Cryptanalysis of the Revised NTRU Signature Scheme

Craig Gentry[1] and Mike Szydlo[2]

[1] DoCoMo USA Labs, San Jose, CA, USA
cgentry@docomolabs-usa.com
[2] RSA Laboratories, Bedford, MA, USA
mszydlo@rsasecurity.com

Keywords: NSS, NTRU, NTRUSign, Signature Scheme, Lattice Reduction, Cryptanalysis, Orthogonal Lattice, Cyclotomic Integer, Galois Congruence.

Abstract. In this paper, we describe a three-stage attack against Revised NSS, an NTRU-based signature scheme proposed at the Eurocrypt 2001 conference as an enhancement of the (broken) proceedings version of the scheme. The first stage, which typically uses a transcript of only 4 signatures, effectively cuts the key length in half while completely avoiding the intended hard lattice problem. After an empirically fast second stage, the third stage of the attack combines lattice-based and congruence-based methods in a novel way to recover the private key in polynomial time. This cryptanalysis shows that a passive adversary observing only a few valid signatures can recover the signer's entire private key. We also briefly address the security of NTRUSign, another NTRU-based signature scheme that was recently proposed at the rump session of Asiacrypt 2001. As we explain, some of our attacks on Revised NSS may be extended to NTRUSign, but a *much* longer transcript is necessary. We also indicate how the security of NTRUSign is based on the hardness of several problems, not solely on the hardness of the usual NTRU lattice problem.

1 Introduction

The Revised NTRU Signature Scheme (R-NSS) and "NTRUSign" are the two most recent of several signature schemes related to the NTRU encryption scheme (now called NTRUEncrypt). NTRUEncrypt and the related signature schemes are not based on traditional hard problems such as factoring or computing discrete logarithms, like much of today's cryptography. Instead, NTRUEncrypt was originally conceived as a cryptosystem based on polynomial arithmetic. Based on an early attack found by Coppersmith and Shamir [7], however, the underlying hard problem was soon reformulated as a lattice problem. See [22] for an update on how lattices have recently been used both as a cryptanalytic tool and as a potential basis for cryptography.

There are two reasons for seeking alternative hard problems on which cryptography may be based. First, it is prudent to hedge against the risk of potential breakthroughs in factoring and computing discrete logarithms. A second and more significant reason is efficiency. NTRU-based algorithms, for example, are touted to run hundreds of times faster while providing the same security as competing algorithms. The drawback in using alternative hard problems is that they may not be as well understood. Although lattice theory has been studied for over 100 years,[1] the *algorithmic* nature of hard lattice problems such the "shortest vector problem" (SVP) was not really studied *intensively* until Lenstra, Lenstra and Lovász discovered a polynomial-time lattice basis reduction algorithm in 1982. Moreover, NTRU-based schemes use specific types of lattices based on an underlying polynomial ring, and these lattices generate specific types of lattice problems that may be easier to solve than general lattice problems. Since these specific lattice problems have been studied intensively only since NTRU-Encrypt's introduction in 1996, we can expect plenty of new results. This paper is a case in point: we use a new polynomial-time algorithm to find the shortest vector in certain lattices that arise in R-NSS, allowing us to break the scheme.

1.1 History of NTRU-based Signature Schemes

Since the invention of NTRUEncrypt in 1996, several related identification and signature schemes have been proposed. These include an identification scheme invented by Kaliski, et.al. in 1997 [12], a "preliminary" version of NSS presented at the rump session of Crypto 2000, and the scheme described in the proceedings of Eurocrypt 2001 [17]. (See also [16] and [4].) All of these have been broken. (See [21] and [9].) In their Eurocrypt presentation, the authors of NSS sketched a revised scheme. They described these revisions in more detail in a technical note entitled "Enhanced Encoding and Verification Methods for the NTRU Signature Scheme" [14], which was revised several months later [15]. They finally committed to a scheme, which we will call "R-NSS," by publishing it in the preliminary cryptographic standard document EESS [5]. They also published analysis and research showing how the new scheme defeated previous attacks. Although R-NSS does indeed appear to be a significantly stronger scheme than previous versions, this paper describes how it can be broken.

Since the initial submission of this paper, NTRU has proposed a new NTRU-based signature scheme called NTRUSign. Although our primary focus is R-NSS, we also provide security analysis of NTRUSign, as requested by the Program Committee.

1.2 Our Cryptanalysis

In our cryptanalysis of R-NSS, we use for concreteness the parameters suggested in the technical note [15] and standards document [5]. We show how a passive adversary who observes only a few valid signatures can recover the signer's entire private key. Although some might consider R-NSS to be even more *ad hoc*

[1] This includes early work by Hermite and Minkowski, the latter calling the topic "Geometrie der Zahlen" (Geometry of Numbers) in 1910.

than previous NTRU-based signature schemes, our attacks against it are more fundamental than previous attacks, in that they target the basic tenets of the scheme rather than its peculiarities.

The rest of this paper is organized as follows: In Section 2, we provide background mathematics, and then in Section 3, we describe R-NSS. In Section 4, we survey the previous attacks on NTRU-based signature schemes that are relevant to our cryptanalysis of R-NSS. In Section 5, we detail the first stage of our attack: the lifting procedure. Next, in Section 6, we describe how to obtain the polynomial $f * \overline{f}$, which we use in the final stage of the attack. In Section 7, we introduce novel techniques for circulant lattices which enable a surprising algorithm to obtain the private key f in polynomial time. We give a summary of our R-NSS cryptanalysis in Section 8. Finally, in Section 9, we describe NTRUSign, consider attacks against it and describe alternative hard problems that underlie its security.

2 Background Mathematics

As with NTRUEncrypt and previous NTRU-based signature schemes, the key underlying structure of R-NSS is the polynomial ring

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

where N is a prime integer (e.g., 251) in practice. In some steps, R-NSS uses the quotient ring $R_q = \mathbb{Z}_q[X]/(X^N - 1)$, where the coefficients are reduced modulo q, and normally taken in the range $(-q/2, q/2]$, where q is typically a power of 2 (e.g., 128). Multiplication in R and in R_q is also called convolution. We define $\|f\|$ to be the Euclidean norm of a polynomial in the basis $\{1, x, \ldots\}$ and the convolution matrix of f to be the matrix whose rows correspond to $\{f, xf, x^2 f \ldots\}$.

At times, we will refer to the reversal $\overline{a}$ of a polynomial a, defined by $\overline{a}_k = a_{N-k}$ (with $\overline{a}_0 = a_0$). The mapping $a \mapsto \overline{a}$ is an automorphism of R, since applying the map twice yields the original polynomial. We use the term "palindromes" in referring to polynomials that are fixed under the reversal mapping on R – i.e., polynomials a such that $a = \overline{a}$. For any $a \in R$, it is easy to see that the product $a * \overline{a}$ is a palindrome. This fact, as well as the reversal mapping, may be described in elementary terms, but also in terms of an automorphism of the underlying cyclotomic field $\mathbb{Q}(\zeta_N)$. We refer the reader to the full version of the paper for further details on the Galois theory of R and $\mathbb{Q}(\zeta_N)$.

2.1 Lattices

The analysis of R-NSS will make frequent use of *lattices*. Formally, a lattice is a discrete subgroup of a vector space, but concretely, a lattice may be presented as the integral span of some set $B = \{b_0, \ldots, b_{m-1}\}$ of linearly independent vectors in $\mathbb{R}^N$ - that is,

$$L = \{v | v = \sum a_i\, b_i \mid a_i \in \mathbb{Z}\} \ . \tag{2}$$

We call m the dimension of the lattice, and B a basis of L. Bases will often be presented as a matrix in which the rows are the basis vectors $\{b_i\}$. Each lattice has an infinite number of bases, related by $B' = UB$ where U is a unimodular matrix, but some bases are more useful than others. The goal of *lattice reduction* is to find useful bases, typically ones with reasonably short, reasonably orthogonal basis vectors. The most celebrated lattice reduction algorithm is LLL [19], which has found many uses in cryptology. The contemporary survey [22] provides an overview of lattice techniques and [2] provides detailed descriptions of LLL variants.

The most famous lattice problem is the shortest vector problem (SVP): given a basis of a lattice L, find the shortest nonzero vector in L. Although LLL and its variants manage to find *somewhat short* vectors in lattices, they do not necessarily find the *shortest* vector. In fact, SVP is an NP-hard problem (under randomized reductions) [1]. In previous cryptanalysis of NTRU and NSS, LLL's inability to recover the shortest (or a very, very short) vector was a significant shortcoming. In some of our attacks, however, we will construct lattices in which even vectors that are only somewhat short reveal information about the signer's private key, and then we will use LLL and its variants as black box algorithms to find these vectors. We will explain other aspects of lattice theory as they become relevant.

2.2 Ideals

Since R-NSS operates with polynomials in the ring R, we will need to consider multiplication in R, as well as ideals in this ring. Recall that an *ideal* is an additive subgroup of a commutative ring which is also closed under multiplication by any element in R, and a *principal ideal* is an ideal of R consisting of all R-multiples of a single element. We write (a) to denote the principal ideal consisting of all R-multiples of a, and say that the ideal is generated by a. We remark that not all ideals are principal, and furthermore, a generator of a principal ideal is not unique since there are infinitely many units $u \in R$, and for each u, both M_f and $M_{(f*u)}$ define the same ideal. We naturally extend these notions to lattices by defining a *lattice ideal* to be a lattice which is also closed under "multiplication" by polynomials in R. Each lattice $L(M_a)$ is a principal lattice ideal, and we will exploit this extra structure in our novel attacks on R-NSS.

3 Description of R-NSS

The signature scheme R-NSS is a triplet (*keygen*, *sign*, *verify*) of algorithms operating on polynomials in $R = \mathbb{Z}[X]/(X^N - 1)$ and $R_q = \mathbb{Z}_q[X]/(X^N - 1)$, where N is prime, and $q < N$, (e.g. $N = 251$, $q = 128$). Other parameters in R-NSS include the modulus p, which is relatively prime to q and is typically chosen to be 3, as well as the integers d_u, d_f, d_g, d_m and d_z, whose suggested values are respectively 88, 52, 36, 80, and 58. The latter parameters are used to define several families of *trinary* polynomials as follows: $\mathcal{L}(d_1, d_2)$ denotes the set of polynomials in R_q, with d_1 coefficients 1, d_2 coefficients -1 and all other coefficients 0.

Key generation: Two polynomials f and g are randomly generated according to the equations

$$f = u + pf_1$$

$$g = u + pg_1$$

where $u \in \mathcal{L}(d_u, d_u + 1)$, $f_1 \in \mathcal{L}(d_f, d_f)$ and $g_1 \in \mathcal{L}(d_g, d_g)$. The signer keeps these two polynomials secret, with f serving as the signer's private key. The public key h is computed as $f^{-1} * g$ in R_q, and it is therefore necessary that f be invertible in R_q (i.e., $f * f^{-1} = 1$ for some $f^{-1} \in R_q$). This is true with very high probability (see [24]); in any case the preceding step may be repeated by choosing a different polynomial f_1.

As in previous versions of NSS, the coefficients of f and g are small – i.e., they lie in a narrow range ($[-4, 4]$ assuming $p = 3$) of $\mathbb{Z}_q$. However, R-NSS introduces a new secret polynomial – namely, u – into the private key generation process. In the previous version of NSS, $f \pmod p$ and $g \pmod p$ were public, allowing a statistical attack on a transcript of signatures. In the full version of the paper we briefly describe this transcript attack, and explain how using u defeats it.

Signature generation: To sign a message, one transforms the message to be signed into a message representative according to a hash function-based procedure such as that described in [4]. We do not base any attack on this encoding, which can be made as safe as for any signature scheme. This message representative m is a polynomial in $\mathcal{L}(d_m, d_m)$. The signer then computes the following temporary variables:

$$y = u^{-1} * m \bmod p$$

$$z \in \mathcal{L}(d_z, d_z)$$

$$w = y + pz \ ,$$

where u^{-1} is computed in $R_p = \mathbb{Z}_p[X]/(X^N - 1)$. Notice that in R, $f * w \equiv m \pmod p$. This is not necessarily the case in R_q, however, since reduction modulo q causes "deviations" in the modulo p congruence, given that p and q are relatively prime. During the rest of the signing process, the signer will try to keep the number of these deviations to a minimum. The signer computes more temporary variables:

$$s = f * w \bmod q$$

$$t = g * w \bmod q$$

$$Dev_s = (s - m) \bmod p$$

$$Dev_t = (t - m) \bmod p \ .$$

Dev_s and Dev_t represent the deviations that the signer would like to correct, but, unfortunately for the signer, correcting a coefficient in s may cause additional deviations in t. Therefore, the signer limits his corrections to coefficient positions j such that $(Dev_s)_j = (Dev_t)_j$. He initializes a polynomial e to 0 and sets e_j to $-(Dev_s)_j$ when $(Dev_s)_j = (Dev_t)_j$. He then lets $e' = u^{-1} * e \pmod{p}$, adds e' to w, and recomputes $s = f * w$ in R_q. The pair (m, s) is the signer's signature of m.

Signature verification: To avoid the forgery attacks presented in [9], verification has become a rather complicated process involving up to 20 distinct steps, detailed in [6] and [15], which fall into three broad categories: the *Quartile Distribution* tests, the *Mod 3 Distribution* tests, and the *L2 Norm* tests. In essence, the verifier checks, respectively, that

1. the coefficients of s and $t = s*h \pmod{q}$ have a roughly normal distribution;
2. the coefficients of s and t deviating from m are few and have a certain distribution; and
3. the L2 norms of $s' = p^{-1}(s - m) \pmod{q}$, $t' = p^{-1}(t - m) \pmod{q}$ and $(s' \| t')$ (concatenated) are below certain thresholds.

Since we do not focus on forgery attacks in this paper, we do not describe the verification process in full detail here. On average, the signer has to run the signing process two or three times to produce a valid signature.

4 Previous Attacks on NSS

In this section, we review some relevant known attacks against NTRUEncrypt and previous NTRU-based signature schemes. This review will help us explain our cryptanalysis of R-NSS, which will occasionally leverage pieces of the attacks mentioned here.

4.1 Coppersmith-Shamir Attack

As with NTRUEncrypt, the security of R-NSS is claimed to be based on a hard lattice problem. Coppersmith and Shamir [7] were the first to present a lattice-based attack against NTRUEncrypt, an attack which is also relevant to R-NSS. Let L_{CS} be the lattice generated by the rows of the following matrix:

$$B_{CS} = \begin{bmatrix} I_{(N)} & M_h \\ 0 & qI_{(N)} \end{bmatrix} , \tag{3}$$

where $I_{(N)}$ is the N-dimensional identity matrix. This lattice clearly contains the vector $(f \| g)$, since $f * h = g \pmod{q}$. Moreover, for technical reasons [7], it is highly probable that $(f \| g)$ is the shortest nonzero vector in this lattice (up to rotation, sign, and excluding trivial vectors such as $(1^N, 1^N)$, the vector of all 1's.) Therefore, recovering the private key is simply a matter of recovering the shortest vector in L_{CS}, which we can presumably do using a lattice reduction algorithm. This attack is very effective when N is small (e.g., 107).

The problem with this approach (and lattice attacks, in general) is that no known lattice reduction algorithm is both very fast and very effective. More specifically, the LLL algorithm is "polynomial-time" – i.e., it terminates in time polynomial in the dimension m of the lattice – but, for high-dimensional lattices, such as those used in NTRU-based schemes, it almost certainly will *not* find the shortest vector. Rather, LLL only guarantees finding a vector that is no more than $2^{(m-1)/2}$ times as long as the shortest vector. Even though, in practice, LLL performs significantly better than this worst case bound, its performance is not sufficient for this lattice; we need the shortest vector or a *very* small multiple thereof. Other lattice reduction algorithms can find shorter vectors, but they naturally have greater time-complexity. The bottom line, based on current knowledge and on extensive empirical tests run by NTRU [23], seems to be that the time necessary to find $(f\|g)$ in L_{CS} grows at least exponentially in the dimension of the lattice $(2N)$. This apparently hard lattice problem of recovering $(f\|g)$ from the $2N$-dimensional lattice is claimed to underlie the security of both NTRUEncrypt and R-NSS.

Remark 1. As mentioned previously, the fact that SVP is an NP-hard problem for general lattices does not necessarily mean that finding short vectors in L_{CS} is hard. May [20] exploited the specifics of NTRUEncrypt's private key structure to construct lower-dimensional "zero-run" lattices and "dimension-reducing" lattices from which an attacker could quickly recover an NTRUEncrypt-107 private key. Gentry [8] used a ring homomorphism from R to $\mathbb{Z}[X]/(X^{N/d}-1)$ to "fold" L_{CS} into a more manageable lattice of dimension $2N/d$ for N having a nontrivial divisor d.

4.2 GCD Lattice Attack

Since reducing L_{CS} does indeed appear to be infeasible, the natural inclination of the cryptanalyst is to look for smaller lattices that contain the private key. The authors of R-NSS mention one such lattice in [16]. They observe that if an attacker is able to recover the values of several $f * w$'s in R – "unreduced" modulo q – then f will likely be the shortest vector in the N-dimensional lattice formed by the rows of the several M_{f*w}'s.

Recall from section 2.2 that, for any polynomial a, there is an equivalence between the ideal (a) of a-multiples and the lattice generated by M_a. Similarly, the lattice spanned by the rows of M_{f*w_1} and M_{f*w_2} corresponds to the ideal $I = (f * w_1, f * w_2)$. Every polynomial in I is a multiple of f. Moreover, if (w_1) and (w_2) are relatively prime – i.e., there exist $a, b \in R$ with $a * w_1 + b * w_2 = 1$ – then $f \in I$, and we may say that $\mathrm{GCD}(f * w_1, f * w_2) = (f)$. This "lattice attack" is, in fact, the standard ideal-GCD algorithm, and is among the lattice ideal operations discussed in [2].

Given how the w_i are produced in R-NSS, (f) often is indeed the GCD of two unreduced signatures, and it is even more likely to be the GCD of several unreduced signatures. Therefore, given a few unreduced signatures, we can construct an N-dimensional lattice whose shortest vector is likely f. The authors of R-NSS note that, although reducing this N-dimensional lattice is still not a trivial problem for R-NSS parameters, it is much easier than reducing the $2N$-dimensional L_{CS}, given the exponential relationship between dimension and running time. R-NSS uses "masking" techniques to prevent recovery of $f * w$'s in R to avoid this lattice attack. (In section 5 we show that recovering $f * w$'s is nonetheless quite easy.)

4.3 Averaging Attack

The so-called "averaging attack" was first considered by Kaliski during collaboration with Hoffstein in the context of an early precursor to NSS (see patent [12]). In this work, it was observed that in the ring R, the value $f * \overline{f}$ could be obtained by an averaging attack. This attack was fatal, since in the scheme [12], f itself is a palindrome (unlike in R-NSS), thus $f * \overline{f} = f^2$. There is an efficient algorithm for taking the square root in R (see, e.g., [13]), so $f * \overline{f}$ revealed f.

In [16], the authors of R-NSS do mention this averaging attack, but also remark that knowledge of $f * \overline{f}$ does not appear to be useful. See [16], [9], [21] for a discussion of this and other ways in which the attacker may average a transcript of signatures in such a way as to get information about the private key.

Here is a description of how the averaging attack works. Suppose we can obtain a set of unreduced $f * w$'s in R. Now, consider the average

$$A_r = (1/r) \sum_{i=1}^{r} (f * \overline{f}) * (w_i * \overline{w_i}) \ ..$$

For each i, $(w_i * \overline{w_i})_0 = \|w_i\|^2$, which is a large positive quantity. However, for $k \neq 0$, $(w_i * \overline{w_i})_k$ has a random distribution of positive and negative quantities that averaging essentially cancels out. Thus, as r increases, A_r essentially converges to a scalar multiple of $f * \overline{f}$. This convergence is quite fast: after a few thousand signatures a close estimate of $f * \overline{f}$ can be computed, meaning that we obtain an estimate z such that for most coefficients, $|z_i - (f * \overline{f})_i| \leq 2$. Even if reduced signatures are used, with some corrections, there is still a convergence to $f * \overline{f}$, albeit about 10 times as slow. Clearly, the more signatures, the better the estimate. In section 6, we explain how this averaging attack may be combined with a lattice attack to recover $f * \overline{f}$ quickly and completely.

5 Lifting the Signatures

In this section, we present our first (and arguably the most important) attack with which we obtain R-multiples of the private key f. More specifically, we assume that, as passive adversaries, we are given a transcript of legitimate signatures $\{(m_1, s_1), \ldots, (m_r, s_r)\}$. Using this transcript and the signer's public key, we directly compute the following elements in R_q:

$$\{f * w_1 \bmod q, \ldots, f * w_r \bmod q\} \quad \text{and} \quad \{g * w_1 \bmod q, \ldots, g * w_r \bmod q\} \ ,$$

where the w_i are computed according to the signing process above. We will then lift these signatures to the ring R, obtaining a list of multiples of f and g:

$$\{f * w_1, \ldots, f * w_r\} \quad \text{and} \quad \{g * w_1, \ldots, g * w_r\} ,$$

which are "unreduced" modulo q. This is a devastating attack against R-NSS, because undoing the q modular reduction of the signatures allows us to use the N-dimensional GCD lattice attack, described in section 4.2, rather than the $2N$-dimensional Coppersmith-Shamir attack, reducing the key recovery time by a factor exponential in N. It also permits the other, more efficient attacks discussed later in this paper.

5.1 The Principle

¿From the signing procedure in the ring R_q, we get the following equations for each i:

$$f * w_i \equiv s_i \bmod q \quad \text{and} \quad g * w_i \equiv t_i \bmod q \; , \tag{4}$$

$$f * w_i \equiv g * w_i \equiv m_i + e_i \bmod p \; . \tag{5}$$

If we knew e_i, we would be able to compute $f * w_i$ and $g * w_i$ modulo pq via the Chinese Remainder Theorem. We could then recover $f * w_i$ and $g * w_i$ over R without too much difficulty, since almost all of the coefficients of $f * w_i$ and $g * w_i$ will lie in the interval $(-pq/2, pq/2]$.[2]

The use of e_i in the signing process makes lifting the signatures much less straightforward. Since e_i will have about 20 nonzero coefficients for the suggested parameters of R-NSS, we cannot simply guess e_i from among $\binom{251}{20}2^{20}$ possibilities. Later, we will mention how specific properties of the signing process make some possibilities much more probable than others, but even with these refinements the guessing approach remains infeasible.

Instead, we will use an iterative approach that, at each step, attempts to improve upon previous approximations. Let S_i denote our approximation of $f * w_i$, and T_i our approximation of $g * w_i$. We initialize S_i and T_i to be the polynomials in R_{pq} that satisfy

$$S_i \equiv s_i \bmod q \quad \text{and} \quad T_i \equiv t_i \bmod q \; , \tag{6}$$

$$S_i \equiv T_i \equiv m_i \bmod p \; . \tag{7}$$

Our approximations will have two different types of errors. First, the kth coefficients of S_i and T_i will be wrong if the kth coefficient of e_i is nonzero. Second, a coefficient of S_i (resp. T_i) may be correct modulo pq but incorrect in R if the corresponding coefficient of $f * w_i$ (resp. $g * w_i$) lies outside the interval $(-pq/2, pq/2]$. On average, about 25 (out of 251) coefficients of our initial approximations will be incorrect.

[2] See [7] for an analysis of the coefficient distributions of convolution products. The key points here are that L2 norms $\|f\|$, $\|g\|$ and $\|w_i\|$ are small, $\|f * w_i\| \approx \|f\| \|w_i\|$ and the coefficients of $f * w_i$ have a roughly normal distribution.

In refining our approximations, we begin with the following observation: For any (i, j),

$$(f * w_i) * (g * w_j) - (f * w_j) * (g * w_i) = 0 \ . \tag{8}$$

Based on this observation, we would expect $S_i * T_j - S_j * T_i$ to tend towards 0 as our approximations improve. In fact, this is the case. We can use the norms $n_{ij} = \|S_i * T_j - S_j * T_i\|$ to decide what adjustments we should make, and to know when our approximations are finally correct. The rest of the lifting procedure is simply an elaboration of this basic idea, and one can imagine a variety of different methods through which an attacker could use these norms to create an effective lifting procedure. In our particular implementation against R-NSS, we used the norms n_{ij}, together with some R-NSS-specific heuristics, to create a very fast lifting procedure that worked almost all the time with a transcript of only four signatures.

5.2 Our Implementation of the Lifting Procedure

For each approximation pair (S_i, T_i), we computed a "norm product" with the other approximation pairs according to the formula $P_i = \prod_{j \neq i} n_{ij}$. Preferring approximation pairs with higher norm products, we then picked a random pair (S_i, T_i) to be corrected. For each coefficient position, we temporarily added or subtracted certain multiples of q to S_i and T_i (since the approximations are already correct modulo q), and recomputed P_i. With a little bookkeeping, this step can be made extremely fast. We preserved the adjustment that reduced P_i by the greatest amount. Finally, we terminated this process when the norm product of some approximation pair reached zero, at which point we would have two correct approximation pairs.

We note that not every adjustment made during the lifting procedure is actually a correction. Often, this procedure will make a previously correct coefficient incorrect, and then switch it back later on. In other words, it behaves somewhat like a "random walk". This fact, together with the heuristic nature of the overall algorithm, admittedly makes the lifting procedure difficult to analyze. For the specified parameters of R-NSS, however, it works quickly and reliably. For a transcript of four signatures, it is able to lift two signatures 90% of the time in an average of about 25 seconds (on a desktop computer). In the remaining 10%, the number of errors never converges to zero. For three signatures, it still works 70% of the time, typically finishing in about 15 seconds.

6 Obtaining $f * \overline{f}$

An important ingredient of the final algorithm is the product of f with its reversal, $\overline{f}$. In order to recover $f * \overline{f}$ in the context of R-NSS, we used a combination of the averaging attack mentioned in section 4.3 and a lattice attack on $f * \overline{f}$ noticed by the authors and Jonsson, Nguyen and Stern.

The lattice attack is a derivative of the Coppersmith-Shamir attack described in section 4.2. Since sending a polynomial to its reversal is an automorphism,

$(f * \overline{f}) * (h * \overline{h}) \equiv (g * \overline{g}) \pmod q$. This means that the vector $(f * \overline{f} \| g * \overline{g})$ is contained in the lattice L_{norm} generated by

$$B_{norm} = \begin{bmatrix} I_{(N)} & M_{h*\overline{h}} \\ 0 & qI_{(N)} \end{bmatrix} . \tag{9}$$

This lattice has dimension $2N$, but it has an $(N+1)$-dimensional sublattice of palindromes, which contains $(f * \overline{f} \| g * \overline{g})$. Conceivably, recovering $(f * \overline{f} \| g * \overline{g})$ from this sublattice could give us useful information about f and g. However, this attack fails for typical NTRU or NSS parameters, since $(f * \overline{f} \| g * \overline{g})$ is normally not the shortest vector.[3]

For R-NSS, we combine ideas from the above attack with the GCD attack in 4.2 and the averaging technique in 4.3. First, we use our unreduced signatures to form the ideal $(f * \overline{f})$ from a few unreduced signature products $s * \overline{s} = f * \overline{f} * w_i * \overline{w_i}$, exactly as described in Section 4.2. Then, we take the subring of $(f * \overline{f})$ consisting of palindromes, which forms a lattice of dimension $(N+1)/2$. In fact, this lattice is generated by $f * \overline{f}$, and $(N-1)/2$ vectors $(X^k + X^{N-k}) * f * \overline{f}$. For the same reason as above, $f * \overline{f}$ might not be the shortest vector in the lattice. However, we may use the averaging attack to obtain a good estimate t of $f * \overline{f}$, modify the lattice to include t, and then use lattice reduction to obtain the (shortest) vector $t - f * \overline{f}$.[4] In practice, this attack is amazingly effective for two reasons: the lattice problem is only $(N+1)/2$ dimensional, and $\|t - f * \overline{f}\|$ will be much less than $\|f * \overline{f}\|$ for even a *very* poor estimate of t. We found that we needed only 10 signatures to obtain a sufficiently accurate estimate t of $f * \overline{f}$ (even though only a handful of coefficients in t were actually correct). With these 10 signatures, we consistently recovered $f * \overline{f}$ in less than 10 seconds.

7 Orthogonal Congruence Attack

In this section, we describe a polynomial-time algorithm for recovering the private key f from $f * \overline{f}$ and one other multiple of f, such as $f * w$, when w is relatively prime to $\overline{f}$. In other words, this algorithm requires $f * \overline{f}$ and a basis B_f of the ideal (f). [5] This algorithm is quite surprising, and uses novel ideas combining orthogonal lattices with number theoretic congruence arising from the cyclotomic field $\mathbb{Q}(\zeta_N)$.

The complete algorithm is rather complex, but here is a brief (and not entirely accurate) sketch: We begin by choosing a large prime number $P \equiv 1 \pmod N$.

---

[3] By the "Gaussian heuristic," the expected length of the shortest vector in a lattice of determinant d and dimension n is $d^{1/n}\sqrt{n/(2\pi e)}$. For L_{norm}, this length is $\sqrt{qN/(\pi e)}$. On the other hand, since $\|f\| > \sqrt{N}$ in NSS and $\|f * \overline{f}\| \geq \|f\|^2$ (the latter inequality following from $(f * \overline{f})_0 = \|f\|^2$), the norm of $f * \overline{f}$ is greater than N and hence greater than the Gaussian heuristic, since N is typically chosen to be greater than q.

[4] One could also use Babai's algorithm to solve the closest vector problem (CVP).

[5] Yet another characterization of the algorithm is that it recovers f from B_f and the relative norm of f over the index 2 subfield of $\mathbb{Q}(\zeta_N)$.

(For now, we defer discussing how large P must be.) Then, using $f * \overline{f}$ and our basis for (f), we use a series of lattice reductions to obtain $f^{P-1} * a$ for some polynomial a, and a guarantee that $\|a\| < P/2$. Using the congruence $f^{P-1} \equiv 1 \pmod{P}$, we will be able to compute $a \pmod{P}$ and hence a exactly, from which we will be able to compute f^{P-1} exactly. We will then use this power of f to recover f.

We describe this algorithm and the theory behind it in more detail below. Our first task will be to find a tool that ensures that when LLL gives us $f^{P-1} * a$, there is a definite bound on $\|a\|$.

7.1 Orthogonal Lattices

Certain lattices possess a basis of N equal length, mutually perpendicular basis vectors. We denote such lattices *orthogonal lattices*. Two lattices are called *homothetic* if up to a constant stretching factor, λ, there is a distance preserving map from one lattice to the other. That is, all orthogonal lattices are homothetic to the trivial lattice $\mathbb{Z}^N$. Similarly, we define f to be an *orthogonal polynomial* if the circulant matrix M_f is the orthogonal basis of an orthogonal lattice. We are interested in orthogonal lattices because they possess a multiplicative norm property.

We note that for randomly chosen polynomials a and f, the norm is quasi-multiplicative, $\|f * a\| \approx \|f\| \cdot \|a\|$. However, if one of the polynomial factors, say f, is *orthogonal*, then equality will hold

$$\|f * a\| = \|f\| \cdot \|a\| \ . \tag{10}$$

For general polynomials f, applying LLL to (f) is guaranteed to find a multiple of f, say $f * a$, such that the norm $\|f * a\|$ is less than a specific factor times the norm of the shortest vector in the lattice. In the case where f is orthogonal, we can additionally bound $\|a\|$ by this factor, since $\|f * a\| = \|f\| \cdot \|a\|$. In the case of LLL, this means that we can be certain that $\|a\| < 2^{(N-1)/2}$.

7.2 Using $f * \overline{f}$ to Construct an Implicit Orthogonal Lattice

What do we do when f is not an orthogonal polynomial, but our objective is to find $f * a$ with small $\|a\|$ (given only $f * \overline{f}$ and lattice basis B_f of (f))? Of course, we may apply LLL to B_f and just hope that the output vector $f * a$ has short a, but this may not work even if f is only slightly nonorthogonal[6]. This section describes how we can accomplish this task by using knowledge of $f * \overline{f}$ to *implicitly* define an orthogonal lattice.

Since B_f and M_f are both bases of (f), they are related by $B_f = U \cdot M_f$ for some unimodular matrix U. Notice that each row of B_f is of the form $f * u_i$ where u_i is the ith row of U. This means that the objective of finding $f * a$ with

[6] The notion of nonorthogonality is made precise with concept called *orthogonality defect*.

bounded $\|a\|$ is equivalent to bounding the norms of the rows of U. So, in some sense, we would like to apply lattice reduction to U. How can we reduce the rows of U when we only know $B_f = U \cdot M_f$, and not U itself?

Supposing that f is a not a zero divisor in R, we can also divide by $f * \overline{f}$. Allowing denominators in our notation, we let $D = M_{(1/(f*\overline{f}))}$ and compute

$$B_f \cdot D \cdot B_f^T = U \cdot U^T , \tag{11}$$

which is the Gram matrix of our unknown unimodular matrix U. Although we do not know U explicitly, $U \cdot U^T$ has all the information that LLL needs to know about U in order to reduce it – namely, the mutual dot products $u_i \cdot u_j$ of each pair of row vectors. We can therefore apply a Gram matrix version of LLL to $U \cdot U^T$, which outputs the unimodular transformation matrix V, and the Gram matrix of the reduced lattice: $(V \cdot U) \cdot (V \cdot U)^T$. By the LLL bound, the norms of the rows of (the unknown) basis $V \cdot U$ will be bounded by $2^{(N-1)/2}$. Now, we can compute a new basis of (f) – namely, $(V \cdot U) \cdot M_f = V \cdot B_f$ – and be certain that each row of this basis equals $f * a_i$ for $\|a_i\| < 2^{(N-1)/2}$. Effectively, we have reduced the orthogonal lattice defined by U, without even knowing an explicit basis for it.

7.3 Galois Congruence

In addition to the orthogonal lattices technique, we use some interesting congruences on the ring R. The first congruence states that for any prime P such that $P \equiv 1 \pmod{N}$,

$$f^P = f \pmod{P} . \tag{12}$$

This implies that for any f which is not a zero divisor[7] in $R_P = \mathbb{Z}_P[X]/(X^N - 1)$ that

$$f^{P-1} = 1 \pmod{P} . \tag{13}$$

We may generalize these equations to arbitrary primes P by using a *Galois Conjugation* function (written as a superscript) $\sigma(r) : R \to R$, defined by

$$f^{\sigma(r)}(x) = f(x^r). \tag{14}$$

For any r not divisible by N, $\sigma(r)$ defines an automorphism on R. There are $N-1$ such automorphisms, since two values of r which differ by a factor of N define the same automorphism. We call $\sigma(r)$ the rth Galois conjugation mapping, and have the congruence

$$f^P = f^{\sigma(P)} \pmod{P} . \tag{15}$$

For elementary proofs of equations 13 and 15 and their relationship with the the Galois theory of $Q(\zeta)$, we refer the reader to the the full version of the paper.

[7] See the full version of the paper for a discussion of the zero divisor issue in R_P.

As mentioned above, our motivation for considering such congruences is that given a multiple of f^{P-1}, say $f^{P-1} * a$, we may use the congruence $f^{P-1} = 1 \pmod{P}$ to conclude that

$$f^{P-1} * a = a \pmod{P} . \tag{16}$$

Now, if a is so small that all of its coefficients lie in the interval $(-P/2, P/2]$, then the representatives for $f^{P-1} * a \pmod{p}$ in this interval reveal a exactly. Assuming that a is not a zero divisor in R, dividing the product by a then yields the exact value of f^{P-1}. With this observation, we are in a position to use orthogonal lattice theory with lattice reduction to obtain small multiples of powers of f and thus exact powers of f.

However, there is a technical difficulty arising from the fact that LLL only guarantees that $\|a\| < 2^{(N-1)/2}$. To ensure that a has coefficients in $(-P/2, P/2]$, P has to be quite large – about the same order of magnitude as $2^{(N-1)/2}$. This means that f^{P-1} has bit-length exponential in N, which makes it impossible for us to even store the value of this polynomial. Initially, this might appear to be a fatal problem. It also indicates that the "brief sketch" given at the beginning of Section 7 could not have been entirely accurate.

Fortunately, we will not need to work with f^{P-1} directly; it will be sufficient for our purposes to be able to compute f^{P-1} modulo some primes. As discussed further in section 7.4, we can make such computations using a process similar to repeated squaring. However, to recover f, we will need some power of f that we can work with directly (unreduced). To solve this problem, we may choose a second prime $P' \equiv 1 \pmod{N}$ with $\mathrm{GCD}(P-1, P'-1) = 2N$ for which we can compute $f^{P'-1}$ modulo some primes. Then, using the Euclidean Algorithm, we may compute f^{2N} modulo these primes, and ultimately f^{2N} exactly via the Chinese Remainder Theorem. We may work with f^{2N} directly, since its bit-length is merely polynomial in N. A more elegant solution (in the full version) computes f^{2N} directly from f^{P-1} if $\mathrm{GCD}(P-1, 2N-1) = 1$. In Section 7.5, we describe how to recover f from f^{2N}.

7.4 Ideal Power Algorithms

In practice, it might be the case that smaller P would be sufficient, allowing us to work with the ideal (f^{P-1}) directly, but some tricks are needed to handle the very large primes P that the LLL bound imposes on us. Suppose we had the chains of polynomials $\{v_0^2 * \overline{v_1}, \ldots, v_{r-1}^2 * \overline{v_r}\}$ and $\{v_0 * \overline{v_0}, \ldots, v_{r-1} * \overline{v_{r-1}}\}$. Then, we could make the following series of computations:

$$v_0^4 * \overline{v_2} = (v_0^2 * \overline{v_1})^2 * (v_1 * \overline{v_1})^{-2} * (v_1^2 * \overline{v_2}) \pmod{P} , \tag{17}$$

$$v_0^8 * \overline{v_3} = (v_0^4 * \overline{v_2})^2 * (v_2 * \overline{v_2})^{-2} * (v_2^2 * \overline{v_3}) \pmod{P} , \tag{18}$$

and so on, ending with the equation:

$$v_0^{2^r} * \overline{v_r} = (v_0^{2^{r-1}} * \overline{v_{r-1}})^2 * (v_{r-1} * \overline{v_{r-1}})^{-2} * (v_{r-1}^2 * \overline{v_r}) \pmod{P} . \tag{19}$$

In other words, we could compute $v_0^{2^r} * \overline{v_r} \pmod{P}$ efficiently even though the exponent 2^r may be quite large. If we could use this approach to get $v_0^{P-1} * \overline{v_r} \pmod{P}$ where $\|\overline{v_r}\| < P/2$, then we could recover $\overline{v_r}$ exactly, and then we could use the same chains of polynomials to compute v_0^{P-1} modulo other primes.

The main tool that we use is the multiplication of ideals (see [2]), and we adapt this technique to use the orthogonal lattice theory above. The algorithm described in the paragraph above is essentially a repeated squaring algorithm, so we first review how to multiply ideals, and in particular, obtain the ideal (f^2) from the ideal (f).

Remark 2. Ideal multiplication in R: If $A = (f)$ and $B = (g)$ are principal ideals generated as $\mathbb{Z}$-modules by $(f * a_1, \ldots, f * a_n)$ and $(g * b_1, \ldots, g * b_n)$, then the ideal product AB is generated as a $\mathbb{Z}$-module by the n^2 elements of the set $\{a_i * b_j * f * g\}$, which defines $(f * g)$.

Note that we describe the ideals as modules – i.e., as the $\mathbb{Z}$-span of a set of polynomials (rather than by giving generators over R). This is because our algorithms represent ideals as lattices – i.e., as lists of polynomials.

We use ideal multiplication as follows: Given the ideal (f) in terms of the basis $B_f = U \cdot M_f$, we can generate (f^2) from the rows $b_i * b_j = u_i * u_j * f^2$. Since we know $f^2 * \overline{f}^2$, we can use the orthogonal lattice method described in section 7.2 to obtain a reduced basis $B_{f^2} = U' \dot{M}_{f^2}$ where the rows of U' have norm less than $2^{(N-1)/2}$.[8] Next, we pick a row of B_{f^2} and name it $f^2 * \overline{v_1}$. We can directly compute $v_1 * \overline{v_1}$ and a basis for v_1: $U' \cdot M_{v_1}$. Now, we can compute a basis for (v_1^2) to begin another iteration of this process.

Thus, we have the chains of polynomials previously introduced, and with a modification to take into account the binary representation of $P-1$, we may obtain $f^{P-1} * \overline{v_r} \pmod{P}$. The polynomial complexity of the algorithm follows from the norm bound on the v_i's and the polynomial running time of LLL. By selecting a second prime P', as described in Section 7.3, we may obtain f^{2N} in polynomial time. In Appendix A, we write this algorithm in terms of pseudocode in an effort to clarify its details as well as its polynomial-time complexity.

7.5 Computing f from f^{2N}

Our final task is to compute the private key f from f^{2N}. We use the following theorem.

Theorem 1. *Galois Polynomial*
Given f^{2N} and $b \in \mathbb{Z}_N^*$, we may compute $z = f^{\sigma(-b)} f^b$ in time polynomial in b, N and bit-length of f.

[8] In the worst case, this involves reduction of an N-dimensional lattice defined by N^2 generators. This reduction would only have polynomial time-complexity, but normally we will be able to do much better, since we will usually be easy to find far fewer (on the order of N) polynomials that span the ideal.

Proof. Let $P_2 > 2\|f^{\sigma(-b)}f^b\|$ be a prime number such that $P_2 = 2c_2N - b$ for some integer c_2. Then, $(f^{2N})^{c_2} \equiv f^{P_2} * f^b \equiv f^{\sigma(-b)}f^b \pmod{P_2}$. Since $P_2 > 2\|f^{\sigma(-b)}f^b\|$, we recover $z = f^{\sigma(-b)}f^b$ exactly.

Now, in terms of recovering f, we first note that f^{2N} uniquely defines f only up to sign and rotation – i.e., up to multiplication by $\pm X^k$, the $2N$th roots of unity in R. The basic idea of our approach is that, given f^{2N}, fixing $f(\zeta)$ for one (complex) Nth root of unity ζ completely determines $f(\zeta^d)$ for all exponents d. Then, we may use the $N-1$ values of $f(\zeta^d)$, together with $f(1)$ (which we will know up to sign), to solve for f using Gaussian elimination. If we set $-b$ to be a primitive root modulo N, the polynomial z given in Theorem 1 will help us iteratively derive the $f(\zeta^d)$ from $f(\zeta)$ as follows:

$$f(\zeta^{(-b)^{i+1}}) = z(\zeta^{(-b)^i})/f^b(\zeta^{(-b)^i}) \ . \tag{20}$$

Repeated exponentiation will not result in a loss of precision, since the value may be corrected at each stage. Since $-b$ is a primitive root modulo N, these evaluations give us $N-1$ linearly independent equations in the coefficients of f, which together with $f(1)$, allow us to recover the private key f completely, up to sign and rotation.

8 Summary and Generalizations of R-NSS Cryptanalysis

For the reader's convenience, we briefly review the main points of the attack. The first two stages of the attack are fast in practice, but they are both heuristic and have no proven time bounds. The lifting procedure lifts a transcript of signatures from R_q to R, obtaining unreduced f-multiples in R. The second stage uses an averaging attack to approximate $f * \overline{f}$, and then solves the closest vector problem (CVP) to recover $f * \overline{f}$ exactly. The algorithm of the final stage, which we have not fully implemented, uses output from the previous two stages to recover the private key in polynomial time. By combining lattice-based methods and number-theoretic congruences, the algorithm of the final stage can be used to:

1. Recover f from $f * \overline{f}$ and a basis B_f of (f);
2. Recover f from only B_f when f is an orthogonal polynomial; and
3. Recover $f/\overline{f}$ from B_f whether f is an orthogonal polynomial or not.

We anticipate that this algorithm could be generalized to recover f given a basis of (f) and the relative norm of f over an index 2 subfield where the degree-2 extension is complex conjugation. In Section 9, we discuss another possible generalization of this algorithm that may be an interesting area of research.

9 NTRUSign

NTRUSign was proposed at the rump session of Asiacrypt 2001 as a replacement of R-NSS [11], and, as requested by the Program Committee, we provide some preliminary security analysis. The scheme is more natural than previous NTRU-based signature schemes, particularly in terms of its sole verification criterion: the signer (or forger) must solve an "approximate CVP problem" in the NTRU lattice – i.e., produce a lattice point that is sufficiently close to a message digest point, à la Goldreich, Goldwasser and Halevi [10]. Similar to the GGH cryptosystem, the signer has private knowledge of a "good" basis of the NTRU lattice having short basis vectors, and publishes the usual "bad" NTRU lattice basis:

$$B_{priv} = \begin{bmatrix} M_f & M_g \\ M_F & M_G \end{bmatrix} \qquad B_{pub} = \begin{bmatrix} I_{(N)} & M_h \\ 0 & qI_{(N)} \end{bmatrix} .$$

Unlike GGH, these bases may be succinctly represented as polynomials: (f, g, F, G) for the private basis, and h for the public basis. (Recall that M_f denotes the circulant matrix corresponding to the polynomial f; see Section 2.) In terms of key generation, the signer first generates short polynomials f and g and computes the public key as $h = f^{-1} * g \pmod q$, as in NTRUEncrypt or R-NSS. Due to lack of space, we refer the reader to [11] for details on how the signer generates his second pair of short polynomials (F, G), but we note the following properties: 1) $f * G - g * F = q$ and 2) $\|F\|$ and $\|G\|$ are 2 to 3 times greater than $\|f\|$ and $\|g\|$.

To sign, the message is hashed to create a random message digest vector (m_1, m_2) with $m_1, m_2 \in R_q$. The signer then computes:

$$G * m_1 - F * m_2 = A + q * C \ , \tag{21a}$$

$$-g * m_1 + f * m_2 = a + q * c \ , \tag{21b}$$

where A and a have coefficients in $(-q/2, q/2]$, and sends his signature:

$$s \equiv f * C + F * c \pmod q \ . \tag{22}$$

The verifier computes $t \equiv s * h \pmod q$ and checks that (s, t) is "close enough" to (m_1, m_2) – specifically,

$$\|s - m_1\|^2 + \|t - m_2\|^2 \leq \text{Normbound} \ . \tag{23}$$

We can see why verification works when we write, say, s in terms of Equations 21a and 21b:

$$s \equiv m_1 - (A * f + a * F)/q \pmod q \ , \tag{24}$$

where $\|A * f + a * F\|$ will be reasonably short since f and F are short.

In the absence of a transcript, the forgery problem is provably as difficult as the approximate CVP problem in the NTRU lattice. However, it is clear that NTRUSign signatures leak some information about the private key. The mapping involution $\mathbb{Z}[X]/(q, X^N - 1)$ sending $m \mapsto s$ is not a permutation, and the associated identification protocol is not zero knowledge (even statistically or computationally). Below, we describe concrete transcript attacks using ideas from our cryptanalysis of R-NSS. We have had fruitful discussions with NTRU regarding these attacks, and they have begun running preliminary tests to determine their efficacy. Based on these tests, some of these attacks appear to require a very long transcript that may make them infeasible in practice. This is a subject of further research. In any case, these attacks show that NTRUSign cannot have any formal security property, since it is not secure against passive adversaries.

9.1 Second Order Attack

Using Equation 24, we may obtain polynomials of the form $(A * f + a * F)$ (similarly, $(A * g + a * G)$). Consider the following average:

$$Avg_{ff}(r) = (1/r)\sum_{i=1}^{r}(a_i * F + A_i * f) * \overline{(a_i * F + A_i * f)} \tag{25}$$

$$= (1/r)\sum_{i=1}^{r}(a_i * \overline{a_i}) * (F * \overline{F}) + (A_i * \overline{A_i}) * (f * \overline{f}) + \text{other terms} . \tag{26}$$

The "other terms" will converge to 0, since A and a are uniformly distributed at random modulo q and, though dependent, have small statistical correlation. (See Remark 3 below.) The explicit portion of the average will converge essentially to a scalar multiple of $f * \overline{f} + F * \overline{F}$, for the same reasons as discussed in Section 4.3. Thus,

$$\lim Avg_{ff}(r) = \gamma(f * \overline{f} + F * \overline{F}) . \tag{27}$$

Because the signatures in a transcript are random variables, the limit converges as $1/\sqrt{r}$ where r is the length of a transcript. We may use this averaging to obtain a sufficiently close approximation of $f * \overline{f} + F * \overline{F}$ to obtain the exact value by solving the CVP in the $(N + 1)$-dimensional lattice given in Equation 9 using lattice reduction. Thus, we recover a polynomial that is quadratic in the private key, and we can obtain $f * \overline{g} + F * \overline{G}$ and $g * \overline{g} + G * \overline{G}$ in a similar fashion.

Remark 3. One may artificially construct situations where $(1/r)\sum_{i=1}^{r} a_i * \overline{A_i}$ does not converge to 0. For example, if we let $f = F$, then $A_i * f + a_i * F \equiv 0 \pmod q$ basically implies $A_i = -a_i$ and hence $a_i * \overline{A_i} = -a_i * \overline{a_i}$, the average of which does not converge to 0. Conceivably, NTRUSign could be modified so as to constrain $f^{-1} * F \pmod q$, but this would likely allow alternative attacks.

It is worth noting that these second order polynomials give us the Gram matrix $B_{priv}^T \cdot B_{priv}$:

$$B_{priv}^T \cdot B_{priv} = \begin{bmatrix} M_{\overline{f}} & M_{\overline{F}} \\ M_{\overline{g}} & M_{\overline{G}} \end{bmatrix} \begin{bmatrix} M_f & M_g \\ M_F & M_G \end{bmatrix} = \begin{bmatrix} M_{f*\overline{f}+F*\overline{F}} & M_{g*\overline{f}+G*\overline{F}} \\ M_{f*\overline{g}+F*\overline{G}} & M_{g*\overline{g}+G*\overline{G}} \end{bmatrix} .$$

This Gram matrix gives us the "shape" of the parallelepiped defined by B_{priv}^T, but the "orientation" of this parallelepiped is unclear. An interesting (and open) question is: Can an attacker recover B_{priv} from $B_{priv}^T \cdot B_{priv}$ and $B_{pub} = U \cdot B_{priv}$, where U is a unimodular matrix? We answered a similar question in the affirmative in Section 7; we showed that an attacker, in polynomial time, can recover M_f from $M_f^T \cdot M_f$ and $U \cdot M_f$, where U is a unimodular matrix. We have not found a way to extend the orthogonal congruence attack to solve the NTRUSign Gram matrix problem, however, where the bi-circulant (rather than purely circulant) nature of the matrices in question (such as B_{priv}) destroys the commutativity that our orthogonal congruence attack appears to require, but this does not imply that the NTRUSign Gram matrix problem is necessarily hard. We note that it would more than suffice to find an algorithm that factors $U \cdot U^T$ for unimodular U. (This factorization is unique up to a signed permutation matrix.) Further research in this area would be interesting, if only because it is relevant to NTRUSign's security.

9.2 Second Order Subtranscript Attack

It is clear that the second order polynomials recovered above contain information not contained in the public key, but using this information to create an effective attack is not so straightforward. Our approach has been to use the second order polynomials to recover, say, $f * \overline{f}$, so that we may then apply the orthogonal congruence attack to recover f. One way to get $f * \overline{f}$ is to use the following subtranscript attack.

First, we notice that since $\|F\| > \|f\|$, the norm $\|A * f + a * F\|$ is dictated more by $\|a\|$ than by $\|A\|$. More relevantly, for our purposes, an $A * f + a * F$ that is longer than normal will usually have $\|a\| > \|A\|$. This suggests a subtranscript attack, including in Equation 25 only those polynomials $A_i * f + a_i * F$ for which $\|A_i * f + a_i * F\|$ is greater than some bound.[9] Then, we have:

$$\text{Subtranscript: } \lim Avg_{ff}(r) = \gamma_1(f * \overline{f}) + \gamma_2(F * \overline{F}) \ , \tag{28}$$

for $\gamma_1 < \gamma_2$. Since this linear combination of $f * \overline{f}$ and $F * \overline{F}$ is distinct from that in Equation 27, we may compute $f * \overline{f}$ and $F * \overline{F}$. The convergence of this subtranscript averaging will be affected by the proportional size of the subtranscript, but more importantly, by the fact that γ_1 may be only a few percentage points greater than γ_2 (in our preliminary experiments using the longest 50% of the $A_i * f + a_i * F$'s). Further experiments are necessary to determine the effectiveness of this attack. Another consideration is that in [11], a possible modification of NTRUSign was proposed in which one chooses the transpose of B_{priv} to be the private key. The basis vectors are then (f, F) and (g, G) with $\|f\| \approx \|g\|$ and $\|F\| \approx \|G\|$. Choosing the private basis in this way appears to defeat this subtranscript attack.

[9] This selection criterion might be refined, as by also considering the norm of $A * \overline{F} - a * \overline{f}$, which may be computed using the above second order polynomials.

9.3 Fourth Order Attack

An alternative way to get $f * \overline{f}$ is to use the following fourth order attack. Viewing the average in Equation 25, one may consider the corresponding variance and conclude, under the assumption of the statistical independence of a and A, that:

$$\lim_{r\to\infty}(1/r)\sum_{i=1}^{r}(s * s_{rev})^2 - (1/r)\beta(\lim_{r\to\infty}\sum_{i=1}^{r}(s * s_{rev}))^2 = \gamma f * \overline{f} * F * \overline{F} \ . \quad (29)$$

The adjustment value of β depends on the scheme parameters n and q, and so the above value may not be exactly the variance. The factor γ also depends on the scheme parameters, and is a constant that slows convergence by a factor $\frac{1}{\gamma^2}$. This limit does converge more slowly than the second order averaging, but, as above, we may use a close approximation in conjunction with the lattice of Equation 9 to obtain the exact value. Preliminary tests show that it may not be practical to obtain an error lower than the Gaussian estimate with a reasonable number of signatures. Assuming we do obtain the value $f * \overline{f} * F * \overline{F}$, we may use it in combination with $(f * \overline{f} + F * \overline{F})^2$ to obtain $(f * \overline{f} - F * \overline{F})^2$, and then $f * \overline{f} - F * \overline{F}$ (using, perhaps, the algorithm given in Section 7.5). Then, $f * \overline{f} + F * \overline{F}$ and $f * \overline{f} - F * \overline{F}$ give us $f * \overline{f}$ and $F * \overline{F}$.

9.4 Preliminary Conclusion

The approach of these initial attacks was to reduce the breaking problem to the orthogonal congruence attack using the results of various averagings. We showed how this could be done, but the practical feasibility of these attacks has yet to be determined. In our experiments, we have found that the second order attack is feasible; for example, by averaging 20000 signatures, an attacker may obtain an approximation whose squared error is about 88, about 1/20 of the squared Gaussian heuristic of the $(N + 1)$-dimensional CVP lattice that would be used to correct this approximation. Although we have not tested this CVP lattice, we believe its reduction would be feasible. Alternatively, more signatures could first be used to obtain a better approximation. We have also shown how the security of NTRUSign rests on the hardness of several new hard problems. These attacks will continue to be analyzed by the authors, NTRU corporation, and the cryptographic community.

10 Acknowledgments

The authors would like to thank Burt Kaliski, Alice Silverberg and Yiqun Lisa Yin for helpful discussions, Jakob Jonsson, Phong Nguyen, and Jacques Stern for discussions and collaboration on the precursor of this article, and Jeffrey Hoffstein, Nick Howgrave-Graham, Jill Pipher, Joseph Silverman and William Whyte, who have given us valuable feedback on our cryptanalysis, particularly on our preliminary cryptanalysis of NTRUSign.

References

1. M. Ajtai, *The shortest vector problem in L_2 is NP-hard for randomized reductions*, in Proc. 30th ACM Symposium on Theory of Computing, 1998, 10–19.
2. H. Cohen, A Course in Computational Algebraic Number Theory, Graduate Texts in Mathematics, 138. Springer, 1993.
3. H. Cohen, *Advanced Topics in Computational Number Theory*, Graduate Texts in Mathematics 138 ,1993.
4. Consortium for Efficient Embedded Security. Efficient Embedded Security Standard (EESS) # 1: Draft 1.0. Previously on `http://www.ceesstandards.org`.
5. Consortium for Efficient Embedded Security. Efficient Embedded Security Standard (EESS) # 1: Draft 2.0. Previously on `http://www.ceesstandards.org`.
6. Consortium for Efficient Embedded Security. Efficient Embedded Security Standard (EESS) # 1: Draft 3.0. Available from `http://www.ceesstandards.org`.
7. D. Coppersmith and A. Shamir, *Lattice Attacks on NTRU*, in Proc. of Eurocrypt '97, LNCS 1233, pages 52–61. Springer-Verlag, 1997.
8. C. Gentry, *Key Recovery and Message Attacks on NTRU-Composite*, in Proc. of Eurocrypt '01, LNCS 2045, pages 182–194. Springer-Verlag, 2001.
9. C. Gentry, J. Jonsson, J. Stern, M. Szydlo, *Cryptanalysis of the NTRU signature scheme*, in Proc. of Asiacrypt '01, LNCS 2248, pages 1–20. Springer-Verlag, 2001.
10. O. Goldreich, S. Goldwasser, S. Halevi, *Public-key Cryptography from Lattice Reduction Problems*, in Proc. of Crypto '97, LNCS 1294, pages 112–131. Springer-Verlag, 1997.
11. J. Hoffstein, N. Howgrave-Graham, J. Pipher, J.H. Silverman, W. Whyte, *NTRUSign: Digital Signatures Using the NTRU Lattice*, December, 2001. Available from `http://www.ntru.com`.
12. J. Hoffstein, B.S. Kaliski, D. Lieman, M.J.B. Robshaw, Y.L. Yin, *Secure user identification based on constrained polynomials*, US Patent 6,076,163, June 13, 2000.
13. J. Hoffstein, D. Lieman, J.H. Silverman, *Polynomial Rings and Efficient Public Key Authentication*, in Proc. International Workshop on Cryptographic Techniques and E-Commerce (CrypTEC '99), Hong Kong, (M. Blum and C.H. Lee, eds.), City University of Hong Kong Press.
14. J. Hoffstein, J. Pipher, J.H. Silverman, *Enhanced Encoding and Verification Methods for the NTRU Signature Scheme*, NTRU Technical Note #017, May 2001. Available from `http://www.ntru.com`.
15. J. Hoffstein, J. Pipher, J.H. Silverman. *Enhanced encoding and verification methods for the NTRU signature scheme (ver. 2)*, May 30, 2001. Available from `http://www.ntru.com`.
16. J. Hoffstein, J. Pipher, J.H. Silverman, *NSS: The NTRU Signature Scheme*, preprint, November 2000. Available from `http://www.ntru.com`.
17. J. Hoffstein, J. Pipher, J.H. Silverman, *NSS: The NTRU Signature Scheme*, in Proc. of Eurocrypt '01, LNCS 2045, pages 211–228. Springer-Verlag, 2001.
18. J. Hoffstein, J. Pipher, J.H. Silverman, *NSS: The NTRU Signature Scheme: Theory and Practice*, preprint, 2001. Available from `http://www.ntru.com`.
19. A.K. Lenstra, H.W. Lenstra Jr., L. Lovász, *Factoring Polynomials with Rational Coefficients*, Mathematische Ann. 261 (1982), 513–534.
20. A. May, *Cryptanalysis of NTRU-107*, preprint, 1999. Available from `http://www.informatik.uni- frankfurt.de/~alex/crypto.html`.
21. I. Mironov, *A Note on Cryptanalysis of the Preliminary Version of the NTRU Signature Scheme*, IACR preprint server, `http://eprint.iacr.org/2001/005.`

22. P. Nguyen and J. Stern, *Lattice Reduction in Cryptology: An Update*, in Proc. of Algorithm Number Theory (ANTS IV), LNCS 1838, pages 85–112. Springer-Verlag, 2000.
23. J.H. Silverman, *Estimated Breaking Times for NTRU Lattices*, NTRU Technical Note #012, March 1999. Available from `http://www.ntru.com`.
24. J.H. Silverman, *Invertibility in Truncated Polynomial Rings.*, NTRU Technical Note #009, October 1998. Available from `http://www.ntru.com`.
25. L. Washington, *Introduction to Cyclotomic Fields*, Graduate Texts in Mathematics 83, 1982.

A Orthogonal Congruence Attack: The Key Algorithm

Here, we present in pseudocode the key algorithm of the orthogonal congruence attack, through which an attacker computes chains of polynomials that allow him to derive modular information about f^{P-1}. This algorithm, as well as other algorithms from the orthogonal congruence attack, are discussed in more detail in the full version of the paper. Note that this algorithm will compute $f^{2^r} * \overline{v_r} \pmod P$ instead of $f^{P-1} * \overline{v_r} \pmod P$ for $r = \lfloor \log_2 P \rfloor$; more general powers require only minor adjustments. Let $v_0 = f$.

Input: $v_0 * \overline{v_0}$; basis B_0 of (v_0); prime $P \equiv 1 \pmod N$, $r = \lfloor \log_2 P \rfloor = (N+1)/2$.
Output: $\{v_0^2 * \overline{v_1}, \ldots, v_{r-1}^2 * \overline{v_r}\}$ and $\{v_0 * \overline{v_0}, \ldots, v_{r-1} * \overline{v_{r-1}}\}$ with $\|v_i\| < 2^{(N-1)/2}$; $v_0^{2^r} * \overline{v_r} \pmod P$.

1. Set $r' := 0$.
2. While $r' < r$ do
   (a) Use $B_{r'}$ to construct a set G of vectors generating $(v_{r'}^2)$. Let $G = H \cdot M_{v_{r'}^2}$.
   (b) Compute the Gram matrix $H \cdot H^T = G \cdot M^{-2}_{v_{r'} * \overline{v_{r'}}} \cdot G^T$.
   (c) Reduce the Gram matrix $H \cdot H^T$ to get $(A \cdot H) \cdot (A \cdot H)^T$ for known A.
   (d) Compute the basis $(A \cdot H) \cdot M_{v_{r'}^2} = A \cdot G$ of $(v_{r'}^2)$. ($A \cdot H$ has short rows.)
   (e) Pick, say, the jth row of $A \cdot G$, call it $v_{r'}^2 * \overline{v_{r'+1}}$. Note: $\overline{v_{r'+1}}$ is the jth row of $A \cdot H$.
   (f) Output $v_{r'}^2 * \overline{v_{r'+1}}$.
   (g) If $r' + 1 \neq r$
   – Compute $v_{r'+1} * \overline{v_{r'+1}} = (v_{r'}^2 * \overline{v_{r'+1}}) * \overline{(v_{r'}^2 * \overline{v_{r'+1}})} * (v_{r'} * \overline{v_{r'}})^{-2}$.
   – Output $v_{r'+1} * \overline{v_{r'+1}}$.
   – Compute $(A \cdot H) \cdot M_{v_{r'+1}} = A \cdot G \cdot M^{-2}_{v_{r'} * \overline{v_{r'}}} \cdot M_{\overline{v_{r'}}^2 * v_{r'+1}}$. This is our basis $B_{r'+1}$ of $(v_{r'+1})$.
   (h) Increment r'.
3. Set $r' := 1$; Set $y := v_0^2 * \overline{v_1} \pmod P$.
4. While $r' < r$ do
   (a) Compute $y := y^2 \pmod P$.
   (b) Compute $y := y * (v_{r'} * \overline{v_{r'}})^{-2} * (v_{r'}^2 * \overline{v_{r'+1}}) \pmod P$. At this point, $y = v_0^{2^{r'+1}} * \overline{v_{r'+1}} \pmod P$.
   (c) Increment r'.
5. Output y.