Key Recovery Attack against an NTRU-type Somewhat Homomorphic Encryption Scheme

Massimo Chenal and Qiang Tang

APSIA group, SnT, University of Luxembourg
6, rue Richard Coudenhove-Kalergi, L-1359 Luxembourg
{massimo.chenal; qiang.tang}@uni.lu

Abstract. In this note, we present our key recovery attacks against NTRU-type somewhat homomorphic encryption schemes.

1 Key Recovery attack against the NTRU-based LTV12 SHE Scheme

In [LATV12], the authors present a multikey fully homomorphic encryption scheme, which allows to operate on inputs encrypted under multiple, unrelated keys. A ciphertext resulting from a multikey evaluation can be jointly decrypted using the secret keys of all the users involved in the computation. Their scheme is based on NTRU [HPS98], more specifically the variant of Stehle and Steinfeld [SS11]. Next, we are going to present key recovery attacks against the SHE Scheme in [LATV12].

1.1 The LATV12 SHE Scheme

Let λ be the security parameter, consider an integer $n = n(\lambda)$ and a prime number $q = q(\lambda)$. Consider also a degree-n polynomial $\phi(x) = \phi_\lambda(x)$: following [LATV12], we will use $\phi(x) = x^n + 1$. Finally, let $\chi = \chi(\lambda)$ a $B(\lambda)$-bounded error distribution over the ring $R := \mathbb{Z}[x]/(\phi(x))$.

The parameters $n, q, \phi(x)$ and χ are public and we assume that given λ, there are polynomial-time algorithms that output n, q and $\phi(x)$, and sample from the error distribution χ. The message space is $\mathcal{M} = \{0, 1\}$, and all operations on ciphertexts are carried out in the ring $R_q := \mathbb{Z}_q[x]/(\phi(x))$.

```
KeyGen(λ) :
    – sample $f', g \leftarrow \chi$
    – set $f := 2f' + 1$ so that $f \equiv 1 \bmod 2$
    – if $f$ is not invertible in $R_q$, resample $f'$
    – $\mathsf{pk} := h = 2gf^{-1} \in R_q$
    – $\mathsf{sk} := f \in R$
Encrypt(pk, m):
    – sample $s, e \leftarrow \chi$
    – output ciphertext $c := hs + 2e + m \in R_q$
Decrypt(sk, , c):
    – let $\mu = f \cdot c \in R_q$
    – output $\mu' := \mu \bmod 2$
```

For the purposes of our attack we don't need the evaluation step, so we omit it from the description of the scheme.

In [LATV12], the authors do not explicitly state how the decryption behaves if $\mu \bmod 2$ is not a constant. We consider three possibilities: (1) output directly $\mu \bmod 2$; (2) output the constant of $\mu \bmod R_2$; (3) output an error.

1.2 Our Key Recovery Attack in Case (1)

Suppose the secret key is in the form of the polynomial

$$f = s(x) = s_0 + s_1 x + s_2 x^2 + \cdots + s_{n-1} x^{n-1} \in R_q$$

with $s_i \in [0, q-1]$ for all $i = 0, 1, \ldots, n-1$.

We remark that

$$\begin{aligned} \#\text{bits}(s_i) &= \lfloor \log_2(q-1) \rfloor + 1 =: N \\ \#\text{bits}(s) &= n \cdot \#\text{bits}(s_i) = n \cdot (\lfloor \log_2(q-1) \rfloor + 1) \end{aligned}$$

The decryption oracle reveals a polynomial

$$\mu'(x) = \mu(x) \bmod 2 = \mu'_0 + \mu'_1 x + \cdots + \mu'_{n-1} x^{n-1} \in R_2$$

with $\mu'_i \in \{0, 1\}$ for $i = 0, 1, \ldots, n-1$. Hence, decryption oracle reveals n bits at a time. Therefore, the minimum number of oracle queries needed to recover s is N. As we will see, our attack needs at most N oracle queries, so in this sense it is optimal.

Here is the idea of our key recovery attack. First of all, we are going to determine the parity of each coefficient $s_i \in [0, q-1]$. Then, we are going to find s_i by gradually reducing (halving) the interval in which it lies. In the last step, s_i will be reduced to belong to some interval with at most two consecutive integers; the exact value of s_i will be deduced by its (known) parity.

Preliminary Step.
Submit to the decryption oracle the 'ciphertext' $c = 1 \in R_q$. The oracle will compute

$$\begin{aligned}\mu \bmod 2 = s \bmod 2 &= (s_0 \bmod 2) + (s_1 \bmod 2)x + \cdots + (s_{n-1} \bmod 2)x^{n-1} \\ &= \sum_{i=0}^{n-1} (s_i \bmod 2) x^i\end{aligned}$$

which tells us the parity of each s_i, $i = 0, 1, \ldots, n-1$.

Step 1.
Put $c = 2 \in R_q$ and submit it to the decryption oracle; it will compute and return the polynomial

$$\begin{aligned}\mu \bmod 2 &= (2s \in R_q) \bmod 2 \\ &= [(2s_0 \bmod q) \bmod 2] + [(2s_1 \bmod q) \bmod 2]\, x + \cdots + [(2s_{n-1} \bmod q) \bmod 2]\, x^{n-1} \\ &= \sum_{i=0}^{n-1} [(2s_i \bmod q) \bmod 2]\, x^i\end{aligned}$$

For all $i \in [0, n-1]$ we have $0 \le s_i \le q-1$, and so

$$0 \le 2s_i \le 2q-2 \tag{A}$$

For each i, we have two cases to distinguish:

Case A_1: $(2s_i \bmod q) \bmod 2 = 0$. Then, condition (A) implies that

$$0 \le 2s_i \le q-1, \quad \text{or} \quad 0 \le s_i \le \frac{q-1}{2}$$

We also have

$$0 \le 4s_i \le 2q-2 \tag{A1}$$

Case B_1: $(2s_i \bmod q) \bmod 2 = 1$. Then, condition (A) implies that

$$q+1 \le 2s_i \le 2q-2, \quad \text{or} \quad \frac{q+1}{2} \le s_i \le q-1$$

We also have

$$2q+2 \le 4s_i \le 4q-4 \tag{B1}$$

Step 2.
Put $c = 4 \in R_q$ and submit it to the decryption oracle; it will compute and return the polynomial

$$\begin{aligned}\mu \bmod 2 &= (4s \in R_q) \bmod 2 \\ &= [(4s_0 \bmod q) \bmod 2] + [(4s_1 \bmod q) \bmod 2]\, x + \cdots + [(4s_{n-1} \bmod q) \bmod 2]\, x^{n-1} \\ &= \sum_{i=0}^{n-1} [(4s_i \bmod q) \bmod 2]\, x^i\end{aligned}$$

For each i, we have four cases to distinguish:

Case A_2: In Step 1 case A_1 held, and $(4s_i \bmod q) \bmod 2 = 0$. Then, condition (A1) implies that

$$0 \le 4s_i \le q-1, \quad \text{or} \quad 0 \le s_i \le \frac{q-1}{4}$$

We also have

$$0 \le 8s_i \le 2q-2 \tag{A2}$$

Case B_2: In Step 1 case A_1 held, and $(4s_i \bmod q) \bmod 2 = 1$. Then, condition (A1) implies that

$$q+1 \le 4s_i \le 2q-2, \quad \text{or} \quad \frac{q+1}{4} \le s_i \le \frac{q-1}{2}$$

We also have

$$2q+2 \le 8s_i \le 4q-4 \tag{B2}$$

Case C_2: In Step 1 case B_1 held, and $(4s_i \bmod q) \bmod 2 = 0$. Then, condition (B1) implies that

$$2q+2 \le 4s_i \le 3q-1, \quad \text{or} \quad \frac{q+1}{2} \le s_i \le \frac{3q-1}{4}$$

We also have

$$4q+4 \le 8s_i \le 6q-2 \tag{C2}$$

Case D_2: In Step 1 case B_1 held, and $(4s_i \bmod q) \bmod 2 = 1$. Then, condition (B1) implies that

$$3q+1 \le 4s_i \le 4q-4, \quad \text{or} \quad \frac{3q+1}{4} \le s_i \le q-1$$

We also have

$$6q+2 \le 8s_i \le 8q-8 \tag{D2}$$

Generalizing.
Put $I_{0,1} := [0, q-1]$. We can see that, after Step 1, we have reduced each s_i to one of the two intervals

$$I_{1,1} := \left[0, \frac{q-1}{2}\right], \quad I_{1,2} := \left[\frac{q+1}{2}, q-1\right]$$

After Step 2, we have reduced each s_i to one of the four intervals:

$$\begin{aligned} I_{2,1} &:= \left[0, \frac{q-1}{4}\right], & I_{2,2} &:= \left[\frac{q+1}{4}, \frac{q-1}{2}\right] \\ I_{2,3} &:= \left[\frac{q+1}{2}, \frac{3q-1}{4}\right], & I_{2,4} &:= \left[\frac{3q+1}{4}, q-1\right]\end{aligned}$$

Reasoning in similar ways, in Step 3 we put $c = 8$ and, after the decryption oracle query, we are able to reduce each s_i to one of the eight intervals

$$\begin{aligned} I_{3,1} &:= \left[0, \frac{q-1}{8}\right], & I_{3,2} &:= \left[\frac{q+1}{8}, \frac{q-1}{4}\right] \\ I_{3,3} &:= \left[\frac{q+1}{4}, \frac{3q-1}{8}\right], & I_{3,4} &:= \left[\frac{3q+1}{8}, \frac{q-1}{2}\right] \\ I_{3,5} &:= \left[\frac{q+1}{2}, \frac{5q-1}{8}\right], & I_{3,6} &:= \left[\frac{5q+1}{8}, \frac{3q-1}{4}\right] \\ I_{3,7} &:= \left[\frac{3q+1}{4}, \frac{7q-1}{8}\right], & I_{3,8} &:= \left[\frac{7q+1}{8}, q-1\right] \end{aligned}$$

It is easy to see that, after each step k, a smallest interval is given by

$$I_{k,2^k} = \left[\frac{\left(2^k-1\right)\cdot q+1}{2^k}, q-1\right]$$

The numbers of elements in $I_{k,2^k}$ is given by

$$\begin{aligned} |I_{k,2^k}| &= q-1-\left\lceil\frac{\left(2^k-1\right)\cdot q+1}{2^k}\right\rceil+1 \\ &= \left\lfloor\frac{q-\left(2^k+1\right)}{2^k}\right\rfloor+1 \end{aligned}$$

A largest interval is given by

$$I_{k,1} = \left[0, \frac{q-1}{2^k}\right]$$

with

$$|I_{k,1}| = \left\lfloor\frac{q-1}{2^k}\right\rfloor+1$$

Now, we keep on reasoning in the same way until we reach Step m, where m is such that

$$2^m \leq q-1 \leq 2^{m+1}, \quad \text{i.e.} \quad m = \lfloor\log_2(q-1)\rfloor$$

In fact, at Step m we have

$$\begin{aligned} |I_{m,1}| &= \left\lfloor\frac{q-1}{2^m}\right\rfloor+1 = 2, \quad \text{and} \\ |I_{m,2^m}| &= \left\lfloor\frac{q-(2^m+1)}{2^m}\right\rfloor+1 = 1 \end{aligned}$$

where last equality holds since

$$\frac{q-(2^m+1)}{2^m} = \frac{q-1}{2^m}-1, \quad \text{and} \quad 1 \leq \frac{q-1}{2^m} < 2$$

so we have

$$0 \leq \frac{q-(2^m+1)}{2^m} < 1, \quad \text{i.e.} \quad \left\lfloor\frac{q-(2^m+1)}{2^m}\right\rfloor = 0$$

So, at Step m, the interval $I_{0,1} = [0, q-1]$ is divided into 2^m intervals $\{I_{m,t}\}_{t=1}^{2^m}$. Moreover,

$$1 = |I_{m,2^m}| \leq |I_{m,t}| \leq |I_{m,1}| = 2, \quad \forall t = 1,2,3,\ldots,2^m$$

Final step.
For any given $t \in [1, 2, 3, \ldots, 2^m]$ we have to distinguish two cases:

- if $|I_{m,t}| = 1$, then $s_i \in I_{m,t}$ is determined;
- if $|I_{m,t}| = 2$, then $I_{m,t} = \{r, r+1\}$ for some $r \in [0, q-2]$. Therefore, $s_i \in I_{m,t}$ is determined by its parity (which was computed in the preliminary step).

How many oracle queries we need in order to recover s? We have to perform m steps each one with one call to the oracle query, plus one additional call to the oracle query in the preliminary step. Hence, we need

$$m + 1 = \lfloor \log_2(q-1) \rfloor + 1 = N$$

oracle queries to recover s, and for what we have said previously, our attack is indeed optimal.

General form of the intervals $I_{k,t}$, for $k \geq 1$ and $1 \leq t \leq 2^k$.
For a given prime q, we give the explicit general form of the set of intervals $\{I_{k,1}, \ldots, I_{k,2^k}\}_{k=0}^{m}$. For any integer $a \in I_{0,1} = [0, q-1] = \mathbb{Z}_q$, consider the function

$$\begin{aligned} f : \mathbb{Z}_q &\to \mathbb{N} \\ a &\mapsto \max\big\{2^r \text{ s.t. } 2^r \mid a, \text{ with } r \in \mathbb{N} = \{0, 1, 2, \ldots\}\big\} \end{aligned}$$

We put $f(0) := 0$.

So, for a given $a \in \mathbb{Z}_q \backslash \{0\}$, $f(a)$ is the maximum power of 2 dividing a. Then we can check that

$$I_{k,t} = \left[\frac{f(t-1) \cdot \left(\frac{t-1}{f(t-1)} \cdot q + 1 \right)}{2^k}, \frac{f(t) \cdot \left(\frac{t}{f(t)} \cdot q - 1 \right)}{2^k} \right]$$

Special care has to be taken when $t = 1$; in fact

$$f(t-1) \cdot \left(\frac{t-1}{f(t-1)} \cdot q + 1 \right) \bigg|_{t=1} = 0 \cdot \left(\frac{0}{0} \cdot q + 1 \right)$$

To avoid confusion given by division by 0, we write $I_{k,t}$ as before for $2 \leq t \leq 2^k$, and $I_{k,1} = \big[0, \frac{q-1}{2^k}\big]$.

Our Key Recovery Attack, formally We can now generalize our key recovery attack, and more formally write the generic k-th step, for $1 \leq k \leq m$. Let $I_{0,1} := [0, q-1]$. The preliminary step (or Step 0) is as before. For any $1 \leq k \leq m$, Step k can be written as follows.
Step k: Put $c = 2^k \in R_q$ and submit it to the decryption oracle; it will compute and return the polynomial

$$\begin{aligned} \mu \bmod 2 &= (2^k \cdot s \in R_q) \bmod 2 \\ &= \big[(2^k \cdot s_0 \bmod q) \bmod 2\big] + \big[(2^k \cdot s_1 \bmod q) \bmod 2\big]\, x + \cdots + \big[(2^k \cdot s_{n-1} \bmod q) \bmod 2\big]\, x^{n-1} \\ &= \sum_{i=0}^{n-1} \big[(2^k \cdot s_i \bmod q) \bmod 2\big]\, x^i \end{aligned}$$

For each i, we know from Step $(k-1)$ the value $t \in [1, 2^{k-1}]$ such that $s_i \in I_{k-1,t}$. Now, we have two cases to distinguish:

- if $(2^k \cdot s_i \bmod q) \bmod 2 = 0$, then $s_i \in I_{k,2t-1}$;
- if $(2^k \cdot s_i \bmod q) \bmod 2 = 1$, then $s_i \in I_{k,2t}$

Repeat Step k for $k = 1, 2, \ldots, m$.

Final step is as before.

1.3 Our Key Recovery Attack in Case (2)

We show that, even in this Case (2), our attack works with few modifications. Instead of recovering all coefficients s_i of the polynomial $s(x) = s_0 + s_1 x + \cdots + s_{n-1}x^{n-1} \in R_q$ at once, we are going to recover in sequence $s_0, s_1, \ldots, s_{n-1}$.

Recovering s_0 .
It is clear that, by performing the same attack as described above, we recover coefficient s_0 with at most N oracle queries; but no information will be leaked about s_i, for $1 \leq i \leq n-1$.

Recovering s_1 .
In order to recover s_1, we repeat the same attack as before, with the following modifications:

Preliminary step: submit to the decryption oracle the 'ciphertext' $c = -x^{n-1} \in R_q$. This way,

$$\begin{aligned}\mu(x) \bmod 2 &= s \cdot (-x^{n-1}) \bmod 2 = \\ &= (s_1 \bmod 2) + (s_2 \bmod 2)x + \cdots + (s_{n-1} \bmod 2)x^{n-2} - (s_0 \bmod 2)x^{n-1}\end{aligned}$$

since $x^n = -1$ in R_q. So $\mu'_0 = s_1 \bmod 2$.
Similarly,
Step k, for $1 \leq k \leq m$, with $m = \lfloor \log_2(q-1) \rfloor$: Submit to the decryption oracle the 'ciphertext'

$$c = 2^k \cdot (-x^{n-1}) \in R_q$$

These modifications lead to a full recovery of s_1 (the final step is the same as in the original key-recovery attack).

Recovering s_i, for $0 \leq i \leq n-1$.
Similarly, and more generally, we are going to recover $s_i \in [0, q-1]$, for all $0 \leq i \leq n-1$. Steps are as follows:

Preliminary step: submit to the decryption oracle the 'ciphertext' $c = -x^{n-i} \in R_q$.
Step k, for $1 \leq k \leq m$, with $m = \lfloor \log_2(q-1) \rfloor$: Submit to the decryption oracle the 'ciphertext'

$$c = 2^k \cdot (-x^{n-i}) \in R_q$$

Final step: same as in the original key-recovery attack.

We can consider an efficiency analysis of our key-recovery attack for the modified [LATV12] SHE scheme. In this case, decryption oracle reveals only one bit at a time. Since secret polynomial $s(x)$ has $n \cdot N$ bits, we need at least $n \cdot N$ oracle queries in order to recover $s(x)$. In our previous attack, we recover each s_i, for $0 \leq i \leq n-1$, with N oracle queries; repeating the attack for every i, gives us a total of $n \cdot N$ oracle queries. Hence, our attack is optimal.

2 Summary

So far, we have only successfully mount key recovery attacks for Case (1) and (2). It is likely that we can adapt our attack to Case (3), but we have not succeeded so far. This is still an ongoing work. It is worth noting that our attack can be applied to the NTRU variant of Stehle and Steinfeld [SS11] and others straightforwardly.

References

[HPS98] Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. NTRU: A ring-based public key cryptosystem. In JoeP. Buhler, editor, *Algorithmic Number Theory*, volume 1423 of *Lecture Notes in Computer Science*, pages 267–288. Springer Berlin Heidelberg, 1998.

[LATV12] Adriana López-Alt, Eran Tromer, and Vinod Vaikuntanathan. On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In *Proceedings of the Forty-fourth Annual ACM Symposium on Theory of Computing*, STOC '12, pages 1219–1234, New York, NY, USA, 2012. ACM.

[SS11] Damien Stehlé and Ron Steinfeld. Making ntruencrypt and ntrusign as secure as standard worst-case problems over ideal lattices. In K. G. Paterson, editor, *Advances in Cryptology — EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 27–47. Springer, 2011.