The Impact of Decryption Failures on the Security of NTRU Encryption

Nick Howgrave-Graham[1], Phong Q. Nguyen[2], David Pointcheval[2], John Proos[3], Joseph H. Silverman[1], Ari Singer[1], and William Whyte[1]

[1] NTRU Cryptosystems, 5 Burlington Woods, Burlington, MA 02144
{nhowgravegraham,jhs,asinger,wwhyte}@ntru.com
[2] CNRS/ENS–DI, 45 rue d'Ulm, 75005 Paris, France
{phong.nguyen,david.pointcheval}@ens.fr
[3] University of Waterloo, 200 University Ave. West, Waterloo, Canada N2L 3G1
japroos@math.uwaterloo.ca

Abstract. NTRUEncrypt is unusual among public-key cryptosystems in that, with standard parameters, validly generated ciphertexts can fail to decrypt. This affects the provable security properties of a cryptosystem, as it limits the ability to build a simulator in the random oracle model without knowledge of the private key. We demonstrate attacks which use decryption failures to recover the private key. Such attacks work for all standard parameter sets, and one of them applies to any padding. The appropriate countermeasure is to change the parameter sets and possibly the decryption process so that decryption failures are vanishingly unlikely, and to adopt a padding scheme that prevents an attacker from directly controlling any part of the input to the encryption primitive. We outline one such candidate padding scheme.

1 Introduction

An unusual property of the NTRU public-key cryptosystem is the presence of *decryption failures*: with standard parameters, validly generated ciphertexts may fail to decrypt. In this paper, we show the importance of decryption failures with respect to the security of the NTRU public-key cryptosystem. We believe this fact has been much overlooked in past research on NTRU.

First, we notice that decryption failures cannot be ignored, as they happen much more frequently than one would have expected. If one strictly follows the recommendations of the EESS standard [3], decryption failures happen as often as every 2^{12} messages with $N = 139$, and every 2^{25} messages with $N = 251$. It turns out that the probability is somewhat lower (around 2^{-40}) with NTRU products, as the key generation implemented in NTRU products surprisingly differs from the one recommended in [3]. In any case, decryption failures happen sufficiently often that one cannot dismiss them, even in NTRU products.

One may *a priori* think that the only drawback with decryption failures is that the receiver will not be able to decrypt. However, decryption failures have a big impact on the confidence we can have in the security level, because they limit the ability to build a simulator in the random oracle model without knowledge of the private key. This limitation is independent of the padding used. This implies that all the security proofs known (see [14, 21]) for various NTRU paddings may not be valid after all, because decryption failures have been ignored in such proofs. This also means that existing generic padding schemes (such as REACT [23]) may not apply to NTRU as, to our knowledge, no existing padding scheme takes into account the possibility of decryption failures, perhaps because the only competitive cryptosystem that experiences decryption failures is NTRUEncrypt.

From a security point of view, the situation is even worse. It turns out that decryption failures not only influence the validity of a security proof, they leak information on the private key. We demonstrate this fact by presenting new efficient chosen-ciphertext attacks on NTRUEncrypt (with or without padding) that recover the private key. These chosen-ciphertext attacks are very different from chosen-ciphertext attacks [17, 15] formerly known against NTRU: they only use valid ciphertexts, while the attacks of [17, 15] use fake ciphertexts and can therefore be easily thwarted. Moreover, these chosen-ciphertext attacks do not use the full power of chosen-ciphertext attacks, but reaction attacks only [9]: Here, the attacker selects various messages, encrypts them, and checks whether the receiver is able to correctly decrypt the ciphertexts: eventually, the attacker has gathered sufficiently many decryption failures to be able to recover the private key. The only way to avoid such chosen-ciphertext attacks is to make sure that it is computationally infeasible to find decryption failures. This requires different parameter sets and certain implementation changes in NTRUEncrypt, which we hint in the final section of this paper.

The rest of the paper is organized as follows. In Sections 2 and 3, we recall NTRUEncrypt and the padding used in EESS [4]. In Section 4, we explain decryption failures, and their impact on security proofs for NTRU paddings. In Section 5, we present several efficient attacks based on decryption failures: some are tailored to certain paddings, while the most powerful one applies to any padding. In Appendix A, we give additional information on NTRUEncrypt, pointing out the difference between the specification of the EESS standard and the implementation in NTRU products. In Appendix B, we describe an alternative attack based on decryption failures that work against certain NTRU paddings.

2 The NTRU Encryption Scheme

2.1 Definitions and Notation

NTRUEncrypt operations take place in the quotient ring of polynomials $\mathcal{P} = \mathbb{Z}[X]/(X^N - 1)$. In this ring, addition of two polynomials is defined as pairwise addition of the coefficients of the same degree, and multiplication is defined as convolution multiplication. The convolution product $\mathsf{h} = \mathsf{f} * \mathsf{g}$ of two polynomials f and g is given by taking the coefficient of X^k to equal $\mathsf{h}_k = \sum_{i+j \equiv k \bmod N} \mathsf{f}_i \cdot \mathsf{g}_j$. Several different measures of the size of a polynomial turn out to be useful. We define the *norm* of a polynomial f in the usual way, as the square root of the sum of the squares of its coefficients. We define the *width* of a polynomial f as the difference between its largest coefficient and its smallest coefficient.

The fundamental parameter in NTRUEncrypt is N, the ring dimension. The parameter N is taken to be prime to prevent attacks due to Gentry [6], and sufficiently large to prevent lattice attacks. We also use two other parameters, p and q, which are relatively prime. Standard practice is to take q to be the power of 2 between $N/2$ and N, and p to be either the integer 3 or the polynomial $2+X$. We thus denote p as a polynomial p in the following, and we focus on the case $\mathsf{p} = 2 + X$ as $p = 3$ is no longer recommended in NTRU standards [3, 4].

2.2 Overview

The basic NTRUEncrypt key generation, encryption, and decryption primitives are as follows.

Key Generation — Requires a source of (pseudo-)random bits, and subspaces $\mathcal{L}_\mathsf{f}, \mathcal{L}_\mathsf{g} \subseteq \mathcal{P}$ from which the polynomials f and g are drawn. These subspaces have the property that all polynomials in them have small width – for example, they are now commonly taken to be the space of *all binary polynomials* with d_f, d_g 1s respectively.

- Input: Values N, p, q.
  - Randomly choose $\mathsf{f} \in \mathcal{L}_\mathsf{f}$ and $\mathsf{g} \in \mathcal{L}_\mathsf{g}$ in such a way that both f and g are invertible mod$q$;
  - Set $\mathsf{h} = \mathsf{p} * \mathsf{g} * \mathsf{f}^{-1} \bmod q$.
- Public output: The input parameters and h.
- Private output: The public output, f, and $\mathsf{f_p} \equiv \mathsf{f}^{-1} \bmod \mathsf{p}$.

Encryption — Requires a source of (pseudo-)random bits, subspaces $\mathcal{L}_\mathsf{r}$ and $\mathcal{L}_\mathsf{m}$ from which the polynomials r and m are to be drawn, and an invertible means $\mathcal{M}$ of converting a binary message m to a message representative $\mathsf{m} \in \mathcal{L}_\mathsf{m}$. The subspaces $\mathcal{L}_\mathsf{r}, \mathcal{L}_\mathsf{m}$ also have the property that all polynomials in them have low width.

- Input: A message m and the public key h.
  - Convert m to the message representative $\mathsf{m} \in \mathcal{L}_\mathsf{m}$: $\mathsf{m} = \mathcal{M}(m)$;
  - Generate random $\mathsf{r} \in \mathcal{L}_\mathsf{r}$.
- Output: The ciphertext $\mathsf{e} = \mathsf{h} * \mathsf{r} + \mathsf{m} \bmod q$.

Decryption —

- Input: The ciphertext e, the private key f, $\mathsf{f_p}$ and the parameters p and q.
  - Calculate $\mathsf{a} = \mathsf{f} * \mathsf{e} \bmod q$. Here, "mod q" denotes reduction of a into the range $[A, A+q-1]$, where the "centering value" A is calculated by a method to be discussed later;
  - Calculate $\mathsf{m} = \mathsf{f_p} * \mathsf{a} \bmod \mathsf{p}$, where the reduction is to the range $[0, 1]$.
- Output: The plaintext m, which is m converted from a polynomial in $\mathcal{L}_\mathsf{m}$ to a message.

2.3 Decryption Failures

In calculating $\mathsf{a} = \mathsf{f} * \mathsf{e} \bmod q$, one actually calculates

$$\mathsf{a} = \mathsf{f} * \mathsf{e} = \mathsf{f} * (\mathsf{r} * \mathsf{h} + \mathsf{m}) = \mathsf{p} * \mathsf{r} * \mathsf{g} + \mathsf{f} * \mathsf{m} \bmod q. \tag{1}$$

The polynomials p, r, g, m, and f are chosen to be small in $\mathcal{P}$, and so the polynomial $\mathsf{p} * \mathsf{r} * \mathsf{g} + \mathsf{f} * \mathsf{m}$ will, with "very high probability", have width less than q. If this is the case, it is possible to reduce a into a range $[A, A + q - 1]$ such that the mod q equality in the Equation (1) is an exact equality in $\mathbb{Z}$. If this equality is exact, the second convolution gives

$$\mathsf{f_p} * \mathsf{a} = \mathsf{p} * (\mathsf{f_p} * \mathsf{r} * \mathsf{g}) + \mathsf{f_p} * \mathsf{f} * \mathsf{m} = 0 + 1 * \mathsf{m} = \mathsf{m} \bmod \mathsf{p}, \text{ recovering } \mathsf{m}.$$

Decryption only works if the equality $\bmod q$ in Equation (1) is also an equality in $\mathbb{Z}$. This condition will not hold if A has been incorrectly chosen so that some coefficients of $\mathsf{p} * \mathsf{r} * \mathsf{g} + \mathsf{f} * \mathsf{m}$ lie outside the centering range, or if $\mathsf{p} * \mathsf{r} * \mathsf{g} + \mathsf{f} * \mathsf{m}$ happens to have a width greater than q (so that there is no mod q range that makes the Equation (1) an exact equality). In this case, the recovered m will differ from the encrypted m by some multiple of q mod p. These events are the *decryption failures* at the core of this paper.

Before NTRUENCRYPT can be used, the subspaces $\mathcal{L}_\mathsf{f}, \mathcal{L}_\mathsf{g}, \mathcal{L}_\mathsf{r}$ must be specified. See appendix A for details of the precise form of the polynomials $\mathsf{f}, \mathsf{g}, \mathsf{r}$ used in the standard [3] and the (slightly) different one deployed in NTRU Cryptosystems' products.

2.4 The NTRU Assumption

Among all the assumptions introduced in [21], the most important one is the onewayness of the NTRU primitive, namely that the following problem is asymptotically hard to solve:

Definition 1 (The NTRU Inversion Problem). *For a given security parameter k, which specifies N, p, q and the spaces $\mathcal{L}_\mathsf{f}$, $\mathcal{L}_\mathsf{g}$, $\mathcal{L}_\mathsf{m}$, and $\mathcal{L}_\mathsf{r}$, as well as a random public key h and $\mathsf{e} = \mathsf{h} * \mathsf{r} + \mathsf{m}$, where $\mathsf{m} \in \mathcal{L}_\mathsf{m}$ and $\mathsf{r} \in \mathcal{L}_\mathsf{r}$, find m. We denote by $\mathsf{Succ}^{\mathsf{ow}}_{\mathsf{ntru}}(\mathcal{A})$ the success probability of any adversary $\mathcal{A}$.*

$$\mathsf{Succ}^{\mathsf{ow}}_{\mathsf{ntru}}(\mathcal{A}) = \Pr\left[\begin{array}{l}(\mathsf{h}, \star) \leftarrow \mathcal{K}(1^k), \mathsf{m} \in \mathsf{Messages}, \mathsf{r} \in \mathsf{Random}, \\ \mathsf{e} = \mathsf{h} * \mathsf{r} + \mathsf{m} \bmod q : \mathcal{A}(\mathsf{e}, \mathsf{h}) = \mathsf{m}\end{array}\right].$$

3 The NTRU Paddings

For clarity reasons, in the description of the paddings, we consider the encryption scheme $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$, which is an improvement of the plain NTRU cryptosystem that includes the two public encodings $\mathcal{M}$ and $\mathcal{R}$:

$$\left\{\begin{array}{rll} \mathcal{K}(1^k) = (\mathsf{pk} = \mathsf{h}, \mathsf{sk} = (\mathsf{f}, \mathsf{f_p})), & \text{with,} & \\ \mathcal{E}_\mathsf{pk}(m; r) = \mathcal{M}(m) + \mathcal{R}(r) * \mathsf{h} \bmod q, & \mathcal{M} : \mathsf{Messages} = \{0,1\}^{\mathsf{mLen}} & \to \mathcal{L}_\mathsf{m} \\ \mathcal{D}_\mathsf{sk}(\mathsf{e}) = \mathcal{M}^{-1}((\mathsf{e} * \mathsf{f} \bmod q)_A * \mathsf{f_p}) & \mathcal{R} : \mathsf{Random} = \{0,1\}^{\mathsf{rLen}} & \to \mathcal{L}_\mathsf{r} \end{array}\right.$$

Because of the encodings, without any assumption, recovering the bit-string m is as hard as recovering the polynomial $\mathsf{m} = \mathcal{M}(m)$. However, recovering ℓ bits of m does not necessarily provide ℓ bits of the polynomial $\mathsf{m} = \mathcal{M}(m)$, which is the reason why stronger assumptions were introduced in [21].

3.1 Description

Following the publication of [17], several padding schemes were proposed in [14, 13] to protect NTRUEncrypt against adaptive chosen-ciphertext attacks. The resulting schemes were studied in [21], but under the assumption that the probability of decryption failures was negligible. We briefly review one of these schemes, known as SVES-1 and standardized in [3].

Let m be the original plaintext represented by a k_1-bit string. For each encryption, one generates a random string b, whose bit-length k_2 is between 40 and 80 [14]. However, $k_1 + k_2 \leq \mathsf{mLen}$. Let $\|$ denote bit-string concatenation.

To encrypt, first split each of m and b into equal size pieces $m = \overline{m} \,\|\, \underline{m}$ and $b = \overline{r} \,\|\, \underline{r}$. Then, use two hash functions F and G that map $\{0,1\}^{k_1/2+k_2/2}$ into itself, to compute:

$$m_1 = (\overline{m} \,\|\, \overline{r}) \oplus F(\underline{m} \,\|\, \underline{r}) \text{ and } m_2 = (\underline{m} \,\|\, \underline{r}) \oplus G(m_1).$$

A third hash function, $H : \{0,1\}^{\mathsf{mLen}} \to \{0,1\}^{\mathsf{rLen}}$, is applied to yield the ciphertext:

$$\mathcal{E}^3_{\mathsf{pk}}(m; b) = \mathcal{E}_{\mathsf{pk}}(m_1 \,\|\, m_2; H(m \,\|\, b)).$$

The decryption algorithm consists of recovering m, b from the plain NTRU decryption, and re-encrypting to check whether one obtains the given ciphertext. This is equivalent to extracting, from m and e, the alleged r, and check whether it is equal to $\mathcal{R}(H(m \,\|\, b))$. We denote by Π^3 the corresponding encryption scheme.

3.2 Chosen-Ciphertext Attacks

First of all, one may note that because of the random polynomial r that is generated from $H(m \,\|\, b)$, nobody can generate a valid ciphertext without knowing both the plaintext m and the random b, except with negligible probability, for well-chosen conversion $\mathcal{R}$ (which is not necessarily injective, according to [3, 4]). Indeed, for a given ciphertext e, at most one r is acceptable. Without having asked $H(m \,\|\, b)$, the probability for $\mathcal{R}(H(m \,\|\, b))$ to be equal to r is less than ε_r:

$$\varepsilon_r = \max_{\mathsf{r}}\{\Pr_{x \in \{0,1\}^{\mathsf{rLen}}}[\mathsf{r} = \mathcal{R}(x)]\}.$$

However, to lift any security level from the CPA scenario to the CCA-one, one needs a good simulation of the decryption oracle: since any valid ciphertext has been correctly constructed, one looks at the list of the query-answers to the H-oracle, and can re-encrypt each possible plaintext to check which one is the good one. This perfectly simulates the decryption of validly generated ciphertexts, unless a *decryption failure* occurs (Fail event). In the latter case, the output decrypted plaintext is not the encrypted one, whereas the output simulated plaintext is: the probability of bad simulation is finally less than $\varepsilon_r + p(\mathsf{m}, \mathsf{r}, \mathsf{h})$, where

$$p(\mathsf{m}, \mathsf{r}, \mathsf{h}) = \Pr_{\mathsf{f},\mathsf{g}}[\mathcal{D}_{\mathsf{f},\mathsf{f}_\mathsf{p}}(\mathcal{E}_\mathsf{h}(\mathsf{m}, \mathsf{r})) \text{ Fail } \mid \mathsf{h} = \mathsf{p} * \mathsf{g} * \mathsf{f}^{-1} \pmod q].$$

The security analyses in [21] were performed assuming that $p(\mathsf{m}, \mathsf{r}, \mathsf{h})$ is negligible (and even 0). But we shall see later that this is unfortunately not the case. We thus refine the security analyses and even show that the parameters have to be chosen differently.

4 Decryption Failures and Provable Security

4.1 Wrap Failures and Gap Failures

When decrypting, the recipient must place the coefficients of $\mathsf{a} = \mathsf{f} * \mathsf{e} \bmod q$ into the correct range $[A, A+q-1]$. To calculate A, we use the fact that convolution multiplication respects the homomorphism $(\mathsf{a} * \mathsf{b})(1) = \mathsf{a}(1) \cdot \mathsf{b}(1)$, where $\mathsf{a}(1)$ is the sum of the coefficients of the polynomial a. The decrypter knows $\mathsf{r}(1)$ and $\mathsf{h}(1)$, and so he can calculate $I = \mathsf{m}(1) = \mathsf{e}(1) - \mathsf{r}(1) \cdot \mathsf{h}(1) \bmod q$. Assuming $\mathsf{m}(1)$ lies in the range $[N/2 - q/2, N/2 + q/2]$, we can calculate the average value of a coefficient of $\mathsf{p} * \mathsf{r} * \mathsf{g} + \mathsf{f} * \mathsf{m}$ and set

$$A = \left\lfloor \frac{\mathsf{p}(1) \cdot \mathsf{r}(1) \cdot \mathsf{g}(1) + \mathsf{f}(1) \cdot I}{N} \right\rceil - \frac{q}{2}.$$

The assumption about the form of m is a reasonable one, because for randomly chosen m with $N = 251$ there is a chance of less than 2^{-56} that $\mathsf{m}(1)$ will be less than $N/2 - q/2$ or greater than $N/2 + q/2$. In any case, the value of $\mathsf{m}(1)$ is known to the encrypter, so they do not learn anything from a decryption failure based on m being too thick or too thin.

Having obtained A, we reduce a into the range $[A, A+q-1]$. If the actual values of any of the coefficients of $\mathsf{p} * \mathsf{r} * \mathsf{g} + \mathsf{f} * \mathsf{m}$ lie outside this range, decryption will produce the wrong message and this will be picked up in the re-encryption stage of decryption. However, if $\mathsf{p} * \mathsf{r} * \mathsf{g} + \mathsf{f} * \mathsf{m}$ has a width less than q, there is still an A for which decryption is possible. This case, where the initial choice of A does not work but there is a choice of A which could work, is referred to as a "*wrap failure*".

Wrap failures are more common than "*gap failures*", where the width of $\mathsf{p} * \mathsf{r} * \mathsf{g} + \mathsf{f} * \mathsf{m}$ is strictly greater than q. The standard [3] therefore recommends that, on the occurrence of a decryption failure, the decrypter adjusts the decryption range by setting A' successively equal to $A \pm 1, \pm 2, \ldots$, placing the coefficients of a into the new range $[A', A'+q-1]$, and performing the mod p reduction. This is to be carried out until A' differs from A by some set T, the "*wrapping tolerance*"; if decryption has not succeeded at that point, the decryption function outputs the invalid symbol $\perp$.

This method increases the chance that a ciphertext will eventually decrypt; however, an attacker with access to timing information can tell when this re-centering has occurred. For standard $N = 251$ parameters and NTRUEncrypt implemented as in NTRU products, a wrap failure on random m occurs once every 2^{21} messages, while a gap failure occurs about once every 2^{43} messages. The centering method above will therefore leak information at least once every million or so decryptions, and possibly more often if the attacker can carry out some precomputation as in [19]. For NTRUEncrypt as implemented following the EESS standard, the number of messages is much lower: for $N = 251$, a gap failure occurs once every 2^{25} message.

4.2 Provable Security

In order to deal with any padding, one needs more precise probability informations than just $p(\mathsf{m}, \mathsf{r}, \mathsf{h})$:

$$\begin{aligned} & & p_0 &= E_{\mathsf{h}}[\max_{\mathsf{m},\mathsf{r}}\{p(\mathsf{m}, \mathsf{r}, \mathsf{h})\}];\\ p(\mathsf{m}, \mathsf{h}) &= \Pr_{\mathsf{r}}[p(\mathsf{m}, \mathsf{r}, \mathsf{h})] & p_1 &= E_{\mathsf{h}}[\max_{\mathsf{m}}\{p(\mathsf{m}, \mathsf{h})\}];\\ p(\mathsf{h}) &= \Pr_{\mathsf{m}}[p(\mathsf{m}, \mathsf{h})] = \Pr_{\mathsf{m},\mathsf{r}}[p(\mathsf{m}, \mathsf{r}, \mathsf{h})] & p_2 &= E_{\mathsf{h}}[p(\mathsf{h})]. \end{aligned}$$

Note that all of these probabilities are averages over the whole space of h. Implicit in these definitions is the assumption that, even if some keys (f, g) are more likely than others to experience decryption failures, an adversary cannot tell from the public key h which private key is more failure-prone.

Clearly, one cannot ensure that $p(\mathsf{m}, \mathsf{r}, \mathsf{h})$ is small, so p_0 is likely to be non-negligible. However, in several paddings, $\mathsf{r} = \mathcal{R}(H(m \,\|\, b))$, where H is a random oracle, therefore the probability of bad simulation involves at least p_1, or even p_2. As discussed above, with recommended parameters, p_2 can be as small as 2^{-43}. Even this is not negligible, but better parameters may hopefully make these values negligible. However, there is a gap between the existence of such a pair (m, r) that makes a *decryption failure*, and the feasibility, for an adversary, to find/build some:

$$\mathsf{Succ}^{\mathcal{A}}_{\mathsf{fail}}(\mathsf{h}) = \Pr_{\mathsf{f},\mathsf{g}}[\mathcal{D}_{\mathsf{f},\mathsf{f_p}}(\mathcal{E}_{\mathsf{h}}(\mathsf{m}, \mathsf{r}))\ \mathsf{Fail} \mid \mathsf{h} = \mathsf{p} * \mathsf{g} * \mathsf{f_p} \bmod q, (\mathsf{m}, \mathsf{r}) \leftarrow \mathcal{A}(\mathsf{h})].$$

As above, one needs to study the probabilities over some classes of adversaries, or when the adversary does not have the entire control over m or r:

$$\begin{aligned} \tilde{p}_0(t) &= \max\{E_{\mathsf{h}}[\mathsf{Succ}^{\mathcal{A}}_{\mathsf{fail}}(\mathsf{h})], |\mathcal{A}| \leq t\}\\ \tilde{p}_1(t, Q) &= \textit{idem} \text{ where } (\mathsf{m}, y) \leftarrow \mathcal{A}(\mathsf{h}), \mathsf{r} = G(\mathsf{m}, y)\\ \tilde{p}_2(t, Q) &= \textit{idem} \text{ where } (x, y) \leftarrow \mathcal{A}(\mathsf{h}), \mathsf{m} = F(x, y), \mathsf{r} = G(x, y). \end{aligned}$$

In the above bounds, for $\tilde{p}_0(t)$, we consider any adversary whose running time is bounded by t. For $\tilde{p}_1(t, Q)$ and $\tilde{p}_2(t, Q)$, F and G are furthermore assumed to be random oracles, to which the adversary can ask up to Q queries. Clearly, for any t and any Q,

$$\tilde{p}_0(t) \leq p_0 \quad \tilde{p}_1(t, Q) \leq Q \times p_1 \quad \tilde{p}_2(t, Q) \leq Q \times p_2.$$

We now reconsider the SVES-1 padding scheme, keeping these probabilities in mind. We note that the adversary controls m, by deriving m and b from the required m_1 and m_2. However, r is out of the adversary's control. Therefore after q_D queries to the decryption oracle and q_H queries to the random oracle H, the probability of a decryption failure is less than $q_D \times \tilde{p}_1(t, q_H)$, where t is the running time of the adversary. Denoting by $T_{\mathcal{E}}$ the time for one encryption, one gets the following improvement for a CCA attacker over a CPA one:

$$\mathsf{Adv}^{\mathsf{ind-cca}}_{\Pi^3}(t) \leq \mathsf{Adv}^{\mathsf{ind-cpa}}_{\Pi^3}(t + q_H T_{\mathcal{E}}) + 2q_D \times (\varepsilon_r + \tilde{p}_1(t, q_H)) .$$

4.3 Improved Paddings

In [21], new paddings have been suggested, with better provable security (based on the NTRU inversion problem only). But decryption failures have been ignored again.

The OAEP-based Scheme — The first suggestion was similar to the SVES-1 padding, also using two more hash functions

$$F : \{0,1\}^{k_1} \to \{0,1\}^{k_2} \text{ and } G : \{0,1\}^{k_2} \to \{0,1\}^{k_1}.$$

One first computes $s = m \oplus G(b)$ and $t = b \oplus F(s)$. The ciphertext consists of $\mathcal{E}_{\mathsf{pk}}(s \,\|\, t; H(m \,\|\, b))$. Of course, the decryption checks the validity of r, relatively to $H(m \,\|\, b)$. The OAEP construction provides semantic security, while the H function strengthens it to chosen-ciphertext security (as already explained).

Here, the adversary can choose s, t directly, then reverse the OAEP construction to obtain m, b. However, they cannot control r. Therefore

$$\mathsf{Adv}^{\mathsf{ind-cca}}_{\mathsf{oaep}'}(t) \leq 2\mathsf{Succ}^{\mathsf{ow}}_{\mathsf{ntru}}(t + QT_{\mathcal{E}}) + 2q_D \times (\varepsilon_r + \tilde{p}_1(t, q_H)) + \frac{4q_H}{2^{k_1}} + \frac{2q_G}{2^{k_2}}.$$

where $Q = q_F q_G + q_H$. But this makes a quadratic reduction, as for any OAEP-based cryptosystem [2, 5]. The particular above construction admits a better reduction, but under a stronger computational assumption.

SVES-2 — The successor to SVES-1 proposed for standardization is a minor variant of the above [4], designed to handle variable length messages. In SVES-2, one uses two hash functions F and G, and form $M_1 = b \,\|\, \mathrm{len}(m) \,\|\, m_1$, $M_2 = m_2 \,\|\, 000\ldots$. In SVES-2, the message length is restricted to be an integer number of bytes, and the length is encoded in a single byte. The final N mod 8 bits of M_2 will always be zeroes (we use N_8 to denote N mod 8). We form $s = M_2 \oplus F(M_1)$, $t = M_1 \oplus G(s)$, and calculate the ciphertext as $\mathcal{E}_{\mathsf{pk}}(t \,\|\, s; H(m \,\|\, b))$. On decryption, the usual checks are performed, and in addition the decrypter checks that the length is valid and that M_2 consists of 0s from the end of the message onwards.

In this case, an attacker who chooses s, t and reverses the OAEP construction must get the correct length of m and the correct 0 bits at the end. However, an attacker can choose s, then select t such that $t \oplus G(s)$ has the correct form to be M_1, that is, such that $\text{len}(m)$ is its maximum value. The reverse OAEP operation on s, t will then yield a valid M_1, M_2 if the last N_8 bits of $s \oplus F(M_1)$ are 0. Therefore an attacker can control all the bits of s, t with probability 2^{8+N_8}, or all but 8 of the bits of s, t with probability 2^{N_8}. Therefore

$$\mathsf{Adv}^{\mathsf{ind-cca}}_{\mathrm{SVES-2}}(t) \leq 2\mathsf{Succ}^{\mathsf{ow}}_{\mathsf{ntru}}(t + QT_{\mathcal{E}}) + 2q_D \times \left(\varepsilon_r + \frac{\tilde{p}_1(t, q_H)}{2^{N_8}}\right) + \frac{4q_H}{2^{k_1}} + \frac{2q_G}{2^{k_2}}.$$

where $Q = q_F q_G + q_H$.

NTRU-REACT — Thanks to the OW-PCA–security level of the NTRU primitive (granted the pseudo-inverse of h), one can directly use the REACT construction [23], in which the decryption algorithm of course checks the validity of c, but also the validity of r. The semantic security is clear, since the adversary has no advantage without having asked some crucial queries to the hash functions. With chosen-ciphertext attacks, the adversary cannot produce a valid ciphertext without having built correctly the authentication tag, except with probability $1/2^{k_2}$. Therefore, the simulation of the decryption oracle is perfect, unless a decryption failure occurs: the adversary knows b and thus m, but makes a decryption failure, that is not detected by the simulation. Since the adversary has the control over both m and r, the security against chosen-ciphertext attacks is not very high:

$$\mathsf{Adv}^{\mathsf{ind-cca}}_{\mathsf{react}}(t) \leq 2\mathsf{Succ}^{\mathsf{ow}}_{\mathsf{ntru}}(t + (q_G + q_H)T_{\mathcal{E}}) + 2q_D \times \left(\frac{1}{2^{k_2}} + \tilde{p}_0(t)\right).$$

In the *Improved NTRU-REACT*, the adversary completely loses control over r, which improves the security level, but not enough, since $\tilde{p}_0(t)$ is replaced by $\tilde{p}_1(t)$ only.

4.4 Comments

It is clear that decryption failures on valid ciphertexts mean that NTRUENCRYPT with the parameter sets given cannot have provable security to the level claimed. In the presence of decryption failures, it is impossible to correctly simulate the decryption oracle: the simulator will output a valid decryption on certain ciphertexts which the genuine decryption engine will fail to decrypt. Without knowledge of the private key, it is impossible to build a simulator; and if the simulator requires knowledge of the private key, it is impossible to have provable security. In the next section we will see how to use this information to recover the private key with considerably less effort than a standard lattice attack would take.

5 Some Attacks Based on Decryption Failures

In this section we present some attacks against NTRUENCRYPT as implemented in NTRU Cryptosystems' products for the $N = 251$ parameter set. See [4] for details of this parameter set, and Appendix A for information about the precise structure of f, g and r used. We stress that the attacks work even better on the EESS standard [3, 4], because decryption failures arise much more frequently there. Although no paddings can prevent decryption failures, it turns out that some paddings are more prone to attacks based on decryption failures: this is because the attacker has more or less flexibility on the choice of the actual message and random nonce actually given as input to the encryption primitive, depending on the padding used.

5.1 Review: The Reversal of a Polynomial

Before outlining the attacks, we review the notion of the *reversal* $\bar{\mathsf{c}}(X) \equiv \mathsf{c}(X^{-1})$ of a polynomial c. If we represent c as the array $\mathsf{c} = [\mathsf{c}_0, \mathsf{c}_1, \mathsf{c}_2, \ldots, \mathsf{c}_{N-1}]$ then its reversal is $\bar{\mathsf{c}} = [\mathsf{c}_0, \mathsf{c}_{N-1}, \mathsf{c}_{N-2}, \ldots, \mathsf{c}_1]$: this is a ring automorphism. We denote by $\hat{\mathsf{c}}$ the product of c and $\bar{\mathsf{c}}$. This product has the property that $\hat{\mathsf{c}}_i = \mathsf{c} \cdot (X^i * \mathsf{c})$, or in other words that the successive terms of $\hat{\mathsf{c}}$ are obtained by taking the dot product of c with successive rotations of itself. The significance of this is that we know that $\hat{\mathsf{c}}_0 = \sum_i \mathsf{c}_i^2 = \|\mathsf{c}\|^2$, while the other terms of $\hat{\mathsf{c}}$ will be $\mathcal{O}(\|\mathsf{c}\|)$ in size.

We therefore know that $\mathsf{f} * \bar{\mathsf{f}}$ has one term of size d_f, and the others are of size about $\sqrt{d_f}$. The polynomial $\mathsf{f} * \bar{\mathsf{f}}$ is therefore of great width compared to a product of two arbitrary polynomials of the same norm as f. We therefore assume that whenever $\mathsf{p} * \mathsf{r} * \mathsf{g} + \mathsf{f} * \mathsf{m}$ is of great width, it means that r is correlated significantly with $\bar{\mathsf{g}}$ and m is correlated significantly with $\bar{\mathsf{f}}$.

5.2 A General Attack

We derive a powerful chosen-ciphertext attack that works independently of the padding used. Indeed, assume that an attacker is able to collect many triplets $(\mathsf{m}, \mathsf{r}, \mathsf{e})$ such that e is an encryption of m with random nonce r, which cannot be correctly decrypted. If the probability of decryption failure is sufficiently high, an attacker could obtain such triplets by mounting a weak chosen-ciphertext attack, independently of the padding, by simply selecting random messages, encrypting them until decryption failures occur (which can be checked thanks to the decryption oracle). Interestingly, such an attack only uses valid ciphertexts.

For such a triplet $(\mathsf{m}, \mathsf{r}, \mathsf{e})$, we know that $\mathsf{p} * \mathsf{r} * \mathsf{g} + \mathsf{f} * \mathsf{m}$ is of great width, and the previous section suggests that there is an integer i such that r somehow looks like $X^i * \bar{\mathsf{g}}$ and m somehow looks like $X^i * \bar{\mathsf{f}}$. Unfortunately, we do not know the value of i, otherwise it would be trivial to recover $\bar{\mathsf{f}}$ by simply taking the average of $X^{-i} * \mathsf{m}$. However, we can get rid of the unknown i by using the reversal: if m and r look like respectively $X^i * \bar{\mathsf{f}}$ and $X^i * \bar{\mathsf{g}}$, then $\hat{\mathsf{m}}$ and $\hat{\mathsf{r}}$ must look like respectively $\hat{\mathsf{f}}$ and $\hat{\mathsf{g}}$. Once $\hat{\mathsf{f}}$ and $\hat{\mathsf{g}}$ are derived by averaging methods, f and g themselves may be recovered in polynomial time using an algorithm due to Gentry and Szydlo [7]. Strictly speaking, to apply [7], one also needs to determine the ideal spanned by f which can be derived from $\hat{\mathsf{f}} = \mathsf{f} * \bar{\mathsf{f}}$ and $\mathsf{f} * \bar{\mathsf{g}}$ (which is itself obtained by multiplying from $\bar{\mathsf{H}}$ and $\hat{\mathsf{f}} = \mathsf{f} * \bar{\mathsf{f}}$).

B	Messages
18	100,000

binary f

B	Messages	Norm (guess - $\hat{\mathsf{f}}$)
36	500,000	15.5
36	1,400,000	7.38

$\mathsf{f} = 1 + \mathsf{p} * \mathsf{F}$

Table 1. Messages to recover $\hat{\mathsf{f}}$ for various values of B using $\mathcal{O}_B$.

To check the validity of this attack, we would need to find a lot of decryption failures, which is relatively time-consuming, depending on the parameters. Instead, we checked that the attack worked, by experimenting with a weaker attack based on the following oracle $\mathcal{O}_B$: When the oracle $\mathcal{O}_B$ is queried on a valid ciphertext e, it indicates whether or not the width of $\mathsf{p} * \mathsf{r} * \mathsf{g} + \mathsf{f} * \mathsf{m}$ is greater than B. Thus, the oracle $\mathcal{O}_q$ simply detects gap failures. By using values of B much smaller than q, we are able to verify the behavior of our attack in a reasonable time, with the following algorithm:

1. Set $\mathsf{u}, \mathsf{v} = 0$.
2. Generate a large number of valid ciphertexts $\mathsf{e} = \mathsf{r} * \mathsf{h} + \mathsf{m} \bmod q$. For each ciphertext e:
   (a) Call $\mathcal{O}_B(\mathsf{e})$.
   (b) If $\mathcal{O}_B(\mathsf{e})$ shows the width of a as being greater than B, set $\mathsf{u} = \mathsf{u} + \hat{\mathsf{m}}$; set $\mathsf{v} = \mathsf{v} + \hat{\mathsf{r}}$.
3. Divide u and v by the number of valid ciphertexts used.

Over a long enough transcript, u and v should converge to $\hat{\mathsf{f}}$ and $\hat{\mathsf{g}}$. We investigated this for f binary and for $\mathsf{f} = 1 + \mathsf{p} * \mathsf{F}$, to see how many messages with width greater than B were necessary to recover $\hat{\mathsf{f}}$ exactly. The results are shown in tables 1. Experimentally, we find that we approximate $\hat{\mathsf{f}}$ best by $\hat{\mathsf{f}}_i = (\langle \hat{\mathsf{m}} \rangle_i - 61.24747)/0.007882857$.

When the distance from the guess to $\hat{\mathsf{f}}$ is about $\sqrt{N}/2 \approx 7.9$, we can essentially recover $\hat{\mathsf{f}}$ by rounding. We can conclude that from a real decryption oracle ($k_2 \leq 128$) no more than a million decryption failures, and perhaps considerably fewer, are necessary to recover $\hat{\mathsf{f}}$ and $\hat{\mathsf{g}}$. This validates our general chosen-ciphertext attack which applies to all paddings, and shows that the security of NTRU encryption in the EESS standard [3, 4] clearly falls far short of the hoped-for level of 2^{80}.

We note that one could imagine an even more powerful attack, where the attacker would simply average $\hat{\mathsf{e}}$ for those e that cause wraps. For a sufficiently long transcript, this will converge to $A + B\hat{\mathsf{f}} + C\hat{\mathsf{g}}$. We have not investigated this idea in full – it will undoubtedly involve longer convergence times than the other attacks outlined above – but it is interesting that a successful attack may be mounted even by an entirely passive attacker.

5.3 A Specific Attack Based on Controlling m

The previous attack works against any padding and already emphasizes the importance of decryption failures on the security of NTRU encryption. Here, we describe a slightly more efficient chosen-ciphertext attack tailored to the SVES-1 padding scheme, based on the fact that an attacker essentially controls m directly (see above). This attack shows that certain paddings are weaker than others with respect to attacks based on decryption failures. We denote by $\mathsf{r_m}$ the value of r obtained from the m and b obtained from a given m in valid encryption. The strategy will be to try and cause wrap failures. We introduce the notation

$$\mathcal{B}_i = \{\text{binary polynomials with } i \text{ 1s and } N-i \text{ zeroes}\},$$

and denote by $\mathsf{Flat(c)}$ the operation of taking c and setting all terms that are 1 or more to be exactly equal to 1. Experimentally, the average width of the various polynomials is:

$$\langle \mathsf{Width}(\mathsf{p} * \mathsf{r} * \mathsf{g})\rangle \approx 41; \quad \langle \mathsf{Width}(\mathsf{f} * \mathsf{m})\rangle \approx 47; \quad \langle \mathsf{Width}(\mathsf{p} * \mathsf{r} * \mathsf{g} + \mathsf{f} * \mathsf{m})\rangle \approx 62.$$

If the attacker can increase the width of $\mathsf{p} * \mathsf{r} * \mathsf{g} + \mathsf{f} * \mathsf{m}$ to 128, he will cause a gap failure; alternatively, if he can add about 33 to the largest term in $\mathsf{p} * \mathsf{r} * \mathsf{g} + \mathsf{f} * \mathsf{m}$ while leaving the others essentially unchanged, this will cause a wrap failure. The following attack exploits this observation

Step 1: first cleaning of random strings. The attacker picks a random $\mathsf{F}_{15} \in \mathcal{B}_{15}$, $\mathsf{D} \in \mathcal{B}_5$, and forms

$$\mathsf{m} = \mathsf{Flat}((1+X) * \mathsf{D} * \mathsf{F}_{15}).$$

On decryption (with $\mathsf{p} = 2 + X$), a will include a term approximately equal to

$$(1+X) * (2+X) * \mathsf{f} * \mathsf{D} * \mathsf{F}_{15} = (2+3X+X^2) * \mathsf{f} * \mathsf{D} * \mathsf{F}_{15}.$$

If the 1s in F_{15} match a set of 15 1s in $\bar{\mathsf{f}}$, then we know that at least one term in $\mathsf{p} * \mathsf{F} * \mathsf{m}$ will be 45 or more because of the 3 in $(1+X) * (2+X)$. With high odds, this will mean that $\mathsf{p} * \mathsf{F} * \mathsf{m}$ has greater than average width, and so greater than average chance of causing a decryption failure. This lets the attacker attempt to identify substrings of $\hat{\mathsf{F}}$:

Attack: Step 1. The attacker picks a random $\mathsf{F}_{15} \in \mathcal{B}_{15}$, $\mathsf{D} \in \mathcal{B}_5$ and forms $\mathsf{m} = \mathsf{Flat}((1+X) * \mathsf{D} * \mathsf{F}_{15})$. For all rotations of m, he submits $\mathsf{e} = \mathsf{r_m} * \mathsf{h} + \mathsf{m} \bmod q$ to the decryption oracle. If any rotation of m causes a wrap, he stores F_{15}; otherwise, he discards it.

Obviously, there will be a large number of false positives in this step. An m might cause a decryption failure purely through luck; alternatively, an F_{15} which has an overlap of 14 rather than 15 with $\bar{\mathsf{F}}$ will have a good chance of causing a wrap failure. We cannot distinguish between the cases immediately, so the

Overlap	Pr[Overlap] (Theoretical)	Pr[Overlap]) (Experimental)	Pr[Wrap]	No. Left from 2^{30} F_{15}s
15	$2^{-20.7}$	–	$2^{-14.4}$	10
14	$2^{-15.135}$	$2^{-14.7}$	$2^{-15.5}$	200
13	$2^{-10.698}$	$2^{-10.70}$	$2^{-16.6}$	2,000
12	$2^{-6.981}$	$2^{-7.116}$	$2^{-17.5}$	20,000
11	$2^{-3.857}$	$2^{-4.030}$	$2^{-18.5}$	80,000
10	$2^{-1.423}$	$2^{-1.693}$	2^{-22}	40,000
9	–	$2^{-0.91}$	2^{-23}	25,000
8	–	$2^{-3.49}$	2^{-24}	2,000
7	–	$2^{-14.5}$	–	–

Table 2. Effects of first cleaning on a set of 2^{30} F_{15}

strategy is to take an initial set of random F_{15}s, and use the decryption oracle to "clean" them so that the resulting set has a greater proportion of strings of high overlap with f.

Table 2 shows the effect of the first cleaning on a set of 2^{30} random F_{15}s. The wrap probabilities were determined experimentally, using our knowledge of f, by generating strings of a specific overlap and testing to see if they caused wraps. The final column is given by $2^{30} \cdot \Pr[\text{overlap}] \cdot \Pr[\text{wrap}] \cdot N$, as we try all the rotations of each of the 2^{30} ms. The total number of queries to the decryption oracle is $N \cdot 2^{30} \approx 2^{38}$.

From table 2 we see that even the first cleaning is very effective: from a random set of size 2^{30} where most F_{15}s have an overlap of 9 with f, we have created a set of size about 2^{17} where the most commonly occurring overlap is 11. We now want to further improve the quality of our set of F_{15}s.

Step 2: second cleaning. The attacker now queries each surviving F_{15} by using different values of D to create ms, and observing whether or not these cause wraps.

Attack: Step 2. For each F_{15} that survived the first cleaning step, the attacker picks several $\mathsf{D} \in \mathcal{B}_5$. For each D, he forms $\mathsf{m} = \mathsf{Flat}((1+X) * \mathsf{D} * \mathsf{F}_{15})$. For all rotations of m, he submits $\mathsf{e} = \mathsf{r}_\mathsf{m} * \mathsf{h} + \mathsf{m} \bmod q$ to the decryption oracle. If any rotation of m causes a wrap, he stores F_{15}; otherwise, he discards it.

Table 3 shows the results of performing this step, choosing 2^8 Ds for each F_{15}. After this cleaning, there are almost no F_{15}s left with an overlap of less than 11. The total work in this stage is $2^8 \cdot 2^{17} \cdot N \approx 2^{33}$, and there are about 2^{12} F_{15}s left.

Now that the attacker has a relatively good set of F_{15}, he can try and assemble them to recover F.

Step 3: find correct relative rotations. Here the challenge is to find the correct rotations of the F_{15} relative to each other. One possibility would be to

Overlap	No. Tested	No. Left
15	10	3
14	200	40
13	2,000	120
12	20,000	800
11	80,000	1500
10	40,000	2
9	25,000	1
8	2,000	0

Table 3. Effects of second cleaning on the set of F_{15}

pick two F_{15}s and test the ms obtained by all rotations of the two against each other. However, it appears that we get better results by picking sets of three F_{15}s and trying all their relative rotations.

Attack: Step 3. Let $\mathsf{F}_1, \mathsf{F}_2, \mathsf{F}_3$ be any three of the F_{15} that survived the second cleaning step. For each $0 \leq i, j < N$, set

$$\mathsf{F}_{\sim 45} = \mathsf{Flat}(\mathsf{F}_1 + X^i \mathsf{F}_2 + X^j \mathsf{F}_3)$$

(Note that $\mathsf{F}_{\sim 45}$ will typically have slightly fewer than 45 1s). Set $\mathsf{m} = \mathsf{Flat}((1 + X) * \mathsf{F}_{\sim 45})$. For all rotations of m, submit $\mathsf{e} = \mathsf{r}_\mathsf{m} * \mathsf{h} + \mathsf{m} \bmod q$ to the decryption oracle. Store the number of wraps caused by m. Once all i, j pairs have been exhausted, pick another three $\mathsf{F}_1, \mathsf{F}_2, \mathsf{F}_3$ and repeat. Continue until all the F_{15} have been used.

Figure 1 shows the wrap probability for m obtained as specified above. If $\mathsf{F}_1, \mathsf{F}_2, \mathsf{F}_3$ are rotated correctly relative to each other, the overlap with $\bar{\mathsf{F}}$ will typically be 33 or more, leading to a significantly greater chance of a wrap. Note that, because of the $(2 + X)$ in f, if $\mathsf{F}_1 + X^i \mathsf{F}_2 + X^j \mathsf{F}_3$ is the correct relative alignment of $\mathsf{F}_1, \mathsf{F}_2, \mathsf{F}_3$, a large number of wraps will also be caused by $\mathsf{F}_1 + X^{i \pm 1} \mathsf{F}_2 + X^{j \pm 1} \mathsf{F}_3$. This helps us to weed out freak events: rather than simply taking the relative rotation of $\mathsf{F}_1, \mathsf{F}_2, \mathsf{F}_3$ that gives the highest number of wraps, we look for the set of three consecutive rotations that give the highest total number of wraps and pick the rotation in the middle.

This step takes about $N^3 \cdot 2^{12} \approx 2^{36}$ work, and at the end of it we have about 2^{10} strings of length about 45, which will in general have an overlap of 33 or more with $\bar{\mathsf{F}}$. The remaining task is to rotate these strings correctly relative to each other and recover $\bar{\mathsf{F}}$ from them, but this is relatively trivial.

Step 4: from 45s to $\bar{\mathsf{F}}$. We here use the fact that the $\mathsf{F}_{\sim 45}$ will be better correlated with each other when rotated correctly than in other rotations.

Attack: Step 4. Sort the $\mathsf{F}_{\sim 45}$ from the previous step in order by the number of wraps they caused. Set u equal to the $\mathsf{F}_{\sim 45}$ at the top of the list. For each other

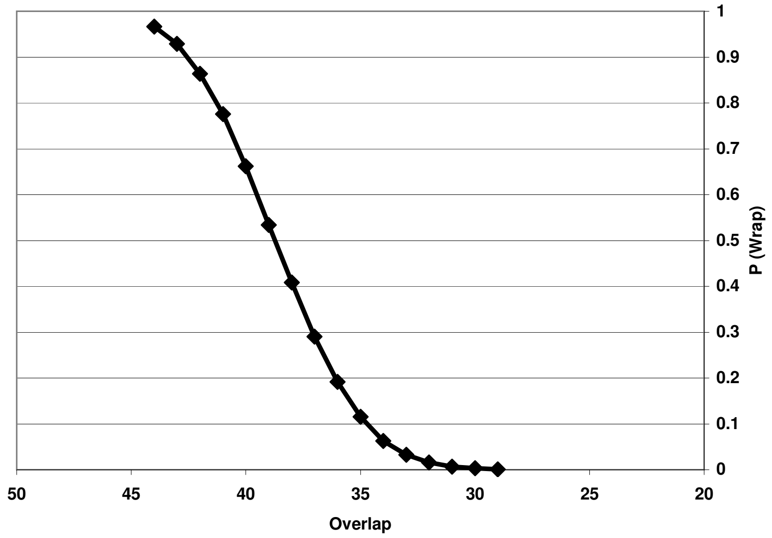

Fig. 1. Wrap probability on a single test for strings of length 45

$\mathsf{F}_{\sim 45}$, find the X^i that maximizes the overlap of $X^i \cdot \mathsf{F}$ and the reference F. Set $\mathsf{u} = \mathsf{u} + X^i \cdot \mathsf{F}$. When all the Fs have been added, take the top d_f entries of u and set them equal to 1. Set the other entries to 0. This recovers F_{rev}.

Since there are 72 1s in $\bar{\mathsf{F}}$ and 179 0s, and since the $\mathsf{F}_{\sim 45}$ have typically 33 correct 1s and 12 incorrect 1s, we expect the entries in u corresponding to 1s in f_{rev} to have an average value of $33/72 \approx 0.46$, and the entries in u corresponding to 0s in f_{rev} to have an average value of $12/179 \approx 0.06$. This makes it easy to distinguish between the two. We have not implemented this part of the attack, but we do not anticipate any problems in its execution.

SVES-1 attack: Summary We have presented an attack on the SVES-1 scheme that allows an attacker with access to decryption timing information to recover the private key in about 2^{40} queries to a decryption oracle with $N = 251$. This is a level of security that clearly falls far short of the hoped-for level of 2^{80}.

6 Countermeasures

NTRUEncrypt as specified in [3] clearly falls short of the desired security levels, since it only involves the probability $\tilde{p}_1$. With the given parameters, even $\tilde{p}_2$ is likely to be non-negligible. One should thus recommend at least the following two countermeasures.

6.1 Changing the parameters

The parameters, and perhaps the form of f, g, and r, should be altered so that decryption failures occur no more often than the claimed security level of the parameter set, so that the probability $\tilde{p}_2$, or even p_2, is indeed negligible. (For example, for $N = 251$, an attacker should be required to carry out 2^{80} work to find a single gap failure). Unfortunately, no efficient method is known to provably compute such a probability, though the paper [27] provides calculations under some simplifying assumptions.

6.2 Changing the padding

A padding scheme with the appropriate provable security properties should be adopted. We have presented both theoretical and experimental reasons for preferring an NTRUEncrypt padding scheme in which an attacker can control neither m nor r. Theoretically, only this padding scheme allows us to use p_2, the smallest of the expected decryption failure probabilities. Experimentally, we have demonstrated an attack which uses direct control of m to recover the private key faster than the attack which does not use control of m.

We therefore suggest the following padding scheme, which we call NAEP, as one that might be suitable for NTRUEncrypt. The construction uses the hash functions

$$G : \{0,1\}^{\mathsf{mLen}} \to \{0,1\}^{\mathsf{rLen}} \text{ and } H : \mathcal{P} \to \{0,1\}^{\mathsf{mLen}}.$$

As before, m is the plaintext of length k_1 bits, and b is a random string, unique for each message, of length $k_2 = \mathsf{mLen} - k_1$ bits. One computes $\mathsf{r} = \mathcal{R}(G(m \,\|\, b))$ and $\mathsf{R} = \mathsf{r}*\mathsf{h} \bmod q$. Then the ciphertext consists of $\mathcal{E}_{\mathsf{pk}}((m \,\|\, b) \oplus H(\mathsf{R}); G(m \,\|\, b))$. Of course, the decryption checks the validity of r, relatively to $G(m \,\|\, b)$.

We do not make any claim on the provable security of this scheme. An analysis of the properties of a variant of this scheme, with a specific instantiation of H, appears in [16] and claims a security result which depends on $\tilde{p}_2$ only (and of course the intractability of the basic NTRU primitive.)

Acknowledgments

We would like to thank Jeff Hoffstein and Jill Pipher for fruitful discussions and contributions.

References

1. M. Bellare, A. Desai, D. Pointcheval, and P. Rogaway. Relations among Notions of Security for Public-Key Encryption Schemes. In *Crypto '98*, LNCS 1462, pages 26–45. Springer-Verlag, Berlin, 1998.
2. M. Bellare and P. Rogaway. Optimal Asymmetric Encryption – How to Encrypt with RSA. In *Eurocrypt '94*, LNCS 950, pages 92–111. Springer-Verlag, Berlin, 1995.

3. EESS: Consortium for Efficient Embedded Security. Efficient Embedded Security Standards #1: Implementation Aspects of NTRU and NSS. Draft Version 3.0 available at `http://www.ceesstandards.org`, July 2001.
4. EESS: Consortium for Efficient Embedded Security. Efficient Embedded Security Standards #1: Implementation Aspects of NTRUEncrypt and NTRUSign. Version 1.0 available at `http://www.ceesstandards.org`, November 2002.
5. E. Fujisaki, T. Okamoto, D. Pointcheval, and J. Stern. RSA–OAEP is Secure under the RSA Assumption. In *Crypto '01*, LNCS 2139, pages 260–274. Springer-Verlag, Berlin, 2001.
6. C. Gentry. Key Recovery and Message Attacks on NTRU-Composite. In *Eurocrypt '01*, LNCS 2045, pages 182–194. Springer-Verlag, Berlin, 2001.
7. C. Gentry and M. Szydlo. Cryptanalysis of the Revised NTRU Signature Scheme. In *Eurocrypt '02*, LNCS 2332, pages 299–320 Springer-Verlag, Berlin, 2002.
8. S. Goldwasser and S. Micali. Probabilistic Encryption. *Journal of Computer and System Sciences*, 28:270–299, 1984.
9. C. Hall, I. Goldberg, and B. Schneier. Reaction Attacks Against Several Public-Key Cryptosystems. In *Proc. of ICICS '99*, LNCS, pages 2–12. Springer-Verlag, 1999.
10. J. Hoffstein and J. Pipher and J. H. Silverman. NTRU: A Ring-Based Public Key Cryptosystem. In *Proc of ANTS 3*, LNCS 1423, pages 267–288. Springer-Verlag, 1998.
11. J. Hoffstein and J. H. Silverman. Random Small Hamming Weight Products With Applications To Cryptography. *Discrete Applied Mathematics*. To appear, available at [22].
12. J. Hoffstein and J. H. Silverman. Invertibility in Truncated Polynomial Rings. Technical report, NTRU Cryptosystems, October 1998. Report #009, version 1, available at [22].
13. J. Hoffstein and J. H. Silverman. Optimizations for NTRU. In *Public-key Cryptography and Computational Number Theory*. DeGruyter, 2000. To appear, available at [22].
14. J. Hoffstein and J. H. Silverman. Protecting NTRU against Chosen Ciphertext and Reaction Attacks. Technical report, NTRU Cryptosystems, June 2000. Report #16, version 1, available at [22].
15. J. Hong, J. W. Han, D. Kwon and D. Han. Chosen-Ciphertext Attacks on Optimized NTRU. Cryptology ePrint Archive: Report 2002/188.
16. N. Howgrave-Graham, J. H. Silverman, A. Singer and W. Whyte. NAEP: Provable Security in the Presence of Decryption Failures. Cryptology ePrint archive, `http://eprint.iacr.org`.
17. E. Jaulmes and A. Joux. A Chosen Ciphertext Attack on NTRU. In *Crypto '00*, LNCS 1880, pages 20–35. Springer-Verlag, Berlin, 2000.
18. A. May and J.H. Silverman. Dimension Reduction Methods for Convolution Modular Lattices. In *Proc. of CaCL 2001*, LNCS 2146, pages 110–125. Springer-Verlag, 2001.
19. T. Meskanen and A. Renvall. Wrap Error Attack Against NTRUEncrypt. To appear in *Proc. of WCC '03*.
20. M. Naor and M. Yung. Public-Key Cryptosystems Provably Secure against Chosen Ciphertext Attacks. In *Proc. of the 22nd STOC*, pages 427–437. ACM Press, New York, 1990.
21. P. Q. Nguyen and D. Pointcheval. Analysis and Improvements of NTRU Encryption Paddings. In *Crypto '02*, LNCS 2442, pages 210–225. Springer-Verlag, Berlin, 2002.

22. NTRU Cryptosystems. Technical reports. Available at `http://www.ntru.com`, 2002.
23. T. Okamoto and D. Pointcheval. REACT: Rapid Enhanced-security Asymmetric Cryptosystem Transform. In *CT – RSA '01*, LNCS 2020, pages 159–175. Springer-Verlag, Berlin, 2001.
24. J. Proos. Imperfect Decryption and an Attack on the NTRU Encryption Scheme. Cryptology ePrint Archive: Report 2003/002.
25. C. Rackoff and D. R. Simon. Non-Interactive Zero-Knowledge Proof of Knowledge and Chosen Ciphertext Attack. In *Crypto '91*, LNCS 576, pages 433–444. Springer-Verlag, Berlin, 1992.
26. J. H. Silverman. Estimated breaking times for NTRU lattices. Technical report, NTRU Cryptosystems, March 1999. Report #012, version 1, available at [22].
27. J. H. Silverman and W. Whyte, Estimating Decryption Failure Probabilities for NTRUEncrypt . Technical report, NTRU Cryptosystems, May 2003. Report #018, version 1, available at [22].

A Standardized and Deployed Versions of NTRUEncrypt

A.1 Format of Objects

NTRUENCRYPT as standardized in [3] uses special forms for f, g and r, and specifies a padding method which is claimed to give provable security. We review these details here.

Private key: f. The private key f has two special features. First, f has the form $1 + \mathsf{p} * \mathsf{F}$, where F is a binary polynomial. This means that $\mathsf{f} = 1 \bmod \mathsf{p}$, and therefore that $\mathsf{f}_\mathsf{p} = 1$, eliminating the need for the second convolution on decryption [13]. Second, the binary polynomial F is of the form $\mathsf{f}_1 * \mathsf{f}_2 + \mathsf{f}_3$, where f_1, f_2 and f_3 are chosen so that:

- $\mathsf{f}_1, \mathsf{f}_2, \mathsf{f}_3$ are binary and have $d_{f_1}, d_{f_2}, d_{f_3}$ 1s respectively;
- $\mathsf{f}_1 * \mathsf{f}_2$ is binary;
- The 1s in f_3 are chosen such that they are not adjacent to any of the 1s in $\mathsf{f}_1 * \mathsf{f}_2$.

It should be pointed out that only the first of these restrictions is documented in [3]: the description above is of the form of private keys used in NTRU CRYPTOSYSTEMS' software and has the effect of decreasing the occurrence of decryption failures (but not to the point of making decryption failures sufficiently unlikely to avoid any security problems). For more details on the use of products of low Hamming weight polynomials in NTRU and other cryptosystems, see [13, 11].

Private key: g. The private key g is chosen to be binary, to have d_g 1s, and to have no consecutive 1s. Note that the last of these restrictions is not documented in [3], but is used in NTRU CRYPTOSYSTEMS' software.

Message: m. The message representative m is a binary polynomial of degree N. An algorithm for converting from octet strings to binary polynomials can be found in [3].

Blinding value: r. The blinding value r is chosen to be of the form $\mathsf{r}_1 * \mathsf{r}_2 + \mathsf{r}_3$. Here, $\mathsf{r}_1, \mathsf{r}_2, \mathsf{r}_3$ are generated by setting them to 0, then selecting $d_{r_1}, d_{r_2}, d_{r_3}$ indices between 0 and $N-1$ and adding one to the coefficient at each index. The difference between this and taking $\mathsf{r}_1, \mathsf{r}_2, \mathsf{r}_3$ to be binary is that the indices used to generate them can repeat: for example, r_1 could consist of $d_{r_1} - 2$ 1s and one 2. A recent paper [19] uses this specific fact to recover g. The results presented here are more general and work for r of any form in the presence of decryption failures.

A.2 Encryption Schemes

There have been several encryption schemes associated with NTRU, but only two have been standardized in EESS #1. The first, SVES-1, proceeds as follows. It takes the number of random bits, d_b, as a parameter, and hash functions F, G, H.

Encryption – To encrypt a message m of length $|m| = N - d_b$ bits:

1. Generate the string b consisting of d_b random bits.
2. Set s equal to the first $|m|/2$ bits of m concatenated with the first $d_b/2$ bits of b. Set t equal to the last $|m|/2$ bits of m concatenated with the last $d_b/2$ bits of b.
3. Set $t' = t \oplus F(s)$. Set $s' = s \oplus G(t')$. Set $\mathsf{m}' = s'||t'$. Set $\mathsf{r} = H(m, b)$.
4. Output the ciphertext $\mathsf{e} = \mathsf{r} * \mathsf{h} + \mathsf{m}' \bmod q$.

Decryption – To decrypt the ciphertext e:

1. Recover m' from e using standard NTRUEncrypt decryption.
2. Recover m, b from m' by reversing the masking defined above.
3. Set $\mathsf{r} = H(m, b)$ and calculate $\mathsf{e}' = \mathsf{r} * \mathsf{h} + \mathsf{m}' \bmod q$. If the result is the same as the received e, output m. Otherwise, output "invalid ciphertext".

SVES-1 was shown in [21] to have inadequate provable IND-CPA properties, due to the decision to split b into two parts. EESS #1 therefore also specifies SVES-2, which is similar to SVES-1 with the exception that all of b is included in the first hash function call. There are other minor differences between the two schemes – SVES-2 is designed to allow variable-length messages more gracefully, for example – but these are more engineering than cryptographic decisions.

One interesting fact to note is that in both SVES-1 and SVES-2 the message is randomized by the mask generation functions, but an adversary is free to choose the value of m' directly and then reverse the masking operation to find the m and b that would have given that m'.

B Another chosen-ciphertext attack

Here we present a brief overview of a second chosen-ciphertext attack against NTRUEncrypt. The attack is based on decryption failures; however, unlike the other attack presented in this paper, does not rely on the secret polynomial f being of the form $1+(2+X)F$. In fact, the new attack is not specific to the case of $\mathsf{p} = 2 + X$ and can also be applied against the originally proposed version of NTRU [10] which had $\mathsf{p} \in \mathbb{Z}$.

The attack assumes that the attacker can detect wrap errors and that the r values used during encryption must be selected at random. For the basic version of the attack we also assume that a message polynomial m can be encrypted with many random r (as is the case for the proposed NTRU-REACT padding schemes). We will discuss below the effect on the attack if each m yields a unique r_{m}. The basic version of the attack consists of repeating the following three steps until the secret key is revealed.

Step 1: finding a decryption failure. The goal of step 1 is to determine a valid pair m, r which lead to a decryption failure. The most straight forward approach is to simply select random m and r until the ciphertext they generate causes a decryption failure.

Instead of a random search, it is also possible to perform a systematic search for the required decryption failure. Given an m and r an attacker can determine exactly the set, $I_{\mathsf{m},\mathsf{r}}$, of (f, g) pairs for which m, r will cause a decryption failure. Determining if m, r causes a decryption failure reveals whether or not (f, g) is in $I_{\mathsf{m},\mathsf{r}}$. So instead of simply selecting m and r at random an attacker could perform some precomputation and obtain a list of m, r pairs for which $\bigcup I_{\mathsf{m},\mathsf{r}}$ is larger than it would have been in a random search.

Step 2: search for more r's. For the majority of m, r pairs which lead to decryption failures, $\mathsf{m} * \mathsf{f}$ will have one coefficient, i, which is both abnormally far from its expected value and further from the expected value than any other coefficient. We shall refer to the difference in the distances of the two coefficients of $\mathsf{m} * \mathsf{f}$ furthest from their expected value as the *gap* of $\mathsf{m} * \mathsf{f}$. The true goal of step 1 is actually to find an m such that $\mathsf{m} * \mathsf{f}$ has both a coefficient which is far from its expected value and a large gap.

Attack: Step 2. By repeatedly picking random r' and determining if m, r' leads to a decryption failure, the attacker can determine a list $\mathsf{r}_0, \mathsf{r}_1, \ldots, \mathsf{r}_k$ of r values which cause decryption failures when used with m.

Suppose that step 1 found an m with the desired properties. The range $[A, A+q-1]$ used during decryption is centered at the expected value of the coefficients of $\mathsf{p} * \mathsf{r}' * \mathsf{g} + \mathsf{f} * \mathsf{m}$. Thus, since the ith coefficient of $\mathsf{m} * \mathsf{f}$ is abnormally far from its expected value, the rate at which the m, r' cause decryption failures will be much higher than for random m, r. Furthermore, the expected value of every coefficient of $\mathsf{p} * \mathsf{r} * \mathsf{g}$ is $\mathsf{p}(1)\mathsf{g}(1)\mathsf{r}(1)/N$. Thus when an m, r' pair causes a decryption failure its most likely cause is the ith coefficient of $\mathsf{p} * \mathsf{r}' * \mathsf{g} + \mathsf{f} * \mathsf{m}$. The strength of this bias towards the i coefficient the will depend on the gap of $\mathsf{m} * \mathsf{f}$. This bias will cause a correlation between the $\mathsf{r}_0, \mathsf{r}_1, \ldots, \mathsf{r}_k$ found in step 2 and $\bar{\mathsf{g}}$.

Step 3: recovering the secret key. If k is sufficiently large then the value of $\bar{\mathsf{g}}$ (and thus g) can be determined directly from the polynomials $\mathsf{r}_0, \mathsf{r}_1, \ldots, \mathsf{r}_k$. However, it is possible to find the secret key with fewer r_j than would allow the direct recovery of $\bar{\mathsf{g}}$. This is accomplished by using the r_j to determine some of coefficients of g and then using this partial knowledge of g in combination with the known lattice attacks on NTRU as in [18].

If the gap of $\mathsf{m} * \mathsf{f}$ is small then the bias towards the coefficient i may not be large enough to allow the recovery of the secret key. If this is the situation then the attack simply returns to step 1. Note that even if an iteration does not reveal the entire secret key some information may still have been determined.

Attack summary and variations. Two important questions arise regarding this attack. First, how much work is involved in one iteration of the steps? Second, how many iterations through the steps will be required? The number of iterations required depends on the maximum work allowed to be done in step 2 of an iteration. The more effort put into finding r_j's in step 2 the more likely step 3 is to succeed. Details on the running time of the attack against the $\mathsf{p} = 3$ parameters suggested in [26] can be found in [24]. Below we include some details on the running time against the $N = 251$ parameter set of NTRUEncrypt as standardized in [4].

If, as with the SVES-1 padding scheme, each polynomial m only has one valid r then the basic attack described above can not be used. The problem arises in step 2, where if m is held constant then the r' used will also be constant. To overcome this problem the attacker can, instead of keeping m fixed, use the cyclic shifts of both m and m with minor changes applied to it. Care must be taken to record the shift amounts with the r_j found so that the shifts can be undone in step 3.

B.1 Implementation results

The attack was implemented against 100 instances of the $N = 251$ parameter set of NTRUEncrypt as standardized in [4] taking $\mathsf{f} = 1 + pF$, where F was a binary polynomial. Our implementation of the attack put a bound of 3 million on the number of r checked in step 2, checked to see if the secret key could be recover after every 25 decryption failures in step 2, and aborted iterations in step 2 if the rate at which the r_j were found was below a given threshold. The implementation assumed that the secret key could be recovered when the dimension of the lattice which would need to be reduced was less than one hundred. Of the 100 instances of the attacks the number of instances which found the secret key after $1, 2, 3, 4, 5, 6, 7$ and 8 iterations were $48, 23, 17, 4, 4, 2, 1$

and 1 respectively. Table 4 shows the average number of m, r pairs tested and decryption failures required over the 100 instances of the attack and during the step 2's of the successful iterations.

	m, r Pairs Checked		Decryption Failures	
	Avg Number	Std Dev	Avg Number	Std Dev
Total Attack	1991909.11	1706591.03	170.65	130.56
Successful Step 2	842589.58	767601.34	118.74	43.77

Table 4. $N = 251$ attack details