Proceedings of Symposia in Applied Mathematics

Towards Faster Cryptosystems, II

William D. Banks

Abstract. We discuss three cryptosystems, *NTRU*, *SPIFI*, and *ENROOT*, that are based on the use of polynomials with restricted coefficients.

Introduction

In this paper, we consider three cryptosystems that are based on the use of polynomials with restricted coefficients and over a finite ring $\mathcal{R}$. For cryptographic purposes, polynomials with restricted coefficients offer many attractive features, including resistance to lattice attacks and other security risks, very modest storage requirements, and competitive computational speed. The cryptosystems described below, namely *NTRU*, *SPIFI*, and *ENROOT*, utilize polynomials with restricted coefficients in different ways, and these constructions have met with varying levels of success in the cryptographic community. Our aim in this paper is not to imply that these particular cryptosystems are optimal in any sense. Instead, we simply hope to illustrate the elegance and simplicity of their design features and perhaps promote the development of the next generation of fast cryptosystems based on polynomials.

1. The NTRU Cryptosystem

The *NTRU Public Key Cryptosystem* is based on ring theory, and its underlying security relies on the difficulty of solving certain lattice problems when the dimension of the given lattice is sufficiently large. In practical implementations of the NTRU protocol, many of the polynomials that are used have coefficients lying in the set $\{0, \pm 1\}$; this means that multiplication in the underlying ring $\mathcal{R}$ can be performed very efficiently, which contributes significantly to the overall performance. Thus, NTRU provides a competitive alternative to cryptosystems based on RSA or Diffie-Hellman. For a more complete account of NTRU, the reader is referred to original paper [**11**] and other related articles that can be found at the NTRU website (`http://www.ntru.com`).

1991 *Mathematics Subject Classification.* Primary ?????, ?????; Secondary ?????, ?????
Key words and phrases. Cryptography, polynomials, number theory.
The author was supported in part by NSF Grant # DMS-0070628.

1.1. The NTRU Protocol. The original version of NTRU depends on three integer parameters (N, p, q) and four sets $\mathcal{L}_f$, $\mathcal{L}_g$, $\mathcal{L}_\phi$ and $\mathcal{L}_m$ of polynomials with integers coefficients and degree at most $N-1$, which are naturally viewed as subsets of the ring $\mathcal{R} = \mathbb{Z}[x]/(x^N - 1)$. The symbol $*$ is used to denote the operation of multiplication in the ring $\mathcal{R}$. The parameters p and q need not be prime numbers (indeed, q is often chosen to be a power of 2 for obvious reasons), but one does require that $\gcd(p, q) = 1$. The basic NTRU protocol consists of the following three steps:

- **Key Creation.** *Bob* creates a public key h by selecting two polynomials $f \in \mathcal{L}_f$ and $g \in \mathcal{L}_g$. The polynomial f should satisfy the additional requirement that it have inverses modulo p and modulo q. These are denoted by F_p and F_q, respectively, so that

  $$f * F_p \equiv 1 \pmod{p} \qquad \text{and} \qquad f * F_q \equiv 1 \pmod{q}.$$

  Bob's public key is the polynomial

  $$h \equiv F_q * g \pmod{q}.$$

  Bob only needs to retain f as his private key, although in practice *Bob* would also want to store the polynomial F_p since this is needed for every decryption.

- **Encryption.** To encrypt a message $m \in \mathcal{L}_m$ to *Bob*, *Alice* randomly chooses a polynomial $\phi \in \mathcal{L}_\phi$. Using *Bob*'s public key h, she then computes

  $$e = p\phi * h + m \pmod{q}$$

  and transmits e to *Bob*.

- **Decryption.** *Bob* has received e from *Alice*. To decrypt, *Bob* first computes

  $$a = f * e \pmod{q},$$

  choosing the coefficients of a to lie in the interval between $-q/2$ and $q/2$. *Bob* then recovers the message m by computing

  $$m = F_p * a \pmod{p}.$$

To see why the decryption works, we observe that in the last step, *Bob* computes the polynomial

$$\begin{aligned} a = f * e \pmod{q} &\equiv f * (p\phi * h + m) \pmod{q} \\ &\equiv p\phi * (f * F_q) * g + f * m \pmod{q} \\ &\equiv p\phi * g + f * m \pmod{q}. \end{aligned}$$

For appropriate choices of the input parameters, the polynomial

$$G = p\phi * g + f * m \in \mathcal{R}$$

will (almost certainly) have coefficients that all lie in the interval $-q/2$ to $q/2$, hence *Bob* recovers exactly the polynomial G after completing the first calculation.

Reducing G modulo p, *Bob* recovers the polynomial $f * m \pmod p$. Then, after multiplying by F_p (the inverse of f modulo p), he obtains *Alice*'s message m.

We remark that the protocol described here has been generalized to work over an arbitrary ring $\mathcal{R}$. In this more general setting, the parameters p and q can be any ideals in $\mathcal{R}$ such that $(p, q) = (1)$.

1.2. Parameter Selection and Efficiency. For the most current recommended parameter sets for NTRU, and an overview of the security analysis, we refer the reader to [**14**].

In practical implementations of NTRU, the most time consuming part of the encryption and decryption steps is multiplication in the ring $\mathcal{R}$; encryption involves the computation of one product $r * h \pmod q$, while decryption requires that two products $a = f * e \pmod q$ and $F_p * a \pmod p$ be computed. In the key creation step, the most time consuming part is the computation of the inverses F_p and F_q. Ordinarily, *Bob* will store F_p along with his private key f in order to avoid the recalculation of F_p each time an encrypted message is sent. To speed up both the key creation and encryption, *Bob* can choose the element f to have the special form $f = 1 + pf_1$ with $f_1 \in \mathcal{R}$. In this case, $F_p = 1$, so it is not necessary to compute the inverse of f modulo p in the key creation step, and the second multiplication in the decryption step is eliminated. This also eliminates the need to store F_p. However, if *Bob* does choose to work with elements f of this form, then since the underlying lattice is changed, special care is required in the parameter selection in order to avoid to potential lattice attacks.

The space of messages $\mathcal{L}_m$ consists of all polynomials modulo p; if p is odd, it is generally convenient to choose

$$\mathcal{L}_m = \{m \in \mathcal{R} \,|\, m \text{ has coefficients lying between } -(p-1)/2 \text{ and } (p-1)/2\},$$

while for $p = 2$ or $p = 2 + X$, $\mathcal{L}_m$ consists of polynomials with $\{0, 1\}$ coefficients. The other sample spaces $\mathcal{L}_f$, $\mathcal{L}_g$ and $\mathcal{L}_\phi$ also depend on the form of p, and are generally chosen to have the form

$$\mathcal{L}_f = \mathcal{L}(d_f), \qquad \mathcal{L}_g = \mathcal{L}(d_g), \qquad \mathcal{L}_\phi = \mathcal{L}(d),$$

or

$$\mathcal{L}_f = \mathcal{L}(d_f, d_f - 1), \qquad \mathcal{L}_g = \mathcal{L}(d_g, d_g), \qquad \mathcal{L}_\phi = \mathcal{L}(d, d),$$

where for all $d_1, d_2 \geq 1$, $\mathcal{L}(d_1)$ consists of all polynomials $F \in \mathcal{R}$ with exactly d_1 coefficients equal to $+1$ and the remaining coefficients equal to 0, and $\mathcal{L}(d_1, d_2)$ consists of all polynomials $F \in \mathcal{R}$ with exactly d_1 coefficients equal to $+1$, exactly d_2 coefficients equal to -1, and the remaining coefficients equal to 0.

We observe that any element of $\mathcal{R}$ modulo p can be encoded with about $N \log p$ bits. Indeed, to encode the coefficient at each position requires about $\log p$ bits, and there are N position altogether. Hence, the private key length is about $2N \log p$ bits. Similarly, the public key length is about $N \log q$ bits. For security equivalent to 1024-bit RSA (or a symmetric cipher with an 80-bit key), the following choices have been recommended:

$$(N, p, q) = (251, 2 + X, 128), \quad \mathcal{L}_f = \mathcal{L}(72), \quad \mathcal{L}_g = \mathcal{L}(72), \quad \mathcal{L}_\phi = \mathcal{L}(72),$$

which gives a private key size of 251 bits and a public key size of 1757 bits (further optimizations reduce the private key size to 192 bits; see [**13**]). For security equivalent to 2048-bit RSA (110-bit symmetric), the following choices are recommended:

$$(N, p, q) = (347, 2 + X, 256), \quad \mathcal{L}_f = \mathcal{L}(64), \quad \mathcal{L}_g = \mathcal{L}(173), \quad \mathcal{L}_\phi = \mathcal{L}(64),$$

which after optimization has a private key size of 184 bits and a public key size of 2776 bits. For security equivalent to 4096-bit RSA (160-bit symmetric), the following choices are recommended:

$$(N, p, q) = (503, 2 + X, 256), \quad \mathcal{L}_f = \mathcal{L}(420), \quad \mathcal{L}_g = \mathcal{L}(251), \quad \mathcal{L}_\phi = \mathcal{L}(170),$$

which has a private key size of 480 bits and a public key size of 4024 bits.

We remark that by working mostly with polynomials whose coefficients lie in $\{0, 1\}$ or $\{0, \pm 1\}$, multiplication in $\mathcal{R}$ is very fast and efficient. Hence, even the most time consuming aspect of the NTRU protocol is greatly improved by the use of polynomials with restricted coefficients.

1.3. Some Security Issues. Here we briefly discuss a few issues of security; for a complete analysis of NTRU security, the reader is referred to [**14**] (see also [**11**]).

1.3.1. *Brute force attacks.* To recover the private key f, an attacker *Attila* can apply a brute force attack by trying all possible $f \in \mathcal{L}_f$ and testing whether $f * h \pmod q$ has small entries (i.e., lies in $\mathcal{L}_g$). Alternatively, *Attila* can try all possible $g \in \mathcal{L}_g$ and test whether $g * h^{-1} \pmod q$ has small entries (i.e., lies in $\mathcal{L}_f$). In practice one usually has $\#\mathcal{L}_g < \#\mathcal{L}_f$, so the key security is determined by $\#\mathcal{L}_g$, while the individual message security is determined by $\#\mathcal{L}_\phi$. There is a meet-in-the-middle attack due to Odlyzko, which cuts the search time by the usual square root (assuming a sufficient amount of storage); thus one finds that

$$\begin{aligned} \text{Key Security} &= \sqrt{\#\mathcal{L}_g} = \frac{1}{d_g!}\sqrt{\frac{N!}{(N-2d_g)!}}, \\ \text{Message Security} &= \sqrt{\#\mathcal{L}_\phi} = \frac{1}{d!}\sqrt{\frac{N!}{(N-2d)!}}. \end{aligned}$$

For instance, in a moderate security implementation of NTRU, with $N = 107$, $d_g = 12$ and $d = 5$, the key security is about 2^{50}, while the individual message security is about $2^{26.5}$.

1.3.2. *Multiple transmission attacks.* If *Alice* sends the same message m several times using the same public key h but different random ϕ's, the attacker *Attila* will be able to recover a large part of the message. For suppose *Alice* transmits $e_j \equiv \phi_j * h + m \pmod q$ for $j = 1, \ldots, r$. *Attila* can then compute $(e_j - e_1) * h^{-1} \pmod q$, obtaining $(\phi_j - \phi_1) \mod q$ for each j. But the coefficients of the ϕ's are so small that *Attila* recovers exactly $\phi_j - \phi_1$, and he can use this to recover many of the coefficients of ϕ_1. If r is of moderate size, then *Attila* will recover enough of ϕ_1 to be able to test the remaining coefficients by brute force, and therefore he can recover m. Thus, multiple transmissions are not advised.

1.3.3. *Malleability and information leakage.* The multiple transmission attack mentioned above is one of many similar attacks on "raw" NTRU. For example, encrypted messages are trivially *malleable*: an attacker can add or subtract 1 to any coefficient of e and there is a good chance that the decrypted m will have 1

added to or subtracted from the same coefficient. In addition, there is at least one bit of *information leakage*: due to the existence of the ring homomorphism

$$(a * b)(1) = a(1)b(1) \quad \forall a, b \in \mathcal{R},$$

a ciphertext e leaks $m(1)$ through $m(1) = e(1) - \phi(1)h(1)$ (note that in the parameter sets above, $\phi(1)$ is a public parameter of the system). A sensible encryption scheme must prevent this type of attack.

Therefore, to insure security, the NTRU primitive must be combined with a secure message processing scheme (similar to the way RSA must be combined with OAEP). In this scheme, *Alice* must combine m with some one-time randomness b in some reversible way to obtain the blinded message m' (this will ensure that $m(1)$ is hidden, and that even if the same m is encrypted twice, a different m' will be encrypted every time). Then she uses m and b to derive ϕ, using a hash function which can be guaranteed to remove mathematical structure from the relationship between m' and ϕ. Finally, she calculates the ciphertext $e = p\phi * h + m'$. On decryption, *Bob* recovers m', extracts m and b from it, regenerates ϕ and checks that $p\phi * h + m'$ is equal to the received e. This construction prevents both information leakage and the adaptive chosen ciphertext attacks described in lecture 4.

1.3.4. *Lattice attacks.* Very briefly, NTRU's security against lattice attacks is derived from the large dimension of the underlying lattice. Initial experiments using lattice reduction techniques such as LLL indicate that the time $t(N)$ needed to break an NTRU cryptosystem with parameter N grows like $N \log N$, i.e., faster than exponential. By extrapolating $t(N)$ from small values of N, one obtains the estimated breaking times and equivalent security levels given in section 1.2.

2. The SPIFI Identification Scheme

In this section, we describe the *SPIFI identification scheme*, we can be defined for an arbitrary finite ring $\mathcal{R}$; see [**2**] and [**3**]. The polynomials used in the SPIFI protocol are taken to have only a few nonzero coefficients, hence particular values of these polynomials can be computed very rapidly, and the storage requirements remain quite modest.

2.1. Notation and Definitions. Throughout this section and the next, $\mathcal{R}$ denotes an arbitrary finite (unitary) ring with $\mathcal{R}^\times$ its multiplicative group of units. We denote by N a fixed multiple of the exponent $\exp(\mathcal{R}^\times)$ of $\mathcal{R}^\times$; then $a^{N+1} = a$ for all $a \in \mathcal{R}^\times \cup \{0\}$. We assume that any element of $\mathcal{R}$ can be encoded using about $\log(\#\mathcal{R})$ bits, where as usual $\log x$ denotes the binary logarithm of $x > 0$.

To illustrate our ideas, we occasionally consider two important special cases referred to as the "$\mathbb{F}_q$-case" and the "$\mathbb{Z}_M$-case." In the $\mathbb{F}_q$-case, $\mathcal{R} = \mathbb{F}_q$ is the finite field q elements, where q is a fixed prime power; in this case, one can take $N = \exp(\mathcal{R}^\times) = q - 1$. In the $\mathbb{Z}_M$-case, $\mathcal{R} = \mathbb{Z}/M\mathbb{Z}$ is the ring of congruence classes with respect to a fixed RSA modulus $M = p\ell$, where p and ℓ are (large) prime numbers; in this case, one can either take $N = \varphi(M) = (p-1)(\ell-1)$, where φ is the *Euler function*, or $N = \lambda(M) = \mathrm{lcm}(p-1, \ell-1)$, where λ is the *Carmichael function*. With any of the choices above, we have $\log(\#\mathcal{R}) \approx \log(\#\mathcal{R}^\times) \approx \log N$.

Let d be a fixed positive integer. For any fixed subset $\mathcal{S} \subseteq \mathcal{R}$, we say that a polynomial $f(x_1, \ldots, x_d) \in \mathcal{R}[x_1, \ldots, x_d]$ is an *$\mathcal{S}$-polynomial* if every coefficient of f belongs to $\mathcal{S}$. Any expression of the form $ax_1^{e_1} \ldots x_d^{e_d}$ is said to be *monomial*

with *coefficient* a and *exponent* $(e_1, \dots, e_d)$. Finally, we say that a polynomial $f(x_1, \dots, x_d) \in \mathcal{R}[x_1, \dots, x_d]$ is *τ-sparse* if f has at most τ nonzero coefficients.

2.2. A Hard Problem. The hard problem underlying the SPIFI scheme can be stated as follows:

> *Given $2m$ arbitrary elements $\alpha_1, \dots \alpha_m, \beta_1, \dots, \beta_m \in \mathcal{R}$ and a set $\mathcal{S} \subseteq \mathcal{R}$ of small cardinality, it is not feasible to find a τ-sparse $\mathcal{S}$-polynomial $f(x) \in \mathcal{R}[x]$ of degree $\deg(f) \le N$ with $f(\alpha_j) = \beta_j$ for each $1 \le j \le m$, provided that N is of "medium" size relative to the choices of $m \ge 1$, the cardinality $\#\mathcal{S}$, and the parameter $\tau \ge 3$.*

More precisely, one expects that if m, $\mathcal{S}$ and τ are fixed, then the problem requires exponential time as $N \to \infty$.

For example, let p be a prime, and consider the case where $\mathcal{R}$ is the finite field $\mathbb{F}_p$ with p elements. Let $a_{ij} \equiv \alpha_j^i \pmod p$ and $b_j \equiv \beta_j \pmod p$ be chosen so that $0 \le a_{ij}, b_j \le p-1$ for $i = 0, \dots, p-1$ and $j = 1, \dots, m$. Then in this simplified situation, the hard problem above is still equivalent to the problem of finding a feasible solution to the *integer programming problem*

$$\sum_{i=0}^{p-1} x_i \varepsilon_i a_{ij} + y_j p = b_j, \qquad j = 1, \dots, m, \qquad \sum_{i=0}^{p-1} \varepsilon_i \le \tau,$$

where $y_j \in \mathbb{Z}$, $x_i \in \mathcal{S}$, and $\varepsilon_i \in \{0, 1\}$ for all i and j.

2.3. The SPIFI Protocol. We fix the ring $\mathcal{R}$ and some integer parameters $k \ge 1$ and $r, s, t \ge 3$; this information is made *public*. The value of N is kept *private*. Since only *Alice* needs to know this value, the choice of the ring $\mathcal{R}$ and the value of N can be made by *Alice*.

To avoid a certain well known attack on the scheme (see [**1**] and the comments below), we also require that $\mathcal{R}$ contains elements of multiplicative order in the interval $[0.5N^{1/4}, 2N^{1/4}]$. Although this imposes certain number theoretic limitations on the ring $\mathcal{R}$ and the integer N, in practice these constraints are easily satisfied.

The SPIFI protocol consists of an initial set-up, or signature creation, and a verification procedure.

2.3.1. *SPIFI Signature Creation.* To create a digital signature, *Alice* uses the following algorithm:

- **Step 1.** *Alice* selects at random k distinct elements $a_0, \dots a_{k-1} \in \mathcal{R}^\times$, where a_0 has a multiplicative order in the interval $[0.5N^{1/4}, 2N^{1/4}]$.

- **Step 2.** *Alice* selects a random $\lceil t/2 \rceil$-sparse $\{0, 1\}$-polynomial $f_1(x) \in \mathcal{R}[x]$ with $\deg(f_1) \le N$ and $f_1(a_0) \in \mathcal{R}^\times$. Next, she selects at random a $\lfloor t/2 \rfloor$-sparse $\{0, 1\}$-polynomial $f_2(x) \in \mathcal{R}[x]$ with $\deg(f_2) \le N$, $f_2(a_0) \ne 0$ and $f_2(a_0) \ne -f_1(a_0)$.

- **Step 3.** *Alice* computes the quantity $A = -f_2(a_0)f_1(a_0)^{-1}$ and creates the polynomial $f(x) = Af_1(x) + f_2(x)$. Then f is a t-sparse $\{0, 1, A\}$-polynomial with $\deg(f) \leq N$, and $f(a_0) = 0$. The polynomial f is *Alice*'s private key.

- **Step 4.** *Alice* computes the numbers $C_j = f(a_j)$ for $j = 1, \ldots, k-1$.

- **Step 5.** *Alice* publishes the set of values $\{A, a_0, \ldots a_{k-1}, C_1, \ldots, C_{k-1}\}$ as her public key.

2.3.2. *SPIFI Verification Protocol.* In order for *Alice* to verify her identity to *Bob*, the following algorithm is used:

- **Step 1.** (Commitment) *Alice* selects a random r-sparse $\{0, 1\}$-polynomial $g(x) \in \mathcal{R}[x]$ with $\deg(g) \leq N$ and $g(0) = 0$. She then computes the numbers
$$D_j = g(a_j), \qquad j = 1, \ldots, k-1,$$
and sends the sum $D = D_1 + \ldots + D_{k-1}$ to *Bob*.

- **Step 2.** (Challenge) *Bob* selects a random s-sparse $\{0, 1, B\}$-polynomial $h(x) \in \mathcal{R}[x]$ of degree $\deg(h) \leq N$ and sends h to *Alice*. Here $B \notin \{0, 1, A\}$.

- **Step 3.** (Response) *Alice* computes
$$F(x) \equiv f(x)g(x)h(x) \pmod{x^{N+1} - x}$$
and sends the polynomial F and $\{D_1, \ldots, D_{k-1}\}$ to *Bob*.

- **Step 4.** (Verification) *Bob* computes
$$E_j = h(a_j), \qquad j = 1, \ldots, k-1,$$
and verifies that
$$D_1 + \ldots + D_{k-1} = D$$
that $F(x)$ is an rst-sparse $\{0, 1, A, B, AB\}$-polynomial of degree at most N, that $F(a_0) = 0$, and that $F(a_j) = C_j D_j E_j$ for $j = 1, \ldots, k-1$.

Of course, there is a chance that the constructed polynomial $F(x)$ is not a $\{0, 1, A, B, AB\}$-polynomial due to collisions; however, if rst is substantially smaller than N, then this chance is negligible (and in this case, *Alice* and *Bob* can repeat the procedure).

2.4. Efficiency. As mentioned earlier, the sparsity of the polynomials plays a dual role in that it not only guarantees the computational efficiency of this scheme, but it also means that the storage requirements are minimal.

Using only (naive) repeated squaring, one can compute the power a^e for any $a \in \mathcal{R}$ and $0 \leq e \leq N$ in about $2 \log N$ arithmetic operations in $\mathcal{R}$ in the worst case, or in about $1.5 \log N$ arithmetic operations "on average"; see Section 1.3 of [**4**], Section 4.3 of [**5**], or Section 2.1 of [**6**]. Consequently, any τ-sparse polynomial

$f(x) \in \mathcal{R}[x]$ of degree at most N can be evaluated at any point using only $O(\tau \log N)$ arithmetic operations in $\mathcal{R}$.

If $0 \in \mathcal{S} \subseteq \mathcal{R}$, then any τ-sparse $\mathcal{S}$-polynomial $f(x) \in \mathcal{R}[x]$ of degree at most N can be encoded with about $\tau \log(N \cdot \#\mathcal{S} - N)$ bits. To do this, we have to identify at most τ positions at which f has a nonzero coefficient. The encoding of each position requires about $\log N$ bits, and for each such position, about $\log(\#\mathcal{S} - 1)$ bits are then required to determine the corresponding element of $\mathcal{S}$. For example, the signature must encode rst positions of the polynomial F (corresponding to its nonzero coefficients), which takes about $rst \log N$ bits. Each position requires two additional bits to distinguish between the possible nonzero coefficients 1, A, B and AB. The encoding of $D_1, \ldots, D_{k-1}$ and their sum D requires about $k \log(\#R)$ bits. Consequently, the total signature size is about $rst \log(4N) + k \log(\#\mathcal{R})$ bits.

Putting everything together, after simple calculations we derive that (using the naive repeated squaring exponentiation)

- the *initial set-up* takes $O(kt \log N)$ arithmetic operations in $\mathcal{R}$;
- the *private key size* is about $t \log(2N)$ bits;
- the *public key size* is about $k(\log(\#\mathcal{R}) + \log(\#\mathcal{R}^\times))$ bits;
- *signature generation*, that is, computation of the polynomial F, the elements D_j, $j = 1, \ldots, k-1$, and their sum D, takes $O(rst)$ arithmetic operations with integer numbers in the range $[0, 2N]$ and $O\left((k-1)r \log N\right)$ arithmetic operations in $\mathcal{R}$;
- the *signature size* is about $rst \log(4N) + k \log(\#\mathcal{R})$ bits;
- *signature verification*, that is, computation of $D_1 + \ldots + D_{k-1}$, $F(a_j)$ and the products $C_j D_j E_j$, $j = 1, \ldots, k-1$, takes about $O\left(krst \log N\right)$ arithmetic operations in $\mathcal{R}$.

We remark that the practical and asymptotic performance of the SPIFI scheme can be improved if one uses more sophisticated algorithms to evaluate powers and sparse polynomials; see [**4, 5, 6, 16, 18**].

2.5. Some Security Issues. Here we briefly discuss a few issues of security; for a complete analysis of SPIFI security, the reader is referred to [**3**].

2.5.1. *Faking the key.* It is clear that recovering or faking the private key (that is, finding a t-sparse $\{0, 1, A\}$-polynomial polynomial $\widetilde{f}(x) \in \mathcal{R}[x]$ with $\widetilde{f}(a_0) = 0$ and $\widetilde{f}(a_j) = C_j$ for $j = 1, \ldots, k-1$) or faking the signature (that is, finding a rst-sparse $\{0, 1, A, B, AB\}$-polynomial $\widetilde{F}(x) \in \mathcal{R}[x]$ with $\widetilde{F}(a_0) = 0$ and $\widetilde{F}(a_j) = C_j D_j E_j$ for $j = 1, \ldots, k-1$) are both versions of the hard problem mentioned in Section 2.2. Hence the security of the SPIFI scheme relies on the difficulty of solving the hard problem in a computationally feasible manner.

2.5.2. *Factorization attacks.* Note that without the reduction

$$f(x)g(x)h(x) \pmod{x^{N+1} - x},$$

one possible attack might be via polynomial factorization. In any practical implementation of this scheme, one should make sure that both f and g have terms of degree greater than $N/2$ so there are some reductions in the Response step. On the other hand, even without the reduction modulo $x^{N+1} - x$, the factorization attack would not be likely to succeed because of the large degrees of the polynomials involved. All known factorization algorithms (as well as their important components such as irreducibility testing and the greatest common divisor algorithms) do not

take any special advantage of sparsity or structure of the coefficients; see [**5, 17**]. Moreover, the first factor that any of these algorithms would find is the trivial one, i.e., the factor $(x-a_0)$. But the quotient $F(x)/(x-a_0)$ is most likely neither sparse nor an $\mathcal{S}$-polynomial for any small set $\mathcal{S}$. Finally, we remark that if one works in the setting of a ring $\mathcal{R}$ that is *not* a field (such as the $\mathbb{Z}_M$-case), then the problem of factorization becomes much more complicated, so this type of attack is even less likely to succeed.

2.5.3. *Lattice attacks.* One might also consider lattice attacks. In particular, one can try to select a rt-sparse $\{0,1,A\}$-polynomial $e(X) \in \mathcal{R}[x]$ with $e(a_0) = 0$, compute

$$D_j = e(a_j)C_j^{-1}, \qquad j = 1, \ldots, k-1,$$

and then send these values together with

$$F(x) \equiv e(x)h(x) \pmod{x^{N+1} - x}$$

to the verifier. In principal, this attack could succeed since finding such a polynomial e is type of knapsack problem. But since the dimension of the corresponding lattice is equal to the (very large) degree N of the polynomials involved, any such attack seems completely infeasible at this time. With current technology, one can reduce lattices of degrees only in the hundreds, while in any practical implementation of the SPIFI scheme the lattices will have dimension N, which is roughly the size of an RSA modulus.

2.5.4. *Discrete logarithm attacks.* One might also attempt to construct such a polynomial e by solving the discrete logarithm in $\mathcal{R}$ to base a_0 (see [**1**]). But a_0 has been specifically selected to have an order that is small enough to defeat this threat. In fact, the basic SPIFI protocol can be modified slightly to avoid all attacks of this type. Specifically, one requests that for each of the polynomials $f(x)$, $g(x)$ and $h(x)$, the sum of the degrees of the monomials is divisible by the integer N. In this case, the same fact also holds for $F(x)$. Indeed, if an s-sparse polynomial and a t-sparse polynomial have monomials of degrees $n_1, \ldots, n_s$ and $m_1, \ldots, m_t$, respectively, with

$$\sum_{i=1}^{s} n_i \equiv 0 \pmod{N} \qquad \text{and} \qquad \sum_{j=1}^{t} m_j \equiv 0 \pmod{N}$$

then their product has st monomials $n_i + m_j$, $i = 1, \ldots, s$, $j = 1, \ldots, t$ (unless a collision occurs, which is very unlikely). Then

$$\begin{aligned} \sum_{i=1}^{s}\sum_{j=1}^{t}(n_i + m_j) &= \sum_{i=1}^{s}\left(tn_i + \sum_{j=1}^{t} m_j\right) \\ &= t\sum_{i=1}^{s} n_i + s\sum_{j=1}^{t} m_j \equiv 0 \pmod{N} \end{aligned}$$

as claimed. With this additional constraint, the discrete logarithm attack from [**1**] can only succeed with probability $1/N$, since in the collection of rt-sparse $\{0,1,A\}$-polynomials $e(X) \in \mathcal{R}[x]$ with $e(a_0) = 0$, the sum of the degrees of the monomials are uniformly distributed modulo N.

2.6. Remarks. Although using composite moduli might add some additional security features to the SPIFI scheme, the security is not compromised in the $\mathbb{Z}_M$-case even when the factorization $M = p\ell$ is known. In fact, in the case where the modulus is a (sufficiently large) prime, that is, in the $\mathbb{F}_q$-case, the scheme is quite secure.

It has turned out that *Alice* must make some commitment about the values of $D_1, \dots, D_{k-1}$ before she receives the polynomial h from *Bob*, otherwise there is a very simple attack on the scheme. On the other hand, sending the whole set to *Bob* before he selects his polynomial h has its own dangers as well. Sending the sum $D = D_1 + \ldots + D_{k-1}$ is just one of many possible ways for *Alice* to undertake some commitments about the values of $D_1, \dots, D_{k-1}$. Another way is to just send about half of the bits of $D_1, \dots, D_{k-1}$ in the Commitment step and then send the remaining bits in the Response step. This latter method seems to improve on the overall security against an on-line attack.

Finally, we mention that the SPIFI scheme can be modified so that the value $N = \varphi(M)$ or $\lambda(M)$ (see Section 2.1) remains a secret known only to *Alice*. Indeed, *Alice* can choose g in Step 1 of the verification protocol so that the reduction modulo $x^{N+1} - x$ that occurs in Step 3 produces a polynomial whose degree is not "too close" to N. In fact, "on average" it should be about $N(1 - 1/2rst)$ for the SPIFI scheme and about $N(1-1/2R)$ for the ENROOT scheme (see Section 3 below) since the corresponding polynomials are rst-sparse and R-sparse, respectively. Thus, in the case that $N = \varphi(M)$, the degree of F gives a worse approximation to N than the value of M itself, at least if $M = p\ell$ is a product of two primes of roughly the same order.

3. The ENROOT Cryptosystem

In this section, we describe the *ENROOT cryptosystem* for an arbitrary finite ring $\mathcal{R}$; see [**7**] and [**3**]. Since the polynomial ring $\mathcal{R}[x_1, \dots, x_d]$, where $d \ge 2$ is fixed in this construction, it is often convenient to employ vector notation, writing $f(\vec{x})$ for $f(x_1, \dots, x_d)$, $\mathcal{R}[\vec{x}]$ for $\mathcal{R}[x_1, \dots, x_d]$, etc.

3.1. A Hard Problem. Our one-way functions are based on the following hard problem:

> *Given the τ-sparse polynomials $f_1(\vec{x}), \dots, f_d(\vec{x}) \subset \mathcal{R}[\vec{x}]$ of degree at most N, it is not feasible to find an element $\vec{a} = (a_1, \dots, a_d) \in \mathcal{R}^d$ with $f_j(\vec{a}) = 0$ for $j = 1, \dots, d$, provided that N is sufficiently large relative to the choices of $d \ge 2$ and $\tau \ge 3$.*

Again, we expect that if one fixes the number $d \ge 2$ and the sparsity $\tau \ge 3$, then the problem requires exponential time as $N \to \infty$ (see Section 3.4 below).

3.2. The ENROOT Protocol. We fix the ring $\mathcal{R}$ and the integers $d > \ell \ge 3$, $s_j, t_j \ge 3$, $j = 1, \dots, d$ such that $t_1 = \ldots = t_\ell$. This information is made *public*. The value of N may be kept *private*. In fact, only *Bob* needs this value so the choice of the ring $\mathcal{R}$ and the value of N is made by *Bob*.

The ENROOT protocol consists of the following three steps:

- **Key Creation.** *Alice* selects d random elements $a_1, \ldots, a_d \in \mathcal{R}^\times$ which become her *private key*.

  Next, *Alice* randomly selects d polynomials $h_j(\vec{x}) \in \mathcal{R}[\vec{x}]$, of degree at most $\#\mathcal{R}$, containing at most $t_j - 1$ monomials, $j = 1, \ldots, d$, such that the first ℓ polynomials $h_1(\vec{x}), \ldots, h_\ell(\vec{x})$ have the same set $\mathcal{E}$ of exponents of their monomials. Finally, *Alice* publishes the polynomials $f_j(\vec{x}) = h_j(\vec{x}) - h_j(\vec{a})$ for $j = 1, \ldots, d$ as her *public key*, where $\vec{a}$ is the vector $(a_1, \ldots, a_d) \in (\mathcal{R}^\times)^d$.

- **Encryption.** To send a message $m \in \mathcal{R}$, *Bob* selects d random polynomials $g_j(\vec{x}) \in \mathcal{R}[\vec{x}]$ of degree at most N, containing at most $s_j - 1$ monomials such that one monomial has an exponent from the set $\mathcal{E}$ and having nonzero constant coefficients. *Bob* then computes the reduction $F(\vec{x})$ of the polynomial

  $$m + f_1(\vec{x})g_1(\vec{x}) + \ldots + f_d(\vec{x})g_d(\vec{x})$$

  modulo the ideal in $\mathcal{R}[\vec{x}]$ generated by $\{x_1^{N+1} - x_1, \ldots, x_d^{N+1} - x_d\}$, and he sends $F(\vec{x})$ to *Alice*.

- **Decryption.** To decrypt the message, *Alice* simply computes $F(\vec{a}) = m$.

3.3. Efficiency. The sparsity of the polynomials involved again provides computational efficiency for this scheme. Using repeated squaring, one can compute the monomial $a_1^{e_1} \ldots a_d^{e_d}$ for any $(a_1, \ldots, a_d) \in \mathcal{R}^d$ and $0 \le e_j \le \#\mathcal{R}$, $j = 1, \ldots, d$, in about $O(d \log(2\#\mathcal{R}))$ arithmetic operations in $\mathcal{R}$. Consequently, any τ-sparse polynomial $f(\vec{x}) \in \mathcal{R}[\vec{x}]$ of degree at most $\#\mathcal{R}$ can be evaluated at any point in $\mathcal{R}^d$ in about $O(\tau d \log(2\#\mathcal{R}))$ arithmetic operations in $\mathcal{R}$.

We remark that any τ-sparse polynomial $f(\vec{x}) \in \mathcal{R}[\vec{x}]$ of degree at most $\#\mathcal{R}$ can be encoded with about $\tau((d+1)\log(\#\mathcal{R}))$ bits. To do this, we have to identify at most τ monomials for which f has a nonzero coefficient. The encoding of each monomial $x_1^{e_1} \ldots x_d^{e_d}$ requires about $d \log(\#\mathcal{R})$ bits, and for each such monomial about $\log(\#\mathcal{R})$ bits are then required to encode the coefficient.

Let us set

$$T = \sum_{j=1}^{d} t_j, \qquad S = \sum_{j=1}^{d} s_j, \qquad R = \sum_{j=1}^{d} t_j s_j, \qquad Q = (d+1)\log(\#\mathcal{R}).$$

Then, after simple calculations, we derive that (using the naive repeated squaring exponentiation)

- *generation of the public key*: producing the vector $\vec{a}$ requires $O(d \log(\#\mathcal{R}))$ random bits; to construct the polynomials $h_j(\vec{x})$ requires the generation of another $(T-d)Q$ random bits; the computation of the $h_j(\vec{a})$, $j = 1, \ldots, d$, takes $O(Td \log(2\#\mathcal{R}))$ arithmetic operations in $\mathcal{R}$;
- the *private key size* is about $d \log(\#\mathcal{R}) + (T-d)Q$ bits;
- the *public key size* is about TQ bits;
- *cost of encryption*: to construct the polynomials $g_j(\vec{x})$ requires the generation of about $d \log(\#\mathcal{R}) + (S-d)Q$ random bits; the computation of the polynomial $F(\vec{x})$ requires about R arithmetic operations in $\mathcal{R}$ plus Rd additions in $\mathbb{Z}/N\mathbb{Z}$;

- the *size of the encrypted message* is about RQ bits;
- the *cost of decryption*: the evaluation of $F(\vec{a}) = m$ takes $O(Rd\log(2N))$ arithmetic operations in $\mathcal{R}$.

In the $\mathbb{F}_q$-case, the above scheme can be accelerated if *Alice* sets $e_1 = 1$, selects a random element $a \in \mathcal{R}^\times$ and $d-1$ random exponents $e_2, \dots, e_d \in \mathbb{Z}/(q-1)\mathbb{Z}$, and defines $\vec{a}$ as $(a^{e_1}, \dots, a^{e_d}) \in (\mathcal{R}^\times)^d$. Again we mention that the performance of the ENROOT algorithm can be improved if one uses more sophisticated algorithms to evaluate powers and sparse polynomials; see [**4, 5, 6, 16, 18**]. Another possible way to improve performance is to use at Step 4 only $k < d$ randomly selected polynomials from the set $\{f_1, \dots, f_d\}$. For the same level of security, there will be a trade-off between the complexity of Step 2 (hence the size of the private key) and the complexity of Step 4.

3.4. Security Considerations. The obvious way to attack the ENROOT cryptosystem is to try to find a simultaneous solution to the system of polynomial equations

$$f_j(\vec{x}) = 0, \qquad j = 1, \dots, d, \tag{3.1}$$

which amounts to solving the hard problem in Section 3.1. All known algorithms to solve systems of polynomial equations of total degree n require (regardless of sparsity) an amount of time that is polynomial in n (that is, exponential time with respect to the bit length of n); see [**8, 17**]. Since the degrees of the polynomials in (3.1) will be very large in practical implementations (about the size of N), this attack is totally infeasible.

Another possibility is to simply "guess" a solution. One expects that a system of τ-sparse polynomial equations in d variables of high degree over $\mathcal{R}$ will have very few zeros if $d \geq 2$, even though the number of zeros of a polynomial over an arbitrary ring is not necessarily bounded by the degree. Working heuristically, if we view the vector of polynomials $\vec{f}(\vec{x}) = (f_1(\vec{x}), \dots, f_d(\vec{x}))$ as defining a "random" map $\vec{f} : \mathcal{R}^d \to \mathcal{R}^d$, then the expected number of roots common to all of the polynomials $f_j(\vec{x})$ (that is, the cardinality of the kernel of $\vec{f}$) is given by

$$\frac{1 - \#\mathcal{R}^{d(1-\#\mathcal{R}^d)}}{1 - (1 - \#\mathcal{R}^{-d})^{\#\mathcal{R}^d}} \approx \frac{1}{1 - e^{-1}} \approx 1.5819,$$

hence this brute force attack will take roughly $0.245\#\mathcal{R}^d$ trials "on average" to succeed. For arbitrary rings, we expect the choice $d \geq 2$ will provide the 2^{90} level of security against this attack if N is at least 50 bits long.

Although it is tempting to choose $d = \ell = 1$, in this case there are more sophisticated attacks that provide some information about *Alice*'s private key. One of these is based upon consideration of the difference set of the powers of monomials occurring in the polynomial $F(x)$. Indeed, if

$$f(x) = \sum_{i=1}^{t} A_i x^{n_i} \qquad \text{and} \qquad g(x) = \sum_{j=1}^{s} B_i x^{m_j}$$

are the polynomials selected by *Alice* and *Bob*, respectively, with $n_1 = m_1 = 0$ (and such that the sets $\{n_1, \dots, n_t\}$ and $\{m_1, \dots, m_s\}$ have a reasonably large intersection) then $F(x)$ contains at most $\tau \leq st$ monomials $C_\nu x^{r_\nu}$, where

$$r_\nu \equiv n_i + m_j \pmod{N}, \qquad \nu = 1, \dots, \tau,$$

for some $i = 1, \ldots, t$ and $j = 1, \ldots, s$.

Consequently, finding the repeated elements in the difference set

$$\Delta = \{r_\nu - r_\eta \pmod N \,|\, \nu, \eta = 1, \ldots, \tau\},$$

may reveal some information about the polynomial $f(x)$.

In addition, if $d = 1$, one can also compute the greatest common divisor of $f(x)$ with $x^{N+1} - x$. Since this polynomial will have lower degree than f in general, it would be easier to find a root from a theoretical standpoint. Although it is not clear how to do this in an amount of time that is polynomial in the sparsity rather than in the degree of $f(x)$, it remains a potential threat.

On the other hand, these attacks on ENROOT fail when $d > \ell \geq 2$. Indeed, the first attack may help to gain some information about the total set of monomials in all of the polynomials $f_1, \ldots, f_d$, but it does not provide any information about the individual polynomials since it is impossible to determine which monomial comes from which product $f_j g_j$, $j = 1, \ldots, d$. In order to try all possible partitions into d groups of $s_j t_j$ monomials, $j = 1, \ldots, d$, needs to examine

$$\mathcal{P} = \frac{R\,!}{(s_1 t_1)\,! \ldots (s_d t_d)\,!} \tag{3.2}$$

total combinations. In particular, the most interesting case is when $s_1, \ldots, s_d$ are of approximately the same size as well as $t_1, \ldots, t_d$, that is, when $s_j \sim s$ and $t_j \sim t$ for $j = 1, \ldots, d$. In this case

$$\log \mathcal{P} \sim st(d \log d),$$

hence the number $\mathcal{P}$ of combinations to consider grows exponentially with respect to all parameters, provided that $d > \ell \geq 2$.

The second attack fails as well since the notion of greatest common divisor for multivariate polynomials is not defined, and taking resolvents to reduce to one variable is too costly.

However the case $d = 2$ is still not secure. Indeed, in this case we have either $d = \ell = 2$ or $\ell = 1$. In either case there are very simple linear algebra attacks which do not apply when $d > \ell \geq 2$ which we believe to be completely secure against the aforementioned attacks. There are some other alternative ways to guarantee that there are sufficiently many common elements in the sets of exponents of monomials of $f_1, \ldots, f_d$. In particular, the first few monomials of each polynomial $h_1, \ldots, h_d$ can be selected the same exponents.

Finally, we remark that the ENROOT cryptosystem is probably secure against lattice attacks for the same reason that the SPIFI scheme is secure (see Section 2.5): most lattice attacks would necessarily be based on lattices of dimension equal to the cardinality of $\#\mathcal{R}$, and in practical implementations this number would be very large.

References

[1] F. Bao, R. H. Deng, W. Geiselmann, C. Schnorr, R. Steinwandt and H. Wu, 'Crytoanalysis of two sparse polynomial based cryptosystems', *Proc. Int. Conf. on Public Key Cryptography, PKC'2001, Lect. Notes in Comp. Sci.*, Springer-Verlag, Berlin, 2001, to appear.

[2] W. Banks, D. Lieman and I. E. Shparlinski, 'An identification scheme based on sparse polynomials', *Lect. Notes in Comp. Sci.*, Springer-Verlag, Berlin, **1751**, 68–74.

[3] W. Banks, D. Lieman, I. Shparlinski, and V. To, "Cryptographic applications of sparse polynomials over finite rings," in *Proceedings of ICISC2000* (*Seoul*), Lecture Notes in Comput. Sci. **2015**, Springer-Verlag, Berlin (2001).

[4] H. Cohen *A course in computational algebraic number theory*, Springer-Verlag, Berlin, 1997.

[5] J. von zur Gathen and J. Gerhard, *Modern computer algebra*, Cambridge Univ. Press, Cambridge, 1999.

[6] D. M. Gordon, 'A survey of fast exponentiation methods', *J. Algorithms*, **27** (1998), 129–146.

[7] D. Grant, K. Krastev, D. Lieman and I. E. Shparlinski, 'A public key cryptosystem based on sparse polynomials', *Proc. International Conference on Coding Theory, Cryptography and Related Areas, Guanajuato, 1998*, Springer-Verlag, Berlin, 2000, 114–121.

[8] M.-D. A. Huang and Y.-C. Wong, 'Solving systems of polynomial congruences modulo a large prime', *Proc. 37 IEEE Symp. on Found. of Comp. Sci.*, 1996, 115–124.

[9] J. Hoffstein, B. S. Kaliski, D. Lieman, M. J. B. Robshaw and Y. L. Yin, 'A new identification scheme based on polynomial evaluation', *US Patent*, No. **No. 6076163**, 2000.

[10] J. Hoffstein, D. Lieman and J. H. Silverman, 'Polynomial Rings and Efficient Public Key Authentication', *Proc. the International Workshop on Cryptographic Techniques and E-Commerce*, City University of Hong Kong Press, to appear.

[11] J. Hoffstein, J. Pipher and J. H. Silverman, 'NTRU: A ring based public key cryptosystem', *Lect. Notes in Comp. Sci.*, Springer-Verlag, Berlin, **1433** (1998), 267–288.

[12] J. Hoffstein and J. H. Silverman, 'Polynomial rings and efficient public key authentication II', *Proc. the International Workshop on Cryptography and Computational Number Theory, Singapore, 1999*, Birkhäuser, to appear.

[13] J. Hoffstein and J. H. Silverman, 'Optimizations for NTRU', *Public Key Cryptography and Computational Number Theory, Warsaw, 2000*, available from `http://www.ntru.com`.

[14] N. Howgrave-Graham, J. H. Silverman, A. Singer, and W. Whyte, 'Choosing Parameter Sets for NTRUEncrypt with NAEP and SVES-3', *Proceedings of CT-RSA 2005*; expanded version available from `http://www.ntru.com`.

[15] A. Knopfmacher and J. Knopfmacher, 'Counting polynomials with a given number of zeros in a finite field', *Linear and Multilinear Algebra*, **26** (1990), 287–292.

[16] N. Pippenger, 'On the evaluation of powers and monomials', *SIAM J. Comp.*, **9** (1980), 230–250.

[17] I. E. Shparlinski, *Finite fields: Theory and computation*, Kluwer Acad. Publ., Dordrecht, 1999.

[18] A. C.-C. Yao, 'On the evaluation of powers', *SIAM J. Comp.*, **5** (1976), 100–103.

Department of Mathematics, University of Missouri, Columbia, Missouri 65211
E-mail address: `bbanks@math.missouri.edu`