any OW-PCA PKE $\Pi' = (\mathcal{K}', \mathcal{E}', \mathcal{D}')$ and let k be the bit length of the secret keys generated by $\mathcal{K}'$. Select a valid Π' message x and k invalid Π' ciphertexts $y_1, y_2, \ldots, y_k$. Form the PKE $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ on the same message space as Π' as follows.

- $\mathcal{K} = \mathcal{K}'$
- $\mathcal{E}_{pk}(m; r) = \mathcal{E}'_{pk}(m; r)$
- $\mathcal{D}_{sk}(y) = \begin{cases} \mathcal{D}'_{sk}(y) & \text{if } y \notin \{y_1, y_2, \ldots, y_k\} \\ x & \text{if } y = y_i \text{ and the i-th bit of sk is 1} \\ \perp & \text{otherwise} \end{cases}$

Π and Π' differ only in the decryption of a few invalid ciphertexts and are thus indistinguishable to a PCA attacker. Since Π' is OW-PCA this implies that Π is also OW-PCA.

Suppose REACT is applied to Π to produce Π^R. The following reaction attack could then be applied to Π^R to recover the secret key. For $1 \leq i \leq k$ form a Π^R ciphertext y_i^R with $S = x$ and c_1 replaced by y_i. The y_i^R can then be sent to the Π^R reaction oracle with the knowledge that y_i^R will decrypt to $\perp$ if and only if the ith bit of sk is zero. Thus Π^R is neither OW-RA nor IND-CCA2.

Alternatively the security results for REACT can be shown to hold for all PKEs when the PCA attack oracle is replaced by an oracle which given y and a message m returns whether or not $\mathcal{D}_{sk}(y) = m$ (instead of returning whether or not there exists an r such that $\mathcal{E}_{pk}(m; r) = y$).

3.1 REACT on IPKEs

In [15] REACT is applied to the NTRU cryptosystem and the claim is made that the system produced is IND-CCA2 provided that the NTRU primitive is OW-PCA. However, as theorem 1 shows, applying REACT to a OW-PCA restricted IPKE may not produce a IND-CCA2 IPKE.

Theorem 1 *Suppose there exists a OW-PCA restricted PKE. Let* $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ *be a OW-PCA restricted IPKE. Let* $\Pi^R = (\mathcal{K}^R, \mathcal{E}^R, \mathcal{D}^R)$ *be the IPKE produced by applying REACT to* Π*. Then* Π^R *may not be IND-CCA2. In fact,* Π^R *may not even be OW-DCA.*

Proof:
Given a binary string z denote the set of indices for which $z_i = 1$ by $I_1(z)$. Let $\Pi' = (\mathcal{K}', \mathcal{E}', \mathcal{D}')$ be any OW-PCA restricted PKE. Let $\mathcal{M}$ be the message space of Π' and let k be the bit size of the secret keys generated by $\mathcal{K}'$. Form the restricted IPKE $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ on $\mathcal{M}$ as follows:

- $\mathcal{K} = \mathcal{K}'$,

- $\mathcal{E}$: To encrypt a message $m \in \mathcal{M}$ first select a Π' nonce r and calculate $y' = \mathcal{E}'_{pk}(m; r)$ then select a k-bit integer s and set $\mathcal{E}_{pk}(m; r||s) = y'||s$.
- $\mathcal{D}$: If y has bit length less than k define $\mathcal{D}_{sk}(y) = \perp$; otherwise set s to be the last k bits of y, set y' such that $y = y'||s$ and define

$$\mathcal{D}_{sk}(y = y'||s) = \begin{cases} m & \text{if } \mathcal{D}'_{sk}(y') = m \text{ and } I_1(s) \subseteq I_1(sk) \\ \perp & \text{otherwise} \end{cases}$$

The valid Π ciphertexts are just the valid Π' ciphertexts with k random bits concatenated to the end. Thus it follows from the OW-PCA of Π' that Π is also OW-PCA. Furthermore, since $\mathcal{D}'$ returns $\perp$ for all invalid Π' ciphertexts, $\mathcal{D}$ will return $\perp$ for all invalid Π ciphertexts. Therefore Π is a OW-PCA restricted IPKE. Note that a valid Π ciphertext will not decrypt correctly exactly when the random string concatenated to the end of the Π' ciphertext has a 1 in a position where the secret key does not.

Let $\Pi^R = (\mathcal{K}^R, \mathcal{E}^R, \mathcal{D}^R)$ be the IPKE formed by applying REACT to Π using the two hash functions G and H. Let k_1 and k_2 be the number of bits returned by G and H respectively.

A DCA attack which can be used on Π^R to recover the secret key is as follows:

1. Select random k_1-bit messages $m_1, \ldots m_k$ for Π^R, random messages $S_1, \ldots, S_k \in \mathcal{M}$ and random nonces $R_1, \ldots, R_k$ for Π'.
2. For $1 \leq i \leq k$ calculate $\mathcal{E}^R_{pk}(m_i; S_i, R_i||e_i) = (c_{1i}, c_{2i}, c_{3i})$, where e_i is the k-bit $\{0,1\}$ vector which is zero except for the i-th coordinate.
3. For $1 \leq i \leq k$ send (c_{1i}, c_{2i}, c_{3i}) to the Π^R DC oracle and note that $sk_i = 1$ if and only if (c_{1i}, c_{2i}, c_{3i}) is decipherable.
4. Output the secret key sk.

Thus Π^R is not OW-DCA and since a decryption oracle can be used to simulate a DC oracle, Π^R is also not IND-CCA2. □

As REACT does provide added security for restricted PKEs and since a DCA/PCA attack is equivalent to a PCA attacks for PKEs it would be an interesting question to determine if REACT applied to a OW-DCA/PCA restricted IPKE will produce a IND-CCA2 IPKE.

4 The NTRU Cryptosystem

The NTRU cryptosystem as presented in [9] depends on three integer parameters (N, p, q) and fours sets of integer polynomials of degree less than N, denoted $\mathcal{L}_f$, $\mathcal{L}_g$, $\mathcal{L}_r$ and $\mathcal{L}_m$. It is also possible to define the system with p being a polynomial as in [4, 10], however for this paper we will only consider the case where p is an integer. The parameters p and q are chosen such that $\gcd(p, q) = 1$ and q is considerably larger than p. All polynomials are in the ring

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

Polynomials in R will occasionally be taken modulo q or p which simply means reducing each coefficient modulo q or p. $\mathcal{L}_m$ is the message space and provided p is odd is defined as

$$\mathcal{L}_m = \left\{ m \in R : m \text{ has all coefficients in } [-\frac{1}{2}(p-1), \frac{1}{2}(p-1)] \right\}$$

Let

$$\mathcal{L}(d_1, d_2) = \left\{ z \in R : \begin{array}{l} z \text{ has } d_1 \text{ coefficients equal to } 1, \\ d_2 \text{ equal to } -1 \text{ and the rest } 0 \end{array} \right\}$$

then

$$\mathcal{L}_f = \mathcal{L}(d_f, d_f - 1), \qquad \mathcal{L}_g = \mathcal{L}(d_g, d_g), \qquad \mathcal{L}_r = \mathcal{L}(d_r, d_r)$$

where d_f, d_g and d_r are positive integers whose values depend on N, q and p (see table 1 below).

4.1 NTRU Key Generation

NTRU key pairs are generated by selecting two polynomials $g \in \mathcal{L}_g$ and $f \in \mathcal{L}_f$ such that there exist polynomials f_p^{-1} and f_q^{-1} which satisfy

$$f \cdot f_p^{-1} \equiv 1 \pmod{p} \qquad \text{and } f \cdot f_q^{-1} \equiv 1 \pmod{q}$$

The secret key consists of the polynomials f and f_p^{-1}, while the public key is the polynomial $h = f_q^{-1} \cdot g \pmod{q}$.

4.2 NTRU Encryption and Decryption

Suppose that a public key h and a message $m \in \mathcal{L}_m$ are given. The encryption algorithm, $\mathcal{E}^N$, then consists of selecting a random element $r \in \mathcal{L}_r$ and calculating $e = m + prh \pmod{q}$ (i.e. $\mathcal{E}_h^N(m; r) = m + prh \pmod{q}$).

Given a ciphertext $e = m + prh \pmod{q}$ the decryption algorithm $\mathcal{D}^N_{f,f_p^{-1}}$ calculates

$$\begin{aligned} a &\equiv ef \pmod{q} \\ &\equiv mf + prhf \pmod{q} \\ &\equiv mf + prg \pmod{q} \end{aligned}$$

The NTRU parameters are selected in such a way that for the vast majority of m and r all the coefficients of $mf + prg$ will have absolute value at most $q/2$. Thus for most m and r when the coefficients of a are selected to be the least residue modulo q (i.e. in $(-q/2, q/2]$) a is equal to $mf + prg$. When this is the case decryption can proceed as follows

$$\begin{aligned} af_p^{-1} \pmod{p} &\equiv (mf + prg)f_p^{-1} \pmod{p} \\ &\equiv (mf)f_p^{-1} \pmod{p} \\ &\equiv m \pmod{p} \end{aligned}$$

By the definition of the message space if $m \in \mathcal{L}_m$ then $m = m \pmod{p}$ and the decryption algorithm will have recovered the message m. Thus if $\mathcal{E}^N_h(m; r) = e$ then $\mathcal{D}^N_{f,f_p^{-1}}(e) = m$ exactly when $ef \pmod{q}$ reduced to the range $(-q/2, q/2]$ equals $mf + prg$. This implies that a valid ciphertext $e = m + prh \pmod{q}$ will be indecipherable if and only if at least one coefficient of $mf + prg$ has a value outside $(-q/2, q/2]$ which provides a nice classification of indecipherable messages.

4.3 Suggested Parameters

The following NTRU parameter values were suggested in [21]. It has been shown that the suggested $N = 107$ parameters can be broken by lattice attacks in a few hours [13, 14]. Thus the $N = 107$ parameters are no longer considered secure, however we include $N = 107$ in the interest of providing run time comparisons versus the other security levels.

N	p	q	$\mathcal{L}_f$	$\mathcal{L}_g$	$\mathcal{L}_r$
107	3	64	$\mathcal{L}(15, 14)$	$\mathcal{L}(12, 12)$	$\mathcal{L}(5, 5)$
167	3	128	$\mathcal{L}(61, 60)$	$\mathcal{L}(20, 20)$	$\mathcal{L}(18, 18)$
263	3	128	$\mathcal{L}(50, 49)$	$\mathcal{L}(24, 24)$	$\mathcal{L}(16, 16)$
503	3	256	$\mathcal{L}(216, 215)$	$\mathcal{L}(72, 72)$	$\mathcal{L}(55, 55)$

Table 1: Suggested Security Parameters

5 NTRU Padding Schemes

In [15] Nguyen and Pointcheval analyzed the security of the three NTRU padding schemes proposed by NTRU Cryptosystems Inc. [10, 11] and suggested three new NTRU padding schemes. During the analysis of the six padding schemes there was an implicit assumption that NTRU is a PKE, when NTRU is in fact only an IPKE. Thus the 'proven' security levels may not actually hold under the stated assumptions.

In the analysis of the three original NTRU padding schemes the first was shown to be semantically insecure (not IND-CPA) and the second and third were claimed to be IND-CCA2 secure in the random oracle model under 'rather unusual assumptions' (see [15]). In this paper we will not directly consider the first two original padding schemes since it is the third scheme and a modified version of the third scheme which are being considered for standardization [4].

Of the three new padding schemes proposed by Nguyen and Pointcheval, one is based on OAEP [2] and the other two are based on REACT [16]. All three of the new padding schemes were claimed to be IND-CCA2 in the random oracle model based on the NTRU primitive being OW-CPA.

Let *mlen* and *rlen* be the length of the bit strings needed to describe polynomials in $\mathcal{L}_m$ and $\mathcal{L}_r$ respectively. All of the NTRU padding schemes work by forming strings of length *mlen* and *rlen* and mapping them to polynomials $m \in \mathcal{L}_m$ and $r \in \mathcal{L}_r$. The mappings used are normally quite natural (see [4]) and will not be discussed here. We will occasionally refer to binary strings of length *mlen* and *rlen* as begin in $\mathcal{L}_m$ and $\mathcal{L}_r$ with the implicit understanding that we are actually dealing with the polynomials to which the binary strings map.

5.1 NTRU-PAD3

There are two versions of the third original NTRU padding scheme which are being proposed for standardization [4]. Both versions form an *mlen* bit binary string S which depends on the message M and a random string R. The two version differ only in how S is formed from M and R.

Once S is determined the message $m \in \mathcal{L}_m$ is formed as follows using two hash functions F and G which map $\{0,1\}^{mlen/2}$ to itself. Let $S = \overline{S}||\underline{S}$, where $\overline{S}$ and $\underline{S}$ are each $mlen/2$ bits in length. Calculate $m_1 = \overline{S} \oplus F(\underline{S})$ and $m_2 = \underline{S} \oplus G(m_1)$. The message $m \in \mathcal{L}_m$ is then formed using the string $m_1||m_2$. The nonce $r \in \mathcal{L}_r$ is formed using the string $H(M||R)$, where H is a third hash function.

For the first method of forming S each of M and R is split in half to form $\overline{M}, \underline{M}, \overline{R}$ and $\underline{R}$. The values of $\overline{S}$ and $\underline{S}$ are then set to $\overline{M}||\overline{R}$ and $\underline{M}||\underline{R}$ respectively. Because of the way that R is split the semantic security provided by this padding scheme is $2^{k_2/2}$, where R is a k_2-bit string [6].

The second method of forming S, which appears in [4], is designed to provide a semantic security level of 2^{k_2}. In this version of the padding scheme S is formed as $S = R||Mlen||M||0$, where $Mlen$ is the length of M and the string is padded with zeros to ensure that S has the desired length.

5.2 NTRU-OAEP

The padding scheme based on OAEP uses three hash functions

$$\begin{aligned} &H : \{0,1\}^{k_1+k_2=mlen} \rightarrow \{0,1\}^{rlen}, \\ &F : \{0,1\}^{k_1} \rightarrow \{0,1\}^{k_2} \quad \text{and} \\ &G : \{0,1\}^{k_2} \rightarrow \{0,1\}^{k_1} \end{aligned}$$

Encoding of a message $M \in \{0,1\}^{k_1}$ proceeds by first selecting a random $R \in \{0,1\}^{k_2}$ and then computing $s = M \oplus G(R)$, $t = R \oplus F(s)$ and $y = \mathcal{E}^N_{pk}(s||t; H(M||R))$. To decode the ciphertext y the user first uses $\mathcal{D}^N_{sk}$ to retrieve $s||t$ from which they can compute $R = t \oplus F(s)$ and $M = s \oplus G(R)$. The message is accepted if M and R are valid and when used correctly generate the ciphertext y.

5.3 NTRU-REACT

The first REACT based padding scheme for NTRU uses two hash functions

$$G : \{0,1\}^{mlen} \rightarrow \{0,1\}^{k_1} \qquad \text{and} \qquad H : \{0,1\}^* \rightarrow \{0,1\}^{k_2}$$

The encoding of a message $M \in \{0,1\}^{k_1}$ is performed by selecting random $S \in \{0,1\}^{mlen}$ and $R \in \{0,1\}^{rlen}$ and computing $a = \mathcal{E}^N_{pk}(S; R)$, $b = M \oplus G(S)$ and $c = H(a, b, M, S)$. The ciphertext is then (a, b, c) from which the user can compute $S' = \mathcal{D}^N_{sk}(a)$, $M' = b \oplus G(S')$ and $c' = H(a, b, M', S')$. The ciphertext is excepted if M' and S' are valid messages and $c' = c$.

During the security analysis of this padding scheme it was implicitly assumed that the NTRU primitive was a OW-PCA restricted PKE, which would allow the use of the REACT transformation. However NTRU is not a restricted PKE (or even a restricted IPKE) and an attack similar to the Jaulmes and Joux [12] attack on the plaintext aware NTRU padding scheme in [19] can be applied against the NTRU primitive to recover the secret key. The basic principal of the attack is to send invalid ciphertexts of the form $e = m + b \pmod q$, instead of $m + prh \pmod q$, for carefully selected polynomials b for which knowledge of whether or not $\mathcal{D}_{sk}(e) = m$ reveals information about the secret key. Furthermore the same attack can be applied to NTRU-REACT by replacing $a = \mathcal{E}^N_{pk}(S; R)$ with $a = S + b \pmod q$. The NTRU-REACT ciphertexts formed in this way will decrypt to $\perp$ exactly when $\mathcal{D}^N_{sk}(a) \neq S$. Thus NTRU-REACT is in fact not OW-RA. Note that this attack relies only on the fact that the NTRU primitive does not return $\perp$ for all invalid ciphertexts and not on the fact that there exist indecipherable ciphertexts. The reason that such an invalid ciphertext attack can be used against NTRU-REACT is that NTRU-REACT does not recover the $R \in \mathcal{L}_r$ used by the NTRU primitive. Thus an NTRU-REACT user does not check if the a contained in the ciphertext is in fact a valid NTRU ciphertext.

5.4 NTRU-REACT2

The second REACT based padding scheme is a modified version of the NTRU-REACT scheme. NTRU-REACT2, which was considered an improved version of NTRU-REACT, requires a symmetric encryption scheme (E, D) with k_1 bit keys and k_2 bit messages and uses two hash functions

$$G : \{0,1\}^{mlen} \to \{0,1\}^{k_1} \qquad \text{and} \qquad H : \{0,1\}^* \to \{0,1\}^{rlen}$$

To encode a message $M \in \{0,1\}^{k_2}$ one selects a random $S \in \{0,1\}^{mlen}$ and computes $K = G(S)$, $b = E_K(M)$, $R = H(S, b)$ and $a = \mathcal{E}^N_{pk}(S; R)$. The ciphertext is $a||b$ and can be decoded using the secret key by determining $S' = \mathcal{D}^N_{sk}(a)$, $K' = G(S')$ and $M' = D_{K'}(b)$. Again the ciphertext $a||b$ is only accepted if it can be corrected generated from S' and M'.

6 Decipherable Ciphertext Attacks on NTRU

Let $\Pi^N = (\mathcal{K}^N, \mathcal{E}^N, \mathcal{D}^N)$ be the NTRU primitive. Suppose that an attacker is given an NTRU public key h and has access to a DC oracle for an instance of the NTRU primitive with public key h and secret key (f, f_p^{-1}). In sections 6.1 and 6.2, we present two DCA attacks against the NTRU primitive which can recover either the user's secret key or an equivalent secret key. The first attack assumes the attacker has complete freedom over the choices of $m \in \mathcal{L}_m$ and $r \in \mathcal{L}_r$, while the second attack assumes that the attacker may select $m \in \mathcal{L}_m$ but that the $r \in \mathcal{L}_r$ must be selected at random. Both attacks begin by searching for a pair (m, r) which lead to an indecipherable ciphertext. The first attack then proceeds by fixing r and vary m to find f. The second attack fixes m and varies r to find g from which f can be found. In sections 6.3, 6.4, 6.5, and 6.6, we extend the attacks to the padding schemes described in section 5.

6.1 NTRU Primitive Attack 1: Fixed r

This attack consists of three main stages. In the first stage the attacker must find a pair $(m, r) \in \mathcal{L}_m \times \mathcal{L}_r$ for which $e = \mathcal{E}_h^N(m; r)$ is indecipherable. The second stage of the attack uses the (m, r) pair found in the first stage to find a pair $(\bar{m}, \bar{r}) \in \mathcal{L}_m \times \mathcal{L}_r$ which generate a ciphertext which is in some sense just indecipherable. The final stage of the attack uses $(\bar{m}, \bar{r})$ to recover the secret key. Note that for all parameters suggested in table 1 the value of p is always 3 and q is always a power of 2. We shall assume these properties of p and q in the presentation of our attack. We emphases, however, that the attack will also work when q is not a power of two and can be easily modified to work when the integer p is not 3.

Stage 1:
As mentioned above, the goal of stage 1 is to determine a pair $(m, r) \in \mathcal{L}_m \times \mathcal{L}_r$ which generate an indecipherable ciphertext. The most straight forward approach is to simply select random m and r and use the DC oracle to determine if the ciphertext they generate is indecipherable. See section 6.1.1 for some experimental results on the expected number of oracle calls which would be required to find (m, r) randomly for the parameters suggested in table 1.

Given an m and r the attacker can determine exactly the set, $I_{m,r}$, of (f, g) pairs for which $e = \mathcal{E}_h^N(m; r)$ will be indecipherable. Thus sending $e = \mathcal{E}_h^N(m; r)$ to the DC oracle is equivalent to checking whether or not (f, g) is in $I_{m,r}$. So instead of simply selecting m and r at random the attacker could use a systematic approach to try to maximize the size of $\bigcup I_{m,r}$ after each DC oracle call. Such a systematic approach would provide a more efficient method of finding an indecipherable (m, r) pair than a simple random search.

Stage 2:
Suppose that stage 1 of the attack has been completed and found $(m, r) \in \mathcal{L}_m \times \mathcal{L}_r$ for which $e = \mathcal{E}_h^N(m; r)$ is indecipherable. Then at least one coefficient of $mf + rpg$ is outside the interval $(-q/2, q/2]$. The attacker now pro-

ceeds to either determine that $\mathcal{E}_h^N(0;r)$ is indecipherable or finds a message $\bar{m} \in \mathcal{L}_m$ for which $\mathcal{E}_h^N(\bar{m};r)$ is indecipherable, the coefficients of $\bar{m}f + rpg$ are in $[-q/2, q/2+1]$ and such that if any nonzero bit of $\bar{m}$ was set to zero then $\mathcal{E}_h^N(\bar{m};r)$ would be decipherable.

To find $\bar{m}$ the attacker starts by setting $\bar{m} = m$. The attacker then selects a nonzero coefficient of $\bar{m}$ and uses the DC oracle to determine if $\mathcal{E}_h^N(\bar{m};r)$ would be decipherable if the selected nonzero coefficient of $\bar{m}$ was set to zero. If the ciphertext would still be indecipherable then the selected coefficient is set to zero. The attacker repeats this process until either $\bar{m} = 0$ or setting any of the remaining nonzero coefficients of $\bar{m}$ to 0 would cause $\mathcal{E}_h^N(\bar{m};r)$ to be decipherable.

For $p = 3$ the j-th coefficient of $\bar{z} = \bar{m}f + rpg$ is

$$\bar{z}_j = \sum_{i=0}^{N-1} \bar{m}_i f_{j-i} + 3 \sum_{i=0}^{N-1} r_i g_{j-i} \qquad (1)$$

where the subscript k indicates the coefficient of x^k and all subscripts are taken modulo N. Since all the coefficients of $\bar{m}$ and f are in $\{-1, 0, 1\}$ setting one coefficient of $\bar{m}$ to zero will alter each coefficient of z by either $-1, 0$ or 1. Thus if the above process terminates with $\bar{m} \neq 0$ then $(\bar{m}, r)$ must satisfy the required conditions. If $\bar{m} \neq 0$ then stage 2 is completed and outputs $\bar{m}$ and $\bar{r} = r$.

Indecipherable (m, r) which lead to indecipherable $(0, r)$ are extremely rare compared to those for which $(0, r)$ is decipherable. We would thus not expect to need to handle the case of an indecipherable $(0, r)$ in stage 2 very often. However, if an indecipherable $(0, r)$ is found then there are two obvious choices as to how to proceed. The first would be simply return to stage 1. The second would be to apply small modifications to r until $(0, r)$ is decipherable and then try random m until (m, r) is indecipherable. Here rpg would still have at least one coefficient near the decipherable boundary meaning only a few oracles calls should be required before finding the desired m. Our implementation of the attack never found any indecipherable $(0, r)$ and thus never had to handle such an occurrence.

Stage 3:

Suppose that stage 2 has now completed and found $(\bar{m}, \bar{r})$ such that $\mathcal{E}_h^N(\bar{m};\bar{r})$ is indecipherable, the coefficients of $z = \bar{m}f + \bar{r}pg$ are in $[-q/2, q/2+1]$ and such that if any nonzero bit of $\bar{m}$ is set to zero then $\mathcal{E}_h^N(\bar{m};\bar{r})$ would be decipherable. Since $\mathcal{E}_h^N(\bar{m};\bar{r})$ is indecipherable z has at least one coefficient in $\{-q/2, q/2+1\}$. Suppose for the moment that exactly one coefficient, j, of z is in $\{-q/2, q/2+1\}$ and the rest are in $[-q/2+2, q/2-1]$. Thus

$$z_j = \sum_{i=0}^{N-1} \bar{m}_i f_{j-i} + 3 \sum_{i=0}^{N-1} \bar{r}_i g_{j-i} \in \{-q/2, q/2+1\}$$

Further assume that $z_j = q/2+1$. Let $\bar{m}^{i-}$, $\bar{m}^{i0}$ and $\bar{m}^{i+}$ be $\bar{m}$ with its i-th coefficient, $\bar{m}_i$, set to $-1, 0$ and 1 respectively. The value of f_{j-i} (for the unknown j) can now be determined as follows:

1. If $\bar{m}_i = 1$: Since $\mathcal{E}^N(\bar{m}^{i0}; \bar{r})$ is decipherable $f_{j-i} = 1$.
2. If $\bar{m}_i = -1$: Since $\mathcal{E}^N(\bar{m}^{i0}; \bar{r})$ is decipherable $f_{j-i} = -1$.
3. If $\bar{m}_i = 0$: If $\mathcal{E}^N(\bar{m}^{i-}; \bar{r})$ is decipherable then $f_{j-i} = 1$, if $\mathcal{E}^N(\bar{m}^{i+}, \bar{r})$ is decipherable then $f_{j-i} = -1$, otherwise $f_{j-i} = 0$.

Recall that if h is the NTRU public key generated from f and g then h will also be the public key generated by $x^i f$ and $x^i g$ for any i and that $x^i f$ can be used equally well as a secret key. Thus wlog we may assume that $j = 0$ and we will have determined a valid secret key f from which we can determine f_p^{-1}.

For the above procedure we assumed that $z_j = q/2+1$. If $z_j = -q/2$ then the f found by the above procedure will be -1 times an actual valid secret key f. Thus by proceeding as if $z_j = q/2+1$ and then checking which of f and $-f$ has the correct number of ones we can always find a valid secret key pair.

There are two other assumptions made in the above procedure for finding f which need to be addressed. One of these assumptions was that exactly one of the coefficients of z was in $\{-q/2, q/2+1\}$ and the other was that the rest of the coefficients of z were in $[-q/2+2, q/2-1]$. The only problem which would be caused if either of these assumptions failed would be that when $\bar{m}_i = 0$ the ciphertexts $\mathcal{E}^N(\bar{m}^{-i}; \bar{r})$ and $\mathcal{E}^N(\bar{m}^{+i}, \bar{r})$ could be indecipherable not because z_j is still out of range, but because some other coefficient z_k is out of range. This would result in incorrectly assigning $f_{j-i} = 0$. The odds of these assumptions not holding is quite small. Furthermore, since we know the number of 1's and -1's in f if only a few bits of f are incorrectly assigned zero then f could be recovered by exhaustively trying all possibilities. If such errors ever prevented f from being recovered the attacker could either apply small modifications to $\bar{m}$ and $\bar{r}$ and return to stage 2 or simply return to stage 1.

In summary, stage 3 is guaranteed to recover a secret key if $z = \bar{m}f + p\bar{r}g$ has exactly one coefficient in $\{-q/2, q/2+1\}$ and the rest in $[-q/2+$

$2, q/2 - 1]$. When z does not meet these requirements some information about f can be determined and there are techniques which may allow f to still be found; however, the attacker may be required to restart the attack. Heuristically one would expect a random indecipherable (m, r) to lead to an indecipherable pair $(\bar{m}, \bar{r})$ for which z has the desired properties and experimentally this is exactly what happened for every indecipherable (m, r) found.

6.1.1 Implementation Results for NTRU Attack 1

The majority of the work and DC oracle calls required for this attack are performed in stage 1 while finding the first indecipherable (m, r) pair. After the first indecipherable ciphertext is found the attack requires only around $3N$ additional calls to the DC oracle, many of which will return indecipherable. Table 2 shows the average number of DC oracle calls and the number of oracle calls returning 'indecipherable' when the attack was applied to one hundred instances of NTRU for each of the suggested parameters in table 1. The table also includes the ranges for the number of calls to the DC oracle which were required to find the first indecipherable ciphertext and complete stage 1. For each security level the attack was successful against all one hundred instances of NTRU.

N	DC Oracle Calls			Indecipherable Calls	
	Average	Std Dev	Stage 1 Range	Average	Std Dev
107	26241.32	26417.98	106 – 163844	222.35	5.70
167	20104.72	21213.11	62 – 116576	218.5	10.60
263	822979.1	778444.9	10632 – 4357309	505.6	11.67
503	27259.39	28640.06	173 – 165589	602.01	19.79

Table 2: NTRU Attack 1

Recall that there exists (m, r) pairs, which would terminate stage 1, but not lead to a successful completion of stages 2 and 3. As mentioned in the last section such (m, r) are unlikely and no such (m, r) pair was found during any runnings of the attack.

6.2 NTRU Primitive Attack 2: Fixed m

The second attack on the NTRU primitive assumes that the attacker has access to a DC oracle and that each time an (m, r) pair is generated for encryption the r must be selected at random from $\mathcal{L}_r$. The reason for requiring the r's to be selected at random will be shown later when the attack is modified to be applied to various padding schemes. The first stage of this attack is the same as for the first NTRU primitive attack and consists of searching for a pair $(m, r) \in \mathcal{L}_m \times \mathcal{L}_r$ which generates an indecipherable ciphertext. The second stage then fixes m and randomly searches for $\bar{r}$ for which $(m, \bar{r})$ is indecipherable. The attack then attempts to determine g and f by analyzing the distribution of 1's and -1's in the $\bar{r}$'s.

Stage 1:
Search for a pair $(m, r) \in \mathcal{L}_m \times \mathcal{L}_r$ which generates an indecipherable ciphertext. See stage 1 of NTRU attack 1 in section 6.1 for details.

Stage 2:
Assume that an indecipherable pair (m, r) was found in stage 1 of the attack and let $y = mf$. Suppose that y has one coefficient, j, which is closer to the boundary of decipherability ($q/2$ for positive coefficients and $-q/2 + 1$ for negative coefficients) than any other coefficient. Since cyclicly shifted secret keys $(x^i f, x^i g)$ work equally well for decryption we may assume wlog that $j = 0$.

Let $\bar{r} \in \mathcal{L}_r$ and consider the calculation of $(\bar{r}g)_0 = \sum_{i=0}^{N-1} \bar{r}_i g_{N-i}$. We shall say that $\bar{r}$ and g have a positive collision whenever $\bar{r}_i = g_{N-i} \neq 0$ and a negative collision whenever $\bar{r}_i = -g_{N-i} \neq 0$. The net collision between $\bar{r}$ and g will be taken to be the number of positive collisions minus the number of negative collisions. Thus $(\bar{r}g)_0$ equals the net collisions between $\bar{r}$ and g.

Select random $\bar{r} \in \mathcal{L}_r$ and use the DC oracle to determine if $(m, \bar{r})$ is decipherable. Note that y_0 is closer to the boundary of decipherability than the other coefficients of y and the expected contribution of $p\bar{r}g$ to each coefficient of $\bar{z} = y + p\bar{r}g$ is equal. Thus if $(m, \bar{r})$ is indecipherable it is more likely to be indecipherable in the 0-th coordinate than any other. Furthermore, when this is the case $z_0 = y_0 + p(rg)_0$ is outside the decipherable range and if y_0 is positive (resp. negative) this implies that the net collision between $\bar{r}$ and g is almost certainly positive (resp. negative).

For the moment suppose that y_0 is positive and that many $\bar{r}$ for which $(m, \bar{r})$ is indecipherable have been found. Since most of the $\bar{r}$'s will have a net positive collision with g by analyze the distribution of 1's and -1's in the $\bar{r}$'s we would expect to be able to recover the nonzero coefficients of g. In practise we see three distinct frequencies for when $\bar{r}_i = 1$: a high frequency when $g_{N-i} = 1$, a random frequency when $g_{N-i} = 0$ and low frequency when $g_{N-i} = -1$. Similarly we see three distinct frequencies for when $\bar{r}_i = -1$. Thus when enough $\bar{r}$ are found the attacker can determine all the coefficients of g from the frequencies of the nonzero coefficients in the $\bar{r}$'s. Since $fh \equiv g \pmod{q}$ once g has been determined f can be found by solving the linear system $Hf' = g'$ modulo q, where H is the $N \times N$ matrix whose rows are the cyclic shifts of the coefficients of h and f' and g' are the $N \times 1$ coefficient vectors of f and g.

To reduce the number of indecipherable $(m, \bar{r})$ pairs required to determine g the above technique can be combined with the lattice techniques for finding NTRU secret keys. This can be accomplished by using the frequencies of the 1's and -1's in the $\bar{r}$'s to guess some of the bits of g. The number of bits of g which can be guessed correctly will depend on the number of $(m, \bar{r})$ found and the rate at which they were found. Suppose that x bits of g have been determined. These bits yield x equations of the form $f_0 h_j + f_1 h_{j-1} + ... + f_{N-1} h_{j-N+1} \equiv g_j \pmod{q}$ which can be used to reduce the dimension of the NTRU lattice reduction problem from a $2N \times 2N$ lattice to a $2(N-x) \times 2(N-x)$ lattice (see [20, 14, 5] for details). Note that the lattice attack will determine both g and f.

If $y = mf$ has at least one large coefficient then the rate at which indecipherable $(m, \bar{r})$ are found will be noticeably larger than for random m and random r. Thus by analyzing the rate at which $\bar{r}$ are found we can determine if y has no large coefficients at which point the attack can simply return to stage 1. If $y = mf$ has two or more large coefficients which are equally close to the decipherable boundary then the frequencies of 1's and -1's will not have the correct form and the attack will need to return to stage 1. Even when the attack is forced to return to stage one some information about g may still be able to be determined. As shown below for randomly determined indecipherable (m, r) there is a good chance that $y = mf$ has the desired properties. Note that if y_0 is negative then the above techniques will find $-g$ and $-f$, but this can easily be corrected since the value of $f(1)$ is known to be one.

6.2.1 Implementation Results for NTRU Attack 2

The second attack on the NTRU primitive was run against one hundred instances of NTRU for each of the sets of suggested parameters. The attack tested to see if a 100×100 dimensional lattice could be used to find the secret key after every 25 $\bar{r}$ were found (a practical bound on the dimension of lattices which can be reduced is closer to 200×200). The attack successfully recovered the secret keys for every instance of every security level after only a few iterations of stages one and two.

Tables 3 and 4 contain some analysis of the implementation of the attack.

Table 3 shows the number of times stage 1 needed to be executed before the attack was successful, along with the maximum number of DC oracles calls which could be done in stage 2 before an attempt was deemed unsuccessful. Note that the number of oracle calls to complete stage 1 was the same as for attack 1 and that the number of iterations required will vary with the amount of oracle calls allowed in stage 2. Table 4 contains details on the successful stage 2's including information on the number of DC oracle calls required, the number of oracle calls returning 'indecipherable', the rate at which the indecipherable $(m, \bar{r})$ were found and the difference in the distances of the two coordinates of mf which were closest to the decipherable boundary.

N	Iterations for Success									Max Stage 2
	1	2	3	4	5	6	7	8	Rate	DC Calls
107	65	26	7	2	0	0	0	0	68.5%	500000
167	76	15	6	2	1	0	0	0	73.0%	500000
263	66	18	12	2	1	1	0	0	63.7%	3000000
503	47	24	18	7	2	1	1	0	50.0%	3000000

Table 3: NTRU Attack 2 Iterations

N	Total DC Calls		Indec DC Calls		Indec	mf Gap	
	Average	Std Dev (SD)	Avg	SD	Rate	Avg	SD
107	134809.17	126114.33	34.6	15.69	3896.2	4.6	2.25
167	110519.4	135748.33	62.45	21.55	1769.7	11.02	4.83
263	575803.99	533204.22	69.75	21.52	8255.3	13.18	4.75
503	565485.64	604270.83	323	133.18	1750.7	20.92	8.73

Table 4: Successful Stage 2 Details

6.3 An attack on NTRU-REACT

We will now show how the two DCA attacks against the NTRU primitive can be used as the basis for attacks against the NTRU padding schemes. The attacks rely on the fact that a validly created ciphertext for any of the padding schemes will decrypt to the original message exactly when the underlying NTRU primitive ciphertext is decipherable (otherwise it will decrypt to $\perp$). Thus a DC oracle for the underlying instance of the NTRU primitive can be simulated by using the oracle from either a DCA or RA attack against the padding scheme.

For NTRU-REACT the encoder has complete control over the values $S \in \mathcal{L}_m$ and $R \in \mathcal{L}_r$ and thus the first DCA attack against the NTRU primitive can be easily modified to apply to NTRU-REACT. The running time of this attack will be the same as the running time for the first DCA attack against the primitive.

6.4 An attack on NTRU-REACT2

During an encryption using NTRU-REACT2 the encoder has complete control over the elements $S \in \mathcal{L}_m$ but the $R \in \mathcal{L}_r$ are determined by using S, M, the hash functions and the symmetric encryption scheme. By keeping S fixed and varying M the attacker is in exactly the scenario of the second attack on the NTRU primitive. Thus the second attack on the NTRU primitive can be applied to NTRU-REACT2 with the same running time and success rate as it had against the NTRU primitive.

6.5 An attack on NTRU-OAEP

When encrypting using the NTRU-OAEP padding scheme the message used in the NTRU primitive is $s||t$ or more accurately some element of $\mathcal{L}_m$ which is derived from $s||t$. While neither attack on the NTRU primitive can be applied directly to NTRU-OAEP a modified version of the second attack can be used. The modification is required because in stage 2 of the attack if the message $s||t$ is held constant then NTRU-OAEP will always use the same nonce from $\mathcal{L}_r$. To overcome this problem we make minor changes to the message used in stage 2 with the hope that the majority of the indecipherable ciphertexts found will still be indecipherable because of the same coordinate.

The minor changes to the $s||t$ found in stage 1 are done by keeping s constant and perturbing t. Suppose R and M lead to $(s||t, H(M||R)) \in \mathcal{L}_m \times \mathcal{L}_r$. The attacker forms many $R_i = R \oplus w$ which are R with minor changes. Then for each R_i calculates $M = s \oplus G(R_i)$ and $t_i = R_i \oplus F(s) = t \oplus w$. Note that because of the hash functions the minor changes to R will cause the elements of $\mathcal{L}_r$ to change randomly.

The success of this attack will depend on the size of t and how $s||t$ is mapped to $m \in \mathcal{L}_m$. More specifically it will depend on how many coefficients of m change when minor changes are applied to t. For our implementation we have assumed that the alterations to t permit the top 30 coefficients of m to be altered as desired.

The NTRU-OAEP attack was run against one hundred instances of each of the sets of suggested NTRU parameters. Tables 5 and 6 contain some analysis of the implementation of the attack. The rate at which the lattice reductions were attempted and the dimension of the lattices used were the same as in the implementation of NTRU attack 2 (See section 6.2.1). In order to provide a good comparison of the success rate and work required for the NTRU-OAEP attack compared to NTRU attack 2 the secret keys and indecipherable ciphertexts used in the implementation of NTRU attack 2 were reused for the modified attack. Thus the NTRU-OAEP attack only generated new indecipherable ciphertexts when the ones used in the implementation of NTRU attack 2 no longer revealed a secret key. One can see from the tables that the NTRU-OAEP attack is only slightly less efficient than the NTRU attack 2.

N	Iterations for Success									Max Stage 2
	1	2	3	4	5	6	7	8	Rate	DC Calls
107	61	27	8	2	2	0	0	0	63.7%	500000
167	73	19	5	2	1	0	0	0	71.9%	500000
263	63	22	12	1	1	1	0	0	63.3%	3000000
503	44	27	18	5	3	2	1	0	48.5%	3000000

Table 5: NTRU-OAEP Attack Iterations

N	Total DC Calls		Indec DC Calls		Indec	mf Gap	
	Avg	SD	Avg	SD	Rate	Avg	SD
107	162972.1	156468.99	35.73	12.69	4561.2	4.62	2.40
167	133308.28	155890.09	66.67	30.99	1999.5	11.33	4.82
263	818353.84	853847.84	70.88	21.82	11545.6	13.1	4.79
503	591276.36	648049.95	319.08	120.18	1853.1	21.65	8.27

Table 6: Successful Stage 2 Details

It is worth noting that the technique used to modified NTRU attack 2 to work against NTRU-OAEP can also be used to produce attacks against the first and second original NTRU padding schemes. The running times of these attacks would be very similar to the running time of the NTRU-OAEP attack.

6.6 An attack on NTRU-PAD3

Like NTRU-OAEP neither of the attacks on the NTRU primitive can be applied directly to NTRU-PAD3, but once again a modified version of NTRU attack 2 can be used. Recall that when encrypting using NTRU-PAD3 the message used in the NTRU primitive is $m_1||m_2$, where $m_1 = \overline{S} \oplus F(\underline{S})$ and $m_2 = \underline{S} \oplus G(m_1)$. Suppose that in stage 1 of the attack it was found that $(m_1||m_2, H(M||R))$ generate an indecipherable message. Then instead of holding $m_1||m_2$ constant in stage 2 of the attack, m_1 will be held constant and small modifications will be applied to m_2. Because of the way the hash function H is used to generate the nonce r used in the NTRU primitive the small changes in m_2 will cause the r used to change randomly.

When $\overline{S} = \overline{M}||\overline{R}$ and $\underline{S} = \underline{M}||\underline{R}$ it is easy to hold the value of m_1 constant while applying small changes to m_2. Suppose we wish to modify m_2 to be $m_2 \oplus w$ while holding m_1 constant. Simply select $\underline{S'} = \underline{S} \oplus w$ and set $\overline{S'} = m_1 \oplus F(\underline{S'})$ then $m'_1 = \overline{S'} \oplus F(\underline{S'}) = m_1$ and

$$\begin{aligned} m'_2 &= \underline{S'} \oplus G(m'_1) \\ &= w \oplus \underline{S} \oplus G(m_1) \\ &= w \oplus m_2 \end{aligned}$$

When $S = b||Mlen||M||0$ the task of keeping m_1 constant while applying small changes to m_2 is slightly harder. The difficulty arrives from the fact that $Mlen$ will be part of $\overline{S}$ but its value is determined by the portion of M in $\underline{S}$. However, $Mlen$ is a short string (in [4] it is always 8 bits). Thus m_1 can be held constant while applying small modifications to m_2 by repeatedly attempting the above procedure until the required $Mlen$ corresponds with the desired value for $\overline{S'}$.

The attack on NTRU-PAD3 was run against one hundred instances of each of the sets of suggested NTRU parameters. The implementation assumed that the $m_1||m_2$ is mapped to $m \in \mathcal{L}_m$ in such a way that m_2 completely determines the coefficients of $x^{(N+1)/2}, \ldots, x^{N-1}$. Tables 7 and 8 contain some analysis of the implementation of the attack. Like the implementation of the NTRU-OAEP attack the NTRU-PAD3 attack was run in the same form as NTRU attack 2 and with the same secret keys and indecipherable messages. One can see from the tables that the NTRU-PAD3 attack has a success rate and running time approximately equal to that of the NTRU-OAEP attack.

7 Conclusions and Future Work

The current provable security results for PKEs assume that the encryption schemes in question provide perfect decryption. As such the results can't be assumed to hold for the more general class of IPKEs, which do

N	Iterations for Success									Max Stage 2
	1	2	3	4	5	6	7	8	Rate	DC Calls
107	65	25	6	1	2	0	1	0	64.9%	500000
167	73	17	7	2	1	0	0	0	70.9%	500000
263	67	22	9	0	1	0	0	0	67.6%	3000000
503	43	27	18	5	3	2	2	0	47.2%	3000000

Table 7: NTRU-PAD3 Attack Iterations

N	Total DC Calls		Indec DC Calls		Indec	mf Gap	
	Avg	SD	Avg	SD	Rate	Avg	SD
107	153274.2	159503.04	35.34	12.36	4337.1	4.7	2.38
167	113046.38	131427.51	66.93	24.58	1689.0	11.37	4.72
263	825059.36	850154.51	71.14	25.99	11597.7	12.89	4.72
503	509594.18	554497.27	310.01	99.90	1643.8	22.25	7.92

Table 8: Successful Stage 2 Details

not necessarily provide perfect decryption. As the REACT transformation demonstrates, some security proofs which hold for PKEs do not hold for all IPKEs. The DCA attacks against the NTRU primitive and its various suggested paddings schemes show that attacks based on indecipherable ciphertexts can be quite dangerous. Thus the security of cryptosystems without perfect decryption against DCA attacks must be considered before the system can be considered secure.

The DCA attacks presented in this paper show that finding even a single indecipherable NTRU ciphertext can lead to the recovery of the user's secret key. The attacks can also be launched against the various suggested NTRU padding schemes. This demonstates that it is very important to consider the security of NTRU padding schemes against DCA attacks. Perhaps the most effective way of protecting against DCA attacks is to select NTRU parameters in such a way that the problem of finding even a single indecipherable ciphertext is intractable.

In [10] many suggestions for the optimization of NTRU were suggested including changing p from 3 to $X+2$. The proposed NTRU standard [4] also contains a modification which causes the amount of work performed by the decryption algorithm to decrypt ciphertexts to be dependent on the secret key. The idea of this modification is to decrease the frequency of indecipherable ciphertexts by allowing the decryption algorithm to do additional work in an attempt to decipher what would otherwise have been indecipherable ciphertexts. The reduction of the frequency of indecipherable ciphertexts will have a direct affect on the success probability of DCA attacks. However, the variation in the amount of work performed by the decryption algorithm allows the possible use of timing and power analysis attacks to determine which ciphertexts require additional processing and such ciphertexts could be used in place of the indecipherable ciphertexts used in the $p = 3$ attacks. Attacks based on such ciphertexts have had some success against the parameters proposed in [4]. A follow up paper [17] will include an analysis of the security of the proposed modified versions of NTRU against DCA based attacks.

8 Acknowledgments

The author would like to thank Dan Brown for his many helpful discussions.

References

[1] Bellare, M., Desai, A., Pointcheval, D., and Rogaway, P. Relations Among Notions of Security for Public-Key Schemes. In *Advances in Cryptology — CRYPTO '98* (1998), vol. 1462 of *LNCS*, Springer-Verlag, pp. 26–46.

[2] Bellare, M., and Rogaway, P. Optimal Asymmetric Encryption. In *Advances in Cryptology — EUROCRYPT '94* (1995), vol. 950 of *LNCS*, Springer-Verlag, pp. 92–111.

[3] Bellare, M., and Sahai, A. Non-malleable Encryption: Equivalence between Two Notions, and an Indistinguishability-Based Characterization. In *Advances in Cryptology — CRYPTO '99* (1999), vol. 1666 of *LNCS*, Springer-Verlag, pp. 519–536.

[4] Consortium for Efficient Embedded Security. EESS #1: Implementation Aspects of NTRUEncrypt and NTRUSign. Version 1, available at www.ceesstandards.org, Nov 2002.

[5] Coppersmith, D., and Shamir, A. Lattice Attacks on NTRU. In *Advances in Cryptology — EUROCRYPT '97* (1997), vol. 1233 of *LNCS*, Springer-Verlag, pp. 52–61.

[6] Dai, W. P1363: NTRU attack with 2^{-40} advantage. Available at http://www.weidai.com/ntru-attack.tex, 2002.

[7] Fujisaki, E., and Okamoto, T. How to Enhance the Security of Public-Key Encryption at Minimum Cost. In *PKC '99* (1999), vol. 1560 of *LNCS*, Springer-Verlag, pp. 53–68.

[8] Goldwasser, S., and Micali, S. Probablilistic Encryption. *J. of Computer and System Sciences*, 28 (1984), 270–299.

[9] Hoffstein, J., Pipher, J., and Silverman, J. NTRU: A Ring-Based Public Key Cryptosystem . In *Proc of ANTS 3* (1998), vol. 1423 of *LNCS*, Springer-Verlag, pp. 267–288.

[10] Hoffstein, J., and Silverman, J. Optimizations for NTRU. In *Public-Key Cryptography and Computational Number Theory.* Warsaw, Sept. 11-15, 2000.

[11] Hoffstein, J., and Silverman, J. Protecting NTRU Against Chosen Ciphertext Attacks. Tech. Rep. 16, NTRU Cryptosystems, June 2000. version 1, available at www.ntru.com.

[12] Jaulmes, E., and Joux, A. A Chosen Ciphertext Attack on NTRU. In *Advances in Cryptology — CRYPTO '00* (2000), vol. 1880 of *LNCS*, Springer-Verlag, pp. 20–35.

[13] May, A. Cryptanalysis of NTRU-107. preprint, April 1999, (unpublished).

[14] May, A., and Silverman, J. Dimension Reduction Methods for Convolution Modular Lattices. In *Proc. of CaCL 2001* (2001), vol. 2146 of *LNCS*, Springer-Verlag, pp. 110–125.

[15] Nguyen, P., and Pointcheval, D. Analysis and improvements of NTRU encryption paddings. In *Advances in Cryptology — CRYPTO 2002* (2002), vol. 2442 of *LNCS*, Springer-Verlag, pp. 210–225.

[16] Okamoto, T., and Pointcheval, D. REACT: Rapid Enhanced-security Asymmetric Cryptosystem Transform. In *Proc. of CT-RSA '01* (2001), vol. 2020 of *LNCS*, Springer-Verlag, pp. 159–175.

[17] Proos, J. Attacks on the EESS Standard for the NTRU Encryption Scheme. In preparation.

[18] Shoup, V. Number Theory C++ Library (NTL). Available at www.shoup.net/ntl.

[19] Silverman, J. Plaintext Awareness and the NTRU PKCS. Tech. Rep. 7, NTRU Cryptosystems, July 1998. version 1,(version 2 available at www.ntru.com).

[20] Silverman, J. Dimension-Reduced Lattices, Zero-Forced Lattices and the NTRU Public Key Cryptosystem. Tech. Rep. 13, NTRU Cryptosystems, March 1999. version 1, available at www.ntru.com.

[21] Silverman, J. Estimated breaking times for NTRU lattices. Tech. Rep. 12, NTRU Cryptosystems, 1999. Available at www.ntru.com.