NAEP: Provable security in the presence of decryption failures

Nick Howgrave-Graham
Joseph H. Silverman
Ari Singer
William Whyte

NTRU Cryptosystems,
5 Burlington Woods, MA 01803.

Abstract

We consider the impact of the possibility of decryption failures in proofs of security for padding schemes, where these failures are both message and key dependent. We explain that an average case failure analysis is not necessarily sufficient to achieve provable security with existing CCA2-secure schemes. On a positive note, we introduce NAEP, an efficient padding scheme similar to PSS-E designed especially for the NTRU one-way function. We show that with this padding scheme we can prove security in the presence of decryption failures, under certain explicitly stated assumptions. We also discuss the applicability of proofs of security to instantiated cryptosystems in general, introducing a more practical notion of *cost* to describe the power of an adversary.

1 Introduction

The issue of finding a correct padding scheme for NTRUEncrypt [8] has been an open one for some time, and incorrect choices of padding scheme have led to various attacks [13, 15, 16]. These schemes were flawed due to there being either a standard chosen ciphertext attack against them, or an attack in which the adversary takes advantage of the fact that NTRU decryption sometimes fails to decrypt validly formed ciphertexts.

Most recently a CCA2-secure padding scheme for NTRUEncrypt was given in [16], but the proof neglects to take decryption failures into account, and is thus flawed when instatiated with current NTRUEncrypt parameter sets. Such ideas are explored further in [12].

In this paper we give the first full proof of security of an NTRUEncrypt padding scheme, and explicitly state all the assumptions that the security rests on. Some of these assumptions are a little different from those commonly met in cryptography, just as is the existence of decryption failures on validly formed ciphertexts. The motivation for making such assumptions is driven by the following thoughts: it is theoretically interesting to consider the impact of the possibility of decryption failures on encryption schemes, the assumptions are apparently true, and making such assumptions yields practical improvements for NTRUEncrypt.

The padding scheme we propose uses hash functions in a similar way to PSS-E [6], though there are several modifications which are specific to the NTRU one-way function. The major contribution of this paper is showing that this construction is still secure in the presence of decryption failures, and making clear the necessary assumptions.

The proof technique we use allows us to prove security for any particular $\mathsf{NTRUEncrypt}$ parameter set, assuming the hardness of inverting an associated one-way function. Here, we use "parameter set" in a broad sense, to denote not simply the spaces that different secret values are chosen from but the generation methods that define their distribution within those spaces. Note that different choices of parameter sets or algorithms lead to different one-way functions[1], and each parameter set chosen should be carefully studied to ensure that the associated function is, in fact, as hard as required.

An additional contribution of this paper is a discussion of the applicability of proofs of security to real parameter sets. We introduce a *cost* function which allows us to capture the meaning of a statement like "eighty-bit security". Using this function, we are able to broaden the discussion of key-specific security issues to encompass not simply decryption failures, but weak keys of any sort.

1.1 Notation

Let $R = \mathbb{Z}[X]/(X^N - 1)$ for some global (prime) parameter $N \in \mathbb{Z}$. For any $a, b \in R$ we will use $a * b$ to denote the natural (convolution) multiplication of elements of R, and $a + b$ to denote the natural (component wise) addition of elements of R. We will use $\widehat{R} \subset R$ to denote the set of elements of R whose coefficients are restricted to be binary, use $\widehat{R}_d \subset \widehat{R}$ to denote the set of elements of $\widehat{R}$ whose elements have exactly d ones, and use $\widehat{R}_{[d_1,d_2)} \subset \widehat{R}$ to denote the set of elements of $\widehat{R}$ whose elements have a number of ones[2] in the range $[d_1, d_2)$. We use $\widetilde{R}$ to denote the space $\widehat{R}_{[(N-q)/2,(N+q)/2)}$ which is commonly used in $\mathsf{NTRUEncrypt}$, for given global parameters $N, q \in \mathbb{Z}$.

We will denote the modular subrings of R by $R_m = R/mR$ for some $m \in \mathbb{Z}[X]$. We will consider only the cases when m has degree 0 or 1. When $m \in \mathbb{Z}$ there is a natural representation of elements of R_m by elements of R whose coefficients are restricted to lie between 0 and $m - 1$. For any $a, b \in R_m$ we will use $a * b$ to denote the natural (convolution) multiplication of elements of R_m, and $a + b$ to denote the natural (component wise) addition of elements of R_m. If $a \in R$ and $b \in R_m$, or vice versa, then the product $a * b$, and sum $a + b$, will be considered to be elements of R_m rather than R, unless stated otherwise.

In the following we will often deal with bit strings of length N, which we denote $B = \{0, 1\}^N$, and elements of $\widehat{R}$. Clearly there is a bijective mapping between the two sets; mapping coefficients to bits. We shall use hats to denote elements of $\widehat{R}$, and upside down hats to denote the corresponding elements of B, e.g. if $\hat{a} \in \widehat{R}$, then $\check{a} \in B$ with the i'th bit set if and only if the i'th coefficient of $\hat{a}$ is one (and vice versa). We let $B_d \subset B$ denote the subset of B whose elements are the bit-strings with exactly d ones.

We write $s \xleftarrow{R} S$ to denote the process of picking an element s from a set S uniformly at random.

1.2 Abstract security notions

The standard specification of an encryption scheme includes a parameter generation algorithm $\mathcal{G}$, which takes as input a security parameter 1^k (typically written in unary notation), and returns a parameter set $\mathcal{P}$. Such a general algorithm has not been specified for $\mathsf{NTRUEncrypt}$ to date; instead several explicit parameters sets have been given.

In this report we will assume the existence of a parameter generation algorithm $\mathcal{G}$ for $\mathsf{NTRUEncrypt}$, and we will explicitly state the properties we expect of it. We do this so that we may

[1]Though we do not rule out the possibility of proving *some* equivalences via reduction algorithms.

[2]Note that d_1, d_2 need not be integers.

give a standard proof of security, which is classically stated in an asymptotic sense in the security parameter k.

Definition 1 *A function* $\nu : \mathbb{N} \to \mathbb{R}$ *is said to be* negligible *if for every constant* $c \geq 0$*, there exists an integer* k_c *such that* $\nu(k) < k^{-c}$ *for all* $k \geq k_c$*.*

We note that although they are conceptually attractive, standard proofs of security do not, by themselves, give assurances of security for any fixed parameter sets. This point is discussed further in section 3.1, where we state some further assumptions that are necessary for practical security.

For a given parameter set $\mathcal{P}$, the encryption scheme is specified by three algorithms

$$\begin{aligned} \mathcal{K} :& \quad \mathcal{R}_{\mathcal{K}} \to \mathcal{PK} \times \mathcal{SK} \\ \mathcal{E} :& \quad \mathcal{PK} \times \mathcal{M} \times \mathcal{R}_{\mathcal{E}} \to \mathcal{C} \\ \mathcal{D} :& \quad \mathcal{SK} \times \mathcal{C} \to \mathcal{M}, \end{aligned}$$

called the *key generation*, *encryption* and *decryption* algorithms respectively. The spaces $\mathcal{R}_{\mathcal{K}}$, $\mathcal{PK}$, $\mathcal{SK}$, $\mathcal{M}$, $\mathcal{R}_{\mathcal{E}}$, $\mathcal{C}$ are called the *key-gen randomness*, *public key*, *secret key*, *message*, *encryption randomness*, and *ciphertext* space respectively.

Classically if $(\mathsf{pk}_\kappa, \mathsf{sk}_\kappa) \leftarrow \mathcal{K}(\kappa)$ then the algorithms should satisfy

$$\mathcal{D}(\mathsf{sk}_\kappa, \mathcal{E}(\mathsf{pk}_\kappa, M, r)) \quad = \quad M$$

for all $\kappa \in \mathcal{R}_{\mathcal{K}}$, $M \in \mathcal{M}$ and $r \in \mathcal{R}_{\mathcal{E}}$.

In this paper we shall relax this requirement slightly, and only require that for all $M \in \mathcal{M}$,

$$\Pr[\mathcal{D}(\mathsf{sk}_\kappa, \mathcal{E}(\mathsf{pk}_\kappa, M, r)) \neq M] \quad \leq \quad \nu_{\mathcal{D}}(k)$$

for some negligible function $\nu_{\mathcal{D}}$ (with k taken to be sufficiently large), and where this probability is defined over $\kappa \stackrel{R}{\leftarrow} \mathcal{R}_{\mathcal{K}}$ and $r \stackrel{R}{\leftarrow} \mathcal{R}_{\mathcal{E}}$. We give an explicit construction of such a cryptosystem, from the NTRU family of pseudo-trapdoor one-way functions (we use pseudo-trapdoor to mean a trapdoor that is not guaranteed to work; see section 2.5 for more details).

For notational convenience we will often drop the public and secret key input to $\mathcal{E}$ and $\mathcal{D}$, and also drop the κ subscript from sk and pk.

Definition 2 *We say a time* t *algorithm* $\mathcal{A}$ *is a* (t, ϵ)*-chosen ciphertext algorithm, with advantage* ϵ *in attacking a randomized encryption scheme* $(\mathcal{K}, \mathcal{E}, \mathcal{D})$ *if there are a pair of subalgorithms*

$$\begin{aligned} \mathcal{A}_1 :& \quad \mathcal{PK} \to \mathcal{M} \times \mathcal{M} \times \mathcal{S} \\ \mathcal{A}_2 :& \quad \mathcal{C} \times \mathcal{S} \to \{0, 1\} \end{aligned}$$

such that if $(M_0, M_1, s) \leftarrow \mathcal{A}_1(\mathsf{pk})$ *then*

$$|\Pr[\mathcal{A}_2(c^*, s) = b^*] - 1/2| \quad = \quad (1/2)\epsilon$$

where $c^* \leftarrow \mathcal{E}(M^*, r^*)$ *for some* $r^* \in \mathcal{R}_{\mathcal{E}}$*, and* $M^* = M_{b^*}$ *for some* $b^* \in \{0, 1\}$*. This probability is defined over the choice of* $r^* \stackrel{R}{\leftarrow} \mathcal{R}_{\mathcal{E}}$*,* $b^* \in \{0, 1\}$ *and* $\kappa \in \mathcal{R}_{\mathcal{K}}$*.*

The algorithms $(\mathcal{A}_1, \mathcal{A}_2)$ *have access to a decryption oracle* $\mathcal{D}$*, which they can call on all but the challenge ciphertext* c^**, but they must make all hash function calls to* $H_1, \ldots H_n$ *public.*

The set $\mathcal{S}$ is just a means by which algorithm $\mathcal{A}_1$ can pass state to algorithm $\mathcal{A}_2$. For notational convenience this will typically be ignored.

This definition captures the notion of an adversary being able to distinguish between encryptions of some messages M_0 and M_1 regardless of the randomness r used (which would mean that r is not playing a useful role). Allowing the adversarial algorithms access to a decryption oracle is important because a high level protocol may well give away some information about each encryption; an adversary who cannot succeed, even when given a decryption oracle, cannot succeed if given partial information. Insisting that $\mathcal{A}$ must make explicit calls to all hash functions $H_1, \ldots, H_n$ means that we are working in the *random oracle model* as proposed in [1].

If there is no efficient algorithm $\mathcal{A}$ for a given encryption scheme, that scheme is said to have the property of being *indistinguishable against adaptive chosen-ciphertext attack*, often abbreviated to IND-CCA2. The property of being IND-CCA2 is widely accepted as being the "right" one for a public-key cryptosystem to aim for.

2 NAEP

2.1 Parameter sets

In this section we consider what is meant by an NTRUEncrypt parameter set $\mathcal{P}$. At the end of the section we will discuss how these quantities grow with the security parameter k.

An NTRUEncrypt parameter set $\mathcal{P}$ consists of the following:

- a prime $N \in \mathbb{Z}$, $N = \Theta(k)$.
- a modulus $q \in \mathbb{Z}$, $q = \Theta(k)$.
- a polynomial $p \in \mathbb{Z}[X]$ of degree at most one, and with small coefficients, and invertible when viewed as an element of R_q,
- three public functions:
$$\begin{aligned} \texttt{genf}: &\quad \mathcal{R}_f \to \mathcal{S}_f \\ \texttt{geng}: &\quad \mathcal{R}_g \to \mathcal{S}_g \\ \texttt{genr}: &\quad \mathcal{R}_r \to \mathcal{S}_r. \end{aligned}$$
- three integers $d_f, d_g, d_r \in \mathbb{Z}$,
- a function $\texttt{center} : R_q \to R$.

Here the sets $\mathcal{R}_f, \mathcal{R}_g, \mathcal{R}_r$ are appropriate spaces of randomness, and the sets $\mathcal{S}_f, \mathcal{S}_g, \mathcal{S}_r$ are ranges for the functions. Note that the actual images of the functions may be a subset of these ranges, and the elements in the image do not necessarily occur with exactly the same probability, even when the preimage is chosen uniformly at random. It is assumed that $\mathcal{S}_f, \mathcal{S}_g, \mathcal{S}_r \subset R$, and for all $f \in \mathcal{S}_f$, $g \in \mathcal{S}_g$, $r \in \mathcal{S}_r$ we have $f(1) = d_f$, $g(1) = d_g$, $r(1) = d_r$ respectively.

For NAEP the functions `genf` and `geng` are assumed to output elements of R which are invertible when viewed as elements of R_q. There are no known restrictions on the integer q. For all values of $q \in \mathbb{Z}$ the chance of picking non-invertible elements can be shown to be very low [11]. The efficiency of `genf` and `geng` is therefore not greatly impacted by this invertibility requirement. The output of `genf` should also be invertible when viewed over R_p; this, too, should not impact efficiency [11].

In terms of the security parameter k, the above quantities have the following characteristics: p is always fixed to be the integer 2 or the polynomial $2+X$, the integers $N, q = \Theta(k)$, the running times of the algorithms $\mathtt{genf}, \mathtt{geng}, \mathtt{genr}$ are quasi-linear in k, and $\log|\mathcal{R}_f| = \log|\mathcal{R}_g| = \log|\mathcal{R}_r| = \Theta(k)$.

Further considerations on the choice of genf, geng, genr are given in appendix C. NAEP uses two hash functions:

$$\begin{aligned} G: &\ \{0,1\}^{N-l} \times \{0,1\}^l \to \mathcal{R}_r \\ H: &\ \{0,1\}^N \to \{0,1\}^N \end{aligned}$$

which will be modeled as random oracles in the proof of security. In terms of the security parameter, we wish $l = \Theta(k)$, and also $N - l = \Theta(k)$.

Assumption 1 *We assume that a parameter generation function $\mathcal{G}$ may be found to generate the above quantities, with the specified asymptotics.*

Although no general parameter generation function $\mathcal{G}$ has been described for NTRU, a particular parameter set has been decided on, which we refer to as $\mathcal{P}_0$, where $\langle N, q, d_f, d_g, d_r\rangle = \langle 251, 239, 145, 72, 72\rangle$. The algorithms geng and genr simply pick elements of $\widehat{R}_{d_g}$, $\widehat{R}_{d_r}$ uniformly at random. The algorithm genf picks an element $f' \in \widehat{R}_{72}$ uniformly at random, and then lets $f = 2f' + 1$. More details of this parameter set are given in EESS#1, version 2 [4].

We are now able to introduce the first (of several) assumptions that are important to the proof proof of security, namely

Assumption 2 *There exists a negligible function ν_r such that for sufficiently large k we have that $\epsilon_r \le \nu_r(k)$, where*

$$\epsilon_r \quad = \quad \max_{r \in \mathcal{S}_r} \left\{ \Pr_{\rho \xleftarrow{R} \mathcal{R}_r} [\mathtt{genr}(\rho) = r] \right\}.$$

For the parameter set $\mathcal{P}_0$, we clearly have that $\epsilon_r = \binom{N}{d_r}^{-1} \approx 2^{-212.9}$, though of course we would have to say how d_r varies as a function of k for us to know whether ϵ_r is asymptotically negligible[3].

Assumption 3 *There exists a negligible function ν_{cen} such that for sufficiently large k we have that $\epsilon_{\mathrm{cen}} \le \nu_{\mathrm{cen}}(k)$, where*

$$\epsilon_{\mathrm{cen}} \quad = \quad \Pr[\mathtt{center}(f * \hat{w} + p * r * g \bmod q) \neq f * \hat{w} + p * r * g],$$

where $f = \mathtt{genf}(\rho_f)$, $g = \mathtt{geng}(\rho_g)$, $r = \mathtt{genr}(\rho_r)$ and the probability is defined over $\rho_f \xleftarrow{R} \mathcal{R}_f, \rho_g \xleftarrow{R} \mathcal{R}_g, \rho_r \xleftarrow{R} \mathcal{R}_r, \hat{w} \xleftarrow{R} \widetilde{R}$.

The role of center is to take the modular quantity $(f * \hat{w} + p * r * g \bmod q)$ and return the non-modular quantity $(f * \hat{w} + p * r * g)$. Appendix A gives two (similar) techniques for doing this, center1 and center2, both of which require $\hat{w} \in \widetilde{R}$ and d_f, d_g, d_r to be known.

It is far harder to approximate the quantity ϵ_{cen} than ϵ_r because it depends on a particular centering algorithm. The centering technique center1 will only fail to recover the non-modular quantity if the spread of coefficients of $f * \hat{w} + p * r * g$ is at least q. The centering technique center2 will fail to recover the non-modular quantity only if one or more coefficients differ from $N/2$ by

[3] And as mentioned above a general parameter generation function $\mathcal{G}$ has not been specified to date, though if $d_r \approx N/3$, then ϵ_r would certainly be negligible.

more than $q/2$. The probability of a failure for either algorithm can be estimated theoretically under some simplifying assumptions which appear to match experimental results well [19]. If no rigorous bound can be proved, then good evidence should be supplied, and the above assumption should be clearly made, since this property strongly impacts the proof of security of NAEP.

For the parameter set $\mathcal{P}_0$ it is estimated that for `center2` $\epsilon_{\text{cen}} \approx 2^{-121.7}$, and for `center1` $\epsilon_{\text{cen}} \approx 2^{-156}$ [19].

2.2 Key generation

Key generation is specified by:

1. Pick $\rho \stackrel{R}{\leftarrow} \mathcal{R}_f$ and let $f = \mathtt{genf}(\rho)$. Let f_q^{-1} denote the inverse of f in R_q, and let f_p^{-1} denote the inverse of f in R_p.

2. Pick $\rho' \stackrel{R}{\leftarrow} \mathcal{R}_g$ and let $g = \mathtt{geng}(\rho')$. Note g should be invertible in R_q.

3. Let $h = g * f_q^{-1} \in R_q$.

The public key is $h \in R_q$ and the above global parameters, while the private key is $f \in \mathcal{S}_f \subset R$, $f_p^{-1} \in R_p$.

Note that the quantity f_p^{-1} need not be part of the explicit output of key generation if `genf` is designed such that f_p^{-1} is always a constant. For example, if $f = 1 + p * f'$ as in the parameter set $\mathcal{P}_0$, then f_p^{-1} will always be 1.

2.3 The NTRU Hard Problem and One-Way Function

The one-way function underlying NTRU is:

$$\begin{array}{rcl} F : \widetilde{R} \times \mathcal{S}_r & \rightarrow & R_q \\ F(\hat{w}, r) & = & \hat{w} + r * p * h, \end{array}$$

where $q, N \in \mathbb{Z}$, $p \in \mathbb{Z}[X]$, $h \in R_q$ are given by the output of key generation. We note that $\mathcal{G}(1^k)$ is assumed to yield parameters, for which there is a algorithm to compute F in time polynomial in k (indeed using fast multiplication techniques there is an algorithm quasi-linear in N, and hence k).

Definition 3 *(The $\mathcal{P}$-NTRU problem) For a parameter set $\mathcal{P}$, we denote by $\mathsf{Succ}^{\mathsf{ow}}_{\mathsf{ntru}}(\mathcal{A}, \mathcal{P})$ the success probability of any adversary $\mathcal{A}$ for finding a preimage of F,*

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

Assumption 4 *(The $\mathcal{G}$-NTRU assumption) For every probabalistic polynomial (in k) time algorithm $\mathcal{A}$ there exists a negligible function $\nu_{\mathcal{A}}$ such that for sufficiently large k, we have $\mathsf{Succ}^{\mathsf{ow}}_{\mathsf{ntru}}(\mathcal{A}, \mathcal{P}) \leq \nu_{\mathcal{A}}(k)$.*

In appendix B we show an equivalence between the $\mathcal{P}$-NTRU problem and a particular kind of module CVP instance. Although this equivalence is not essential for the proof of security it may give the reader some insight in to the properties of the NTRU one-way function. Indeed the best known attacks against NTRUEncrypt [5, 14] consider the module structure as a lattice, and use lattice reduction techniques to recover $\hat{w}$ and r.

Notice that by talking about preimages rather than an inverse of F, we do not necessarily assume that $\hat{w}, r$ are uniquely defined by c. This point is discussed further in section 2.5.

2.4 Encryption

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

$$\texttt{compress}(x) = (x \bmod q) \bmod 2,$$

that is, the coefficients of the modular quantity $x \bmod q$ are put in to the interval $[0, q)$, and then this quantity is reduced modulo 2. If q is even, then this is simply equivalent to $x \bmod 2$, though we do not restrict our parameter choices to ones where q is even (e.g. see $\mathcal{P}_0$). The role of `compress` is to just reduce the size of the input to the hash function H for gains in practical efficiency[4]. The encryption algorithm is then specified by:

1. Pick $\mu \xleftarrow{R} \{0,1\}^l$.
2. Let $\rho = G(M, \mu)$, $r = \texttt{genr}(\rho)$, $\hat{s} = \texttt{compress}(p * r * h)$, and $\check{w} = (M||\mu) \oplus H(\check{s})$.
3. If $\hat{w} \notin \widetilde{R}$, go to step 1.
4. Let $c = F(\hat{w}, r) \in R_q$.

In practice, with the parameter set $\mathcal{P}_0$, the chance of $\hat{w} \notin \widetilde{R}$ at step 3 is approximately 2^{-207}; see [19] for more details. This means that in practice this test is not implemented. The test has no impact on the proof of security, even if the probability is considerably more than 2^{-80} say, but if it does fail, then the decryption algorithm is guaranteed to fail to decrypt the ciphertext c. We will denote this algorithm $c \leftarrow \mathcal{E}(M, \mu)$.

2.5 Decryption

To decrypt a message $c \in R_q$ one does the following:

1. Let $a = \texttt{center}(f * c \bmod q)$. With very high probability `center` is such that we will have $a = f * \hat{w} + p * r * g$, although there is a small probability ϵ_{cen} that this step will fail to recover this correct a.
2. If $f = 1 + p * f'$ (as in the parameter set $\mathcal{P}_0$) then simply let $\hat{w} = a \bmod p$, otherwise let $\hat{w} = f_p^{-1} * a \bmod p$.
3. Let $s' = c - \hat{w}$.
4. Let $\hat{s} = \texttt{compress}(s')$.
5. Let $M||\mu = \check{w} \oplus H(\check{s})$, $r = \texttt{genr}(G(M, \mu))$.
6. If $p * r * h = s' \bmod q$, and $\hat{w} \in \widetilde{R}$, then return the message M, else return the string "invalid ciphertext".

Since there is a possibility of failing to invert the function F, we call the NTRU family of functions, a *pseudo-trapdoor one-way function family.*

We note that with the NTRU one-way function we also cannot rule out the possibility that there are two pairs $(\hat{w}_1, r_1), (\hat{w}_2, r_2) \in \widetilde{R} \times \mathcal{S}_r$, such that

$$p * r_1 * h + \hat{w}_1 = p * r_2 * h + \hat{w}_2 = c \bmod q,$$

[4]The proof of security would exactly work in the same way if `compress` was the identity function.

in which case $F^{-1}(c)$ is not well defined mathematically. However we can of course define it to be the $(\hat{w}, r)$ retrieved by performing the above NTRU "inverse" procedure. By the fact that this process is successful whenever `center` is successful, we observe that "any other" solutions would fail the centering process, which happens with small probability ϵ_{cen}.

We note that allowing for the case of key and message dependent decryption failures does not fall in to the classical notion of a public key encryption scheme, as discussed briefly in [17]. In this report we do not try to fully classify the theoretical properties of such schemes, but instead we argue that the above NAEP scheme is indistinguishable from one that always succeeds to decrypt, if one can ensure various explicitly stated probabilites are sufficiently small.

Such probabilities are not normally considered in standard CCA2 proofs of security, and so we must be careful to do this correctly. The main subtlety here is that since the failures are message dependent, we must calculate the chance of decryption failures, given that the messages may be adversarially chosen.

In section 3 we show that by choosing the above padding scheme, the adversary's power in choosing M does not translate to a power in choosing $\hat{w}$ or r, and thus we can prove security using the following average case probability ϵ_{cen}, rather than have to consider a worst case probability. This is a huge benefit, since the worst case probabilities involved in NTRU, would not yield a useful proof of security, as explained in [12, 17]. Indeed this is exactly the reason that unaltered f-SAEP$^+$ cannot be used for a padding scheme for NTRU, since in this scheme r can be adversarially chosen.

3 On the security of NAEP

The proof of security for NAEP is very similar to that given for PSS-E (although unfortunately our G and H correspond to their H and G, respectively). It is standard practice in cryptography to define adversarial advantage in proofs of security over all possible randomness, including that used by the key generation algorithm. Although this is fine is an asymptotic sense, it does not prevent the existence of weaker keys than average, which may affect security in practice. This point is discussed further in section 3.1.

Theorem 1 *Let $\mathcal{A}$ be a (t, ϵ) chosen ciphertext algorithm against* NAEP *in the random oracle model, which makes at at most q_G, q_H, q_D queries to the random oracles G and H, and decryption oracle $\mathcal{D}$ respectively. Then there is an algorithm $\mathcal{B}$ for solving the $\mathcal{P}$-*NTRU *problem with the following parameters*

$$\begin{aligned} \text{time}(\mathcal{B}) &= \text{time}(\mathcal{A}) + O((q_G + q_H)t_q + q_{\mathcal{D}}) \\ \mathsf{Succ}^{\mathsf{ow}}_{\mathsf{ntru}}(\mathcal{B}, \mathcal{P}) &= p_c(\epsilon - q_G(2^{-l} + \epsilon_{\text{cen}}) - q_{\mathcal{D}}\epsilon_r), \end{aligned}$$

where the probabilites ϵ_r and ϵ_{cen} are defined as above, and t_q is the time to perform a (binary[5]) modulo q convolution product (which is assumed to dominate the other steps, e.g. `genr`, `compress`*). The value p_c is either the constant 1, if q is even, or if q is odd, it is the probability that the inversion challenge c^* has no zero coefficients modulo q, i.e.*

$$p_c = \Pr_{\hat{w} \xleftarrow{R} \widetilde{R}, \rho \xleftarrow{R} \mathcal{R}_r} [c_i^* \neq 0, \ \ 0 \leq i < N \mid c^* = F(\hat{w}, \mathtt{genr}(\rho))].$$

[5]When one of the multiplicands is binary, a convolution product is simply equivalent to some number of rotations and additions modulo q, and can be very efficiently implemented.

Proof: Algorithm $\mathcal{B}$ is given $c^* = F(\hat{w}^*, r^*)$ for some $\hat{w}^* \overset{R}{\leftarrow} \widetilde{R}$, $r^* = \mathtt{genr}(\rho)$ where $\rho \overset{R}{\leftarrow} \mathcal{R}_r$. $\mathcal{B}$'s goal is to find these $\hat{w}^*, r^*$, or any other solution from the space $\widetilde{R} \times \mathcal{S}_r$, i.e. find a preimage of the one-way function F.

$\mathcal{B}$ will do this by interacting correctly with $\mathcal{A}_1$, obtaining two messages M_0 and M_1, and then asking $\mathcal{A}_2$ to distinguish c^*.

Conceptually $\mathcal{B}$ chooses values $\mu^* \overset{R}{\leftarrow} \{0,1\}^l$, and $M^* = M_{b^*}$, $b^* \overset{R}{\leftarrow} \{0,1\}$, independent of $\mathcal{A}$'s view, and asserts[6] $G(M^*||\mu^*) = r^*$ and $H(\check{s}^*) = \hat{w}^* \oplus (M^*||\mu^*)$. Let Win denote the event that $b \leftarrow \mathcal{A}_2$ is such that $b = b^*$.

Let AskG denote the event that $\mathcal{A}$ queries G at $(M_0||\mu^*)$, or $(M_1||\mu^*)$, and let AskH denote the event that $\mathcal{A}$ queries H at $\check{s}^*$. Note that if $\neg\mathsf{AskH}$ then $\mathcal{A}$ has no information about μ^*, so

$$\Pr[\mathsf{AskG} \mid \neg\mathsf{AskH}] \;\leq\; \frac{q_G}{2^l}.$$

Also, if $\mathsf{AskGorH}$ denotes the event $\mathsf{AskG} \vee \mathsf{AskH}$, then notice that if $\neg\mathsf{AskGorH}$ then $\mathcal{A}$ has no information about b^*, i.e.

$$\Pr[\mathsf{Win} \mid \neg\mathsf{AskGorH}] \;=\; \frac{1}{2}.$$

We build $\mathcal{B}$ on the idea that the event AskG is unlikely, and if the event AskH happens, we can invert the function F. Explicitly for each query $\check{s}_i$ to H, $\mathcal{B}$ computes $\hat{w} = \mathtt{compress}(c^* - \hat{s})$, $r = \mathtt{genr}(G(H(\check{s}) \oplus \hat{w}))$ and checks if $F(\hat{w}, r) = c^*$. If so, $\mathcal{B}$ has successfully broken the $\mathcal{P}$-NTRU problem.

However this H simulation is only guaranteed to catch the event AskH if $\mathtt{compress}(c^* - \hat{s}^*) = \hat{w}^*$ when $\hat{s}^* = \mathtt{compress}(c^* - \hat{w}^*)$, and if q is odd[7] this is only true if all of the coefficients of c^* are non-zero modulo q. Since the challenge c^* is chosen when $\hat{w}^* \overset{R}{\leftarrow} \widetilde{R}$, and $r = \mathtt{genr}(\rho)$, $\rho \overset{R}{\leftarrow} \mathcal{R}_r$, and under the reasonable assumption that the coefficients of such c^* are randomly distributed modulo q, then this will happen with probability $p_c \leq (1 - 1/q)^N \approx 0.35$ with the parameter set $\mathcal{P}_0$. If asymptotically $q = \Theta(N)$ then this probability is non-negligible, and therefore will not affect the proof of security.

If one wants to increase $\mathsf{Succ}^{\mathsf{ow}}_{\mathsf{ntru}}(\mathcal{B}, \mathcal{P})$ at the expense of a slightly less efficient reduction, then since one knows the locations $\{l_1, \ldots, l_d\}$ of the zero coefficients of c^*, then for each H-query one can try each of the 2^d possible $\hat{w}$'s with the same coefficients as $\mathtt{compress}(c^* - \hat{s}^*)$ in the $N - d$ non-zero places. Such an approach will always detect the event AskH (i.e. increase p_c to 1), at the expense of making the H-simulation take an expected $2^{N/q}$ times longer. Again if $q = \Theta(N)$ then this will not asymptotically affect the proof of security.

Finally we show that we can successfully simulate a decryption oracle $\mathcal{D}$, without knowing the private key, up to the point at which $\mathcal{A}_2$ makes a $\check{s}^*$ query or a decryption failure occurs.

In such a manner we will construct an algorithm $\mathcal{B}$ which can solve the $\mathcal{P}$-NTRU problem, without any trapdoor information, with the specified time and advantage characteristics given above.

Simulating the oracles — The random oracles G and H are simulated just as in $\mathsf{PSS\text{-}E}$, i.e. their output is chosen completely at random if not previously called, and logs of the input and output queries are kept to maintain this functionality. Explicitly H_{list} is composed of just

[6] Without actually knowing $\check{s}^*$.

[7] If q is even it is always true.

pairs of the form $\langle z, H(z)\rangle$, while G_{list} keeps tuples of the form $\langle M, \mu, \rho, c\rangle$ for each query of the form $(M, \mu) \in \{0,1\}^{N-l} \times \{0,1\}^l$, where $\rho = G(M, \mu)$, $r = \texttt{genr}(\rho)$, $\hat{s} = \texttt{compress}(p * r * h)$, $\check{w} = (M||\mu) \oplus H(\check{s})$ and $c = \hat{w} + p * r * h \bmod q$, which one should notice entails a call to the H oracle (but there is no point checking for the AskH event on such calls). The quantity c is only placed in to the H_{list} entry if $\hat{w} \in \widetilde{R}$.

The decryption oracle is simulated in the following fashion: when called on input c, we check if $\langle M, -, -, c\rangle \in G_{list}$, for some $M \in \mathcal{M}$. If so, we return M; if not, we return "invalid ciphertext". If there is more than one such M the simulation may fail, but we will argue that the probability of this is very low.

Analysis — Let $\Pr_{real}[X]$ and $\Pr_{sim}[X]$ denote the probability of an event X occuring when $\mathcal{A}$ has access to a real or simulated decryption oracle respectively. We firstly explain[8] that $\Pr_{real}[\mathsf{AskH}] \geq \epsilon - q_G 2^{-l}$, where ϵ is the advantage of the algorithm $\mathcal{A}$, i.e. the quantity $2\Pr_{real}[\mathsf{Win}] - 1$.

We have that

$$\begin{aligned}
\Pr_{real}[\mathsf{Win}] &= \Pr_{real}[\mathsf{Win} \mid \neg\mathsf{AskGorH}] \times \Pr_{real}[\neg\mathsf{AskGorH}] + \Pr_{real}[\mathsf{Win} \mid \mathsf{AskGorH}] \times \Pr_{real}[\mathsf{AskGorH}] \\
&= \frac{1}{2} \times (1 - \Pr_{real}[\mathsf{AskGorH}]) + \Pr_{real}[\mathsf{Win} \mid \mathsf{AskGorH}] \times \Pr_{real}[\mathsf{AskGorH}] \\
&\leq \frac{1}{2}(1 + \Pr_{real}[\mathsf{AskGorH}]),
\end{aligned}$$

so

$$\epsilon \leq \Pr_{real}[\mathsf{AskGorH}] = \Pr_{real}[\mathsf{AskG} \wedge \neg\mathsf{AskH}] + \Pr_{real}[\mathsf{AskH}] \leq q_G 2^{-l} + \Pr_{real}[\mathsf{AskH}].$$

We now show that with high probability $\mathcal{A} \equiv (\mathcal{A}_1, \mathcal{A}_2)$ cannot distinguish the real decryption oracle from the simulated one until it issues an $\check{s}^*$ query. Let $\mathsf{GoodSim}$ denote the following event(s):

1. The simulator never rejects a valid decryption query issued by $\mathcal{A}$ that the true decryption function would have decrypted successfully, and
2. The simulator never successfully decrypts a valid encryption that the true decryption function would have rejected (possible since decryption failures exist), and
3. The simulator never decrypts a valid decryption query to a different message than the true decryption function (possible since we cannot prove message unambiguity).

If $\mathsf{GoodSim}$ occurs, then the simulated decryption oracle behaves exactly as the real one does, so we have:

$$\begin{aligned}
\Pr_{sim}[\mathsf{AskH}] &\geq \Pr_{sim}[\mathsf{AskH}|\mathsf{GoodSim}] \cdot \Pr[\mathsf{GoodSim}] \\
&= \Pr_{real}[\mathsf{AskH}] \cdot \Pr[\mathsf{GoodSim}] \\
&\geq \left(\epsilon - q_G 2^{-l}\right) \Pr[\mathsf{GoodSim}].
\end{aligned}$$

In addition we will show that

$$\Pr[\mathsf{GoodSim}] \geq 1 - q_G \epsilon_{\text{cen}} - q_{\mathcal{D}} \epsilon_r,$$

which will confirm that the algorithm $\mathcal{B}$ has the running time and advantage specified above.

[8] We note that this probability was incorrectly bounded by ϵ in the proof of f-SAEP^+.

We firstly bound the probability of event 1. Let $c \neq c^*$ be a decryption query issued by the attacker and rejected by the simulator. This c is only a valid ciphertext if $c = F(\hat{w}, r)$ where $M||\mu = c \oplus H(\hat{s})$,$r = \mathtt{genr}(G(M,\mu))$ and $\hat{s} = \mathtt{compress}(p * r * h)$. Since c is rejected by the simulator we know $\mathcal{A}$ did not issue a query for $G(M,\mu)$, so its value is independent of the adversary's view. Thus the probability that a decryption query is correctly rejected is at least $1 - \epsilon_r$, and the probability that up to $q_{\mathcal{D}}$ queries are correctly rejected is at least $1 - q_{\mathcal{D}}\epsilon_r$.

We may bound the probability of events 2 and 3 at the same time. Let GoodG be the event that at the end of the simulation all the entries of G_{list} are such that the $\hat{w}$ and r corresponding to $\langle M, \mu, \rho, c\rangle$ are such that

$$\mathtt{center}(f * \hat{w} + p * r * g \bmod q) \quad = \quad f * \hat{w} + p * r * g,$$

i.e. the true decryption oracle would correctly decrypt all of the ciphertexts c.

The occurance of event GoodG clearly implies that event 2 occurs. It also implies event 3 occurs because the only way for the simulated decryption oracle to return a different message M, is if the same c occurs as the last entry of one of the entries of G_{list}. However in this case, the true decryption oracle clearly cannot successfully decrypt both of these, which implies $\neg\mathsf{GoodG}$.

Since $\hat{w}$ and r are the output of random oracles, as h varies over key generation, the chance of any one entry of G_{list} corresponding to a centering failure is exactly ϵ_{cen}. Since the adversary makes at most q_G queries to the G oracle, we have that $\Pr[\mathsf{GoodG}] \geq (1 - q_G\epsilon_{\text{cen}})$. Together with the bound on event 1 this allows us to bound $\Pr[\mathsf{GoodSim}]$ as above. □

Corollary 1 *Under assumptions 1,2,3 and 4 there is no polynomial (in k) time algorithm that breaks the chosen ciphertext security of* NAEP*, that makes a polynomial (in k) number of queries to the random oracles G and H, and decryption oracle $\mathcal{D}$.*

3.1 The meaning of proofs of security

It would be nice if a proof of security gave one the assurance that for a given parameter set, e.g. RSA-1024 or NTRU-251, all keys associated with that parameter set were secure. Unfortunately this is not the case. The definition of a (trapdoor) one-way function is an asymptotic one, so a proof of security merely means that legitimate parties can work arbitrarily faster than a chosen ciphertext adversary, if one uses a suitably large security parameter $k > K_0$ (assuming the underlying function is indeed one-way).

In a more positive light, a proof of security in the random oracle model, often shows an efficient reduction from a chosen ciphertext adversary who breaks the semantic security on a *particular key* to an algorithm which inverts the one-way function with *that* key. Our proof shows a similar result, but only when ϵ_{cen} is sufficiently low, even when restricted to a given key h. This is still a useful statement to make, especially if one cannot tell which keys h are likely to result in more gaps. This point is discussed further below.

In practice we are more interested in the (non)-existence of *efficient* algorithms. To quantify this notion, we define the *cost* of a probabilistic algorithm $\mathcal{A}$ to be the quantity tm/ϵ, if $\mathcal{A}$ runs in time t, using memory m, and succeeds with probability ϵ. We may say that an algorithm is efficient if its cost is below some threshold.

This might suggest changing the practical definition of one-way function, to something like the following:

Assumption 5 *There is no efficient algorithm $\mathcal{A}$ which inverts the* NTRU *one-way function, when h is drawn from* NTRUEncrypt *key generation.*

and then trying to use the same proof technique to show that an efficient chosen ciphertext adversary contradicts this assumption. Unfortunately this assumption is not strong enough to achieve a contradiction, since it does not disallow the existence of weak keys.

Here an illustration might be helpful. Suppose we believe that factoring is the best way to break the RSA-assumption, and we know of an algorithm with expected cost C to factor $N = pq$ when the primes p and q are randomly chosen by the RSA key generation procedure. However, what if there was a small class of moduli which can be factored at considerably lower cost? Specifically, assume that with probability P_1 a modulus N created by the RSA generation procedure falls in to a class of moduli which can be factored by an algorithm $\mathcal{A}_{\text{fac}}$ with cost αC, $\alpha < 1$, and that in addition there is an algorithm $\mathcal{A}_{\text{dist}}$ with cost less than CP_1 which will identify such moduli[9]. An adversarial strategy to factor would be to test many keys until a weak one was found (this would be expected after $O(P_1^{-1})$ trials and cost $< C$), and then run the cost-αC algorithm to factor this key. Therefore, even though the expected cost to break a random key is C, one could not claim that there was no adversary with cost less than C.

A similar attack can be launched against any scheme in which the keys are not provably equivalent to each other, e.g. NTRUEncrypt. For this reason we need to strengthen the above assumption to the following (for some suitably chosen bound C_0).

Assumption 6 *There is no cost-C_0 algorithm $\mathcal{A}_1$ and cost-C_0P predicate $\mathcal{A}_2$ such that $\mathcal{A}_1$ can invert the* NTRU *one-way function, when h is drawn from* NTRU *key generation, subject to the constraint $\mathcal{A}_2(h) = 1$, and randomly generated keys satisfy this constraint with probability P.*

In the case of NTRUEncrypt, the proof of security also relies on the fact that there are no decryption failures. Thus we also need the following assumption, for some suitable choice of allowable gap failure probability P_0.

Assumption 7 *There is no cost-C_0P predicate $\mathcal{A}_2$, where $P = \Pr_h[\mathcal{A}_2(h) = 1]$, and*

$$\Pr_{\hat{w},r,h}\left[\mathtt{center}(f * \hat{w} + p * r * g \bmod q) \neq f * \hat{w} + p * r * g \mid \mathcal{A}_2(h) = 1\right] > P_0,$$

where $\hat{w} \xleftarrow{R} \widetilde{R}$, $r = \mathtt{genr}(\rho)$, $\rho \xleftarrow{R} \mathcal{R}_r$, and $\mathsf{pk} = h$, $\kappa \xleftarrow{R} \mathcal{R}_{\mathcal{K}}$.

For example it might be appropriate to take $P_0 = 2^{-l}$ where $l = \Theta(k)$ is defined as coming from $\mathcal{G}(1^k)$. In any practical instantiation of NTRUEncrypt, there ought to be some discussion as to why these are reasonable assumptions to make.

The use of the random oracle assumption is also a slightly controversial topic in cryptography, however we note that a proof in the random oracle model does have meaning in practice: it means that a chosen ciphertext attacker cannot treat the oracles as a black box[10], but must rely on some properties of the hash functions. When the oracles are instantiated with `SHA-1` say, it seems unlikely that an attacker can find useful properties.

[9] We assume, although it is not necessary for the purposes of this illustration, that $\mathcal{A}_{\text{dist}}$ is quite costly to run; otherwise, one could simply run $\mathcal{A}_{\text{dist}}$ at key generation time and reject the keys it identifies

[10] One can imagine computer code making explicit calls like `a=hashFn(b)`.

3.2 Further enhancements to NAEP

A sound engineering principle is to make the hash functions dependent on the public key. This is extremely important for schemes which possess decryption failures because an adversary may be willing to put in a huge amount of effort to find a set of weak $r = G(M, \mu)$ if this breaks a lot of systems. Note that we state this as a heuristic principle, without defining an explicit attack model.

Another sound engineering principle is to make the hash functions dependent on the parameter set, so an attacker is unlikely to be able to mount attacks utilising different variants of NTRUEncrypt.

One way to make these modifications is to let

$$r = \mathtt{genr}(G(b, M, \mu, \mathrm{trunc}(h)),$$

where M and μ are the message and randomness as before, b is an identification number, unique for each parameter set, and trunc() is a suitable compression of h to make key-collision very unlikely. Specifically when $N = 251$ and $q = 239$, and G is instantiated with SHA-1, we might let b take 3 bytes, and let $\mathrm{trunc}(h)$ be the first 10 coefficients[11] of h (each represented as 1 byte). This is the approach taken in the instantiation of NAEP, known as SVES-3, specified in [4].

4 Conclusions

We have explained that considerable care must be taken with using standard, provably secure, CCA2-secure encryption scheme, when instantiating them with underlying schemes which are vulnerable to (key and message dependent) decryption failures. This is true even if the probability of the pseudo-trapdoor failing is negligible on random input (it is the most adversarially advantageous induced distribution from the padding scheme that is important).

We built an NTRUEncrypt padding scheme called NAEP, for which we were able to prove the security, under a number of assumptions. To our knowledge this is the first proof of security that addresses the question of decryption failures, and highlights the complexities.

For any given NTRU parameter set $\mathcal{P}$ it is very important that these assumptions are analyzed extensively. If the mathematical probabilities cannot be rigorously proved to have the required bounds, then the necessary statements must be added as assumptions to the proof of security.

This report deals only with the *proof* of security, and does not attempt to justify that the combination of any parameter sets and algorithms do satisfy these properties. However it is extremely important to have this foundation, and the necessary assumptions made explicit, so that the security of NTRU parameter sets can be analyzed in a rigorous fashion.

Acknowledgements

We thank Phong Nguyen and David Pointcheval for several useful comments.

References

[1] M. Bellare and P. Rogaway. Optimal asymmetric encryption. In Proc. of Eurocrypt '94, volume 950 of LNCS, pages 92–111. IACR, SpringerVerlag, 1995.

[11] Other choices may be considered if one wishes to protect against malicious key providers.

[2] D. Boneh, Simplified OAEP for the RSA and Rabin functions, In proceedings of Crypto '2001, Lecture Notes in Computer Science, Vol. 2139, Springer-Verlag, pp. 275-291, 2001

[3] Consortium for Efficient Embedded Security, *Efficient Embedded Security Standard #1*, Version 1, available from `http://www.ceesstandards.org`.

[4] Consortium for Efficient Embedded Security, *Efficient Embedded Security Standard #1*, Version 2, May 2003, available from `http://www.ceesstandards.org`.

[5] D. Coppersmith and A. Shamir, *Lattice Attack on NTRU*, Advances in Cryptology - Eurocrypt'97, Springer-Verlag

[6] J-S. Coron, M. Joye, D. Naccache and P. Paillier Universal padding schemes for RSA In Advances in Cryptology - Crypto '02 LNCS 2442, pages 226-241. Springer-Verlag, Berlin, 2002.

[7] E. Fujisaki and T. Okamoto. How to enhance the security of publickey encryption at minimum cost. In Proc. of PKC '99, LNCS 1560, pages 53–68. SpringerVerlag, Berlin, 1999.

[8] J. Hoffstein, J. Pipher, J.H. Silverman, *NTRU: A new high speed public key cryptosystem*, in Algorithmic Number Theory (ANTS III), Portland, OR, June 1998, Lecture Notes in Computer Science 1423 (J.P. Buhler, ed.), Springer-Verlag, Berlin, 1998, 267–288. See also `http://www.ntru.com`.

[9] J. Hoffstein and J. H. Silverman. Optimizations for NTRU. In Publickey Cryptography and Computational Number Theory. DeGruyter, 2000. To appear, available at [4].

[10] J. Hoffstein and J. H. Silverman, Random Small Hamming Weight Products With Applications To Cryptography, Discrete Applied Mathematics, to appear, Available from `http://www.ntru.com`.

[11] J. Hoffstein and J. H. Silverman. Invertibility in truncated polynomial rings. Technical report, NTRU Cryptosystems, October 1998. Report #009, version 1, available at `http://www.ntru.com`.

[12] N. Howgrave-Graham, P. Q. Nguyen, D. Pointcheval, A. Singer and W. Whyte. Padding schemes and decryption failures in NTRU Encryption. January 2003 (undergoing submission)

[13] E. Jaulmes and A. Joux. A chosen ciphertext attack on NTRU. In Proc. of Crypto '00, volume 1880 of LNCS. SpringerVerlag, 2000.

[14] A. May, J.H. Silverman, *Dimension reduction methods for convolution modular lattices*, Cryptography and Lattices Conference (CaLC 2001), Volume 2146 of LNCS, Springer-Verlag, 2001

[15] T. Meskanen, A. Renvall, *Wrap Error Attack Against NTRUEncrypt*, Preprint, November 27, 2002.

[16] P. Nguyen, D. Pointcheval, 'Analysis and Improvements of NTRU Encryption Paddings', M. Yung (Ed), Proceedings of Crypto 2002, LNCS 2442, pp. 210-225, Springer-Verlag, 2002

[17] J. Proos *Imperfect Decryption and an Attack on the NTRU Encryption Scheme*, ePrint archive, January 2003.

[18] N. Howgrave-Graham, J. H. Silverman, W. Whyte, A Meet-in-the-Middle Attack on an NTRU Private key, Technical report, NTRU Cryptosystems, May 2003. Report #004, version 2, available at `http://www.ntru.com`.

[19] J. H. Silverman, W. Whyte, Estimating decryption failure probabilities for NTRUEncrypt, Technical report, NTRU Cryptosystems, May 2003. Report #018, version 1, available at `http://www.ntru.com`.

A Centering methods

We describe here two possible centering methods, that is, an algorithm `center` which when given the modular quantity $A = f*\hat{w}+p*r*g \bmod q$ will return the non-modular quantity $a = f*\hat{w}+p*r*g$. We'll assume the coefficients of A are given in the range $[0, q)$.

`center1`:

1. Let $I_1 = (A(1) - pd_r d_g)/(d_f^{-1}) \bmod q$ so $\hat{w}(1) = I_1 \bmod q$.
2. Let $I_2 \in [(N-q)/2, (N+q)/2)$ be such that $I_2 = I_1 \bmod q$. Since $\hat{w} \in \widetilde{R}$ we know $\hat{w}(1) = I_2$ exactly (not just modulo q).
3. Let a be a non-modular polynomial, such that $a = A \bmod q$ and the coefficients of a are in the range $[0, q)$.
4. Let $J = (d_f I_2 + pd_r d_g - I_1)/q$, so know $J = (a(1) - A(1))/q$.
5. Let $J = J_0 + J_1 q$ with J_0 in the range $[0, q)$.
6. Add $J_1 q$ to all the coefficients of a.
7. Add q to the least J_0 coefficients of a.

This centering method will always recover $f*\hat{w}+p*r*g$ exactly, unless the spread of coefficients of $f*\hat{w}+p*r*g$ is at least q. See [19] for more details.

If it is considered slightly costly to calculate the least J coefficients of A, then the following is a faster option.

`center2`:

1. Let $I_1 = (A(1) - pd_r d_g)/(d_f^{-1}) \bmod q$ so $\hat{w}(1) = I_1 \bmod q$.
2. Let $I_2 \in [(N-q)/2, (N+q)/2)$ be such that $I_2 = I_1 \bmod q$. Since $\hat{w} \in \widetilde{R}$ we know $\hat{w}(1) = I_2$ exactly (not just modulo q).
3. Let a be a non-modular polynomial, such that $a = A \bmod q$ and the coefficients of a are in the range $[0, q)$.
4. Let $J = d_f I_2 + pd_r d_g$, so we know $J = a(1)$.
5. Shift all the coefficients of a by multiples of q in to the range $[J/N - q/2, J/N + q/2)$.

This centering method will fail to recover the quantity $f*\hat{w}+p*r*g$ more often than the first algorithm[12], but the chance of failure may hopefully still be made small enough.

[12] Although it is possible to correct in a similar way to the first if the resulting $a(1) \neq J$.

B The $\mathcal{P}$-NTRU problem viewed as a module problem

Definition 4 *Let $\mathcal{L}$ be a rank 2 R-module generated by the basis $\{(1, p*h), (0, q)\}$ where h is the public output of* NTRU *key generation. A time t probabilistic algorithm $\mathcal{A}$ is said to solve the $\mathcal{P}$-*NTRU *problem with advantage ϵ, if given a point $(0, y) \in R_q^2$, and the knowledge that there is a module point of the form $\mathbf{v} = (e_1, y - e_2)$ where $e_1 = \mathtt{genr}(\rho)$ for some $\rho \xleftarrow{R} \mathcal{R}_r$ and $e_2 \xleftarrow{R} \widetilde{R}$, then $\mathcal{A}$ has probability ϵ of finding such a $\mathbf{v}$; determined by whether $(e_1, e_2) \in \mathcal{S}_r \times \widetilde{R}$. This probability is defined over the choice of $y \xleftarrow{R} R_q$ subject to the constraint of $(0, y)$ lying $(e_1, -e_2)$ away from a module point, and the randomness used in key generation.*

Assumption 8 *The $\mathcal{P}$-*NTRU *assumption is that there is no efficient algorithm to solve the $\mathcal{P}$-*NTRU *problem.*

We note that the hardness of this problem clearly requires the hardness of the NTRU key recovery problem, i.e. recovering f and g from $h \in R_q$, which also has a natural CVP/SVP module setting (depending on the form of `genf` and `geng`).

Lemma 1 *Finding a preimage of the* NTRU *one-way function*

$$\begin{aligned} F : \widetilde{R} \times \mathcal{S}_r &\rightarrow R_q \\ F(\hat{w}, r) &= \hat{w} + r * p * h, \end{aligned}$$

when $\hat{w} \xleftarrow{R} \widetilde{R}$, $r = \mathtt{genr}(\rho)$ for $\rho \xleftarrow{R} \mathcal{R}_r$, and h is the output of NTRU *key generation, is equivalent to solving the $\mathcal{P}$-*NTRU *problem.*

Proof: Suppose we have access to an algorithm $\mathcal{A}$ which can find a preimage of F with probability ϵ when $\hat{w} \xleftarrow{R} \widetilde{R}$ and $r = \mathtt{genr}(\rho)$ for some $\rho \xleftarrow{R} \mathcal{R}_r$. If we are given a point $y \in R_q$ such that $\mathbf{v} = (e_1, y - e_2) \in \mathcal{L}$ where $e_1 = \mathtt{genr}(\rho)$ for some $\rho \xleftarrow{R} \mathcal{R}_r$, $e_2 \xleftarrow{R} \widetilde{R}$, then $y - e_2 = e_1 * p * h \bmod q$, i.e. $y = F(e_2, e_1)$. Thus with probability ϵ we will have that $\mathcal{A}(y) = (e_2, e_1)$, which will reveal the close module point $\mathbf{v}$.

Conversely suppose that we have an algorithm $\mathcal{A}'$ which, given $y \in R_q$ can find (with probability ϵ) close module points of the form $(e_1, y - e_2) \in \mathcal{L}$ where $e_1 = \mathtt{genr}(\rho)$ for some $\rho \xleftarrow{R} \mathcal{R}_r$, $e_2 \xleftarrow{R} \widetilde{R}$. If we are given $y = F(\hat{w}, r)$, then notice

$$\begin{aligned} r * (1, p * h) + k(0, q) &= (r, r * p * h \bmod q) \\ &= (r, -\hat{w}) + (0, F(\hat{w}, r)), \end{aligned}$$

so $(r, r * p * h \bmod q) \leftarrow \mathcal{A}'(y)$ with probability ϵ, from which it is trivial to obtain r and $\hat{w}$. $\square$

C Choices of `genf`, `geng` and `genr`

In choosing `genf`, `geng` and `genr`, our aims are to:

1. avoid exhaustive search attacks,

2. avoid lattice attacks,

3. avoid algebraic attacks based on the structures of f, g, r,

4. maintain a negligible probability of average case decryption failure.

For example if the space $\mathcal{S}_r$ is too small, or there is a particularly common point in the image when $\rho \stackrel{R}{\leftarrow} \mathcal{R}_r$, then one might guess r, either by brute force or by using a meet-in-the-middle method [18]. If a brute force attack on (f, g) is infeasible, but f and g are still small, they may fall to a lattice attacker. If g is made of $(\mathtt{0},\mathtt{1}|\mathtt{1},\mathtt{0})^{N/2}$, then we know it is orthogonal to $(\mathtt{1},\mathtt{1},-\mathtt{1},-\mathtt{1},\mathtt{0},\mathtt{0}\ldots)$ and all its (double) rotations, and this knowledge can be used to reduce the dimension of the associated lattice problem. Or if f, g, r, $\hat{w}$ are in some sense too large, the spread of $f * \hat{w} + p * r * g$ will frequently be larger than q, leading to decryption failures.

Here we assume that we can pick `genf`, `geng` and `genr` to satisfy the requirement that for all $M \in \mathcal{M}$,

$$\Pr[\mathcal{D}(\mathcal{E}(M, \rho)) = M] \quad \geq \quad 1 - \epsilon$$

for some negligible quantity ϵ, where this probability is defined over $\kappa \stackrel{R}{\leftarrow} \mathcal{R}_{\mathcal{K}}$ and $r \stackrel{R}{\leftarrow} \mathcal{R}_{\mathcal{E}}$. Note that this requirement does not rule out the possibility that individual keys may experience more decryption failures than average, but it does guarantee that such keys will be extremely rare.

Arguably the most natural choices for the sets $\mathcal{S}_f, \mathcal{S}_g$ and $\mathcal{S}_r$ are $\widehat{R}_{d_f}, \widehat{R}_{d_g}$ and $\widehat{R}_{d_r}$ respectively, for some public constants $d_f, d_g, d_r \in \mathbb{Z}$. Accordingly the most natural choice for the algorithms `genf`, `geng` and `genr` would then sample uniformly at random from these sets, under the condition that they satisfy any invertibility requirements. We will refer to this parameter set with a (0) superscripts, e.g.

$$\mathtt{genf}^{(0)} : \mathcal{R}_f^{(0)} \to \mathcal{S}_f^{(0)} = \widehat{R}_{d_f}.$$

Another natural choice of parameters, denoted with a (1) superscript, is to have $\mathtt{geng}^{(1)}$ and $\mathtt{genr}^{(1)}$ as above, but let

$$\mathcal{S}_f^{(1)} \quad = \quad \left\{ 1 + p * F \;\middle|\; F \in \widehat{R}_{d_f} \right\},$$

and let $\mathtt{genf}^{(1)}$ sample uniformly at random from this space (again under invertibility requirements).

This circumvents the need for keeping $f_p^{-1} = 1$ in step 1 of decryption, since we now have that:

$$\begin{aligned} A &= f * c \\ &= f * \hat{w} + p * r * g \\ &= (1 + p * F) * \hat{w} + p * r * g \\ &= \hat{w} + p * (F * \hat{w} + r * g) \in R_q \end{aligned}$$

Assuming $a = \hat{w} + p * (F * \hat{w} + r * g) \in R$, then just a modular reduction will reveal $\hat{w}$, after which r can be obtained in the usual way.

The downside of using this idea is that this form of decryption will typically give decryption polynomials with a bigger spread of coefficients, increasing the chance of decryption failures.

Yet another choice, is to let f and r have some kind of product structure, e.g. $f = f_1 * f_2 + f_3$. This helps since the computation

$$f * c \quad = \quad f_1 * (f_2 * c) + f_3 * c$$

can be computed very efficiently if the number of ones in f_1, f_2 and f_3 are small. See [10] for more details.

We stress that the choice of algorithms `genf`, `geng`, `genr` does not alter the *proof* of security, but that each choice of algorithms implies a slightly different underlying hard problem. There are currently no known attacks against any of the choices presented in this section, with the first two being the more well-studied.