– An extended abstract of this paper appears in the proceedings of LATINCRYPT 2014 –

On Key Recovery Attacks against Existing Somewhat Homomorphic Encryption Schemes

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 his seminal paper at STOC 2009, Gentry left it as a future work to investigate (somewhat) homomorphic encryption schemes with IND-CCA1 security. At SAC 2011, Loftus et al. showed an IND-CCA1 attack against the somewhat homomorphic encryption scheme presented by Gentry and Halevi at Eurocrypt 2011. At ISPEC 2012, Zhang, Plantard and Susilo showed an IND-CCA1 attack against the somewhat homomorphic encryption scheme developed by van Dijk et al. at Eurocrypt 2010.

In this paper, we continue this line of research and show that most existing somewhat homomorphic encryption schemes are not IND-CCA1 secure. In fact, we show that these schemes suffer from key recovery attacks (stronger than a typical IND-CCA1 attack), which allow an adversary to recover the private keys through a number of decryption oracle queries. The schemes, that we study in detail, include those by Brakerski and Vaikuntanathan at Crypto 2011 and FOCS 2011, and that by Gentry, Sahai and Waters at Crypto 2013. We also develop a key recovery attack that applies to the somewhat homomorphic encryption scheme by van Dijk et al., and our attack is more efficient and conceptually simpler than the one developed by Zhang et al.. Our key recovery attacks also apply to the scheme by Brakerski, Gentry and Vaikuntanathan at ITCS 2012, and we also describe a key recovery attack for the scheme developed by Brakerski at Crypto 2012.

Keywords: Somewhat Homomorphic Encryption, Key Recovery Attack, IND-CCA1 Security.

1 Introduction

In 1978, Rivest, Adleman and Dertouzos [RAD78] introduced the concept of privacy homomorphism and asked whether it is possible to perform arbitrary operations on encrypted ciphertexts. 30 years later, Gentry [Gen09b] gave a positive answer by proposing an ingenious approach to construct fully homomorphic encryption (FHE) schemes. With this approach, we can start with a somewhat homomorphic encryption (SHE) scheme that can perform only limited number of operations on ciphertexts (i.e. it can evaluate only low-degree polynomials). Then, through the so-called bootstrapping step, we can turn this SHE scheme into an FHE scheme. Even though SHE schemes are less powerful than FHE schemes, they can already be used in many useful real-world applications, such as medical and financial applications [NLV11]. Note that researchers have proposed the concept of leveled FHE schemes (e.g. [BGV12,GSW13]), which allow third parties to evaluate any circuits up to a certain depth. In the following discussion, we treat these schemes as SHE.

1.1 Related Work

After Gentry's work, many SHE and FHE schemes have been proposed. Based on the underlying hardness assumptions, these schemes can be categorized as follows.

(1) The first category starts with Gentry [Gen09a,Gen09b]. A number of variations, optimizations and implementations appear in [SV10,GH11b]. The security of these schemes are based on hard problems on lattices.
(2) The second category starts with van Dijk et al. [vDGHV10]. More variants, implementation and optimizations appear in [CMNT11,CNT12,CCK$^+$13]. The security of these schemes rely on the approximate greatest common divisor (AGCD) problem and some variants. It is worth mentioning that Ding and Tao [DT14] claim to have found an algorithm to solve the AGCD problem with some special parameters in polynomial time. However, the AGCD problem and its variants are still believed to be hard.
(3) The third category starts with Brakerski and Vaikuntanathan [BV11b,BV11a]. More variants appear in [NLV11,BGV12,GHS12b,Bra12,GSW13]. The security of these schemes are based on the learning with errors (LWE) and on the ring-learning with errors (RLWE) problems.

See Fig. 1 for a graphical visualization of the main families.

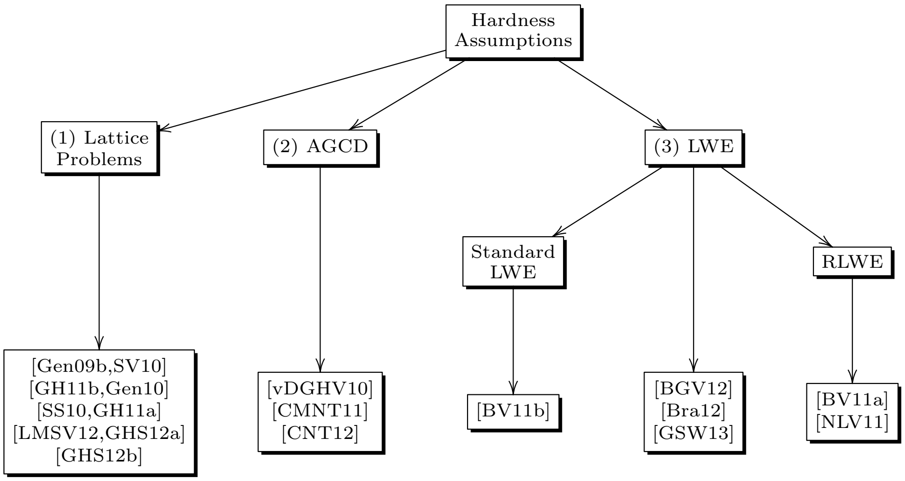

Fig. 1: Hardness assumptions and relevant papers

Recently, Nuida [Nui14] proposed a new framework for noise-free FHE, based on finite non-commutative groups. This is completely different from everything appeared in literature so far, since the ciphertext in all known schemes carry some noise. Nevertheless, a secure instantiation has yet to be found.

There exists another family of schemes, based on the NTRU encryption scheme [HPS98]. In [LATV12] it is shown how to obtain a homomorphic encryption scheme in a multi-user setting, introducing the notion of multi-key homomorphic encryption where it is possible to compute any function on plaintexts encrypted under multiple public keys. The multi-key FHE of [LATV12] is based on the NTRU scheme [HPS98] and on ideas introduced in [BGV12]. We will not focus on NTRU-based multi-key homomorphic encryption schemes.

All known SHE and FHE schemes have been developed with the aim of being IND-CPA secure (resistant against a chosen-plaintext attack). In [Gen09b], Gentry left it as a future work to investigate SHE schemes with IND-CCA1 security (i.e. secure against a non-adaptive chosen-ciphertext attack). At this moment, we have the following results.

- No SHE and FHE scheme can be IND-CCA2 (secure against adaptive chosen-ciphertext attack). The reason is straightforward, based on the fact that the adversary is allowed to manipulate the challenged ciphertext and submit it to the decryption oracle in an IND-CCA2 attack.
- With Gentry's approach, the resulted FHE scheme cannot be IND-CCA1 secure. The reason is also straightforward, based on the fact that the private key is encrypted and the adversary is able to submit the ciphertext to the decryption oracle.
- Loftus et al. [LMSV12] showed that Gentry's SHE scheme [Gen09b] is not IND-CCA1 secure and presented an IND-CCA1 attack against the variation proposed in [GH11b]. They also showed that the same attack applies to the other variant by Smart and Vercauteren [SV10]. In fact, the attacks are both key recovery attacks. Moreover, they modified the SHE in [SV10] and proved its IND-CCA1 security based on a new assumption. Zhang et al. [ZPS12] presented an IND-CCA1 attack against the SHE scheme in [vDGHV10], which can recover the secret key with $O(\lambda^2)$ queries where λ is the security parameter.

In theory, IND-CPA security may be enough for us to construct cryptographic protocols, in particular if we assume semi-honest attackers. However, key recovery attacks will pose serious threat for practical usage of SHE and FHE. If a malicious attacker submits manipulated ciphertexts and observes the behavior (side channel leakage) of the decryptor, then it may be able to recover all plaintexts in the system. Therefore, it is very desirable to design SHE and FHE with IND-CCA1 security, or at least to avoid key recovery attacks.

1.2 Our Contributions

In this paper, we continue the line of work of [LMSV12,ZPS12] to present key recovery attacks for the schemes [BV11b,BV11a,GSW13,Bra12]. Our attacks can also be applied to the SHE scheme in [BGV12]. We also develop a new key recovery attack against the SHE scheme in [vDGHV10], and our attack is more efficient and conceptually simpler than that from [ZPS12]. Our results essentially show that the SHE schemes underlying the FHE schemes in category (3) above are not IND-CCA1 secure. Combining the results from [LMSV12,ZPS12], we can conclude that most existing SHE schemes, except that from [LMSV12], suffer from key recovery attacks so that they are not IND-CCA1 secure.

1.3 Structure of the Paper

In Section 2, we recall some background on SHE and FHE schemes. Starting from Section 4, we are going to develop key recovery attacks against the aforementioned SHE schemes. In Section 8, we conclude the paper.

2 Preliminaries

Let $\mathbb{N}$ be the set of natural numbers, $\mathbb{Z}$ the ring of integers, $\mathbb{Q}$ the field of rational numbers, and $\mathbb{F}_q$ a finite field with q elements, where q is a power of a prime p. In particular, we will consider often $\mathbb{F}_p = \mathbb{Z}/p\mathbb{Z} = \mathbb{Z}_p$. If $r \in \mathbb{Z}_q$, we indicate as r^{-1} its inverse in $\mathbb{Z}_q$, i.e. that value such that $r^{-1} \cdot r = 1 \bmod \; q$. For a ring R and a (two-sided) ideal I of R, we consider the quotient ring R/I. For given vectors $\mathbf{v} = (v_1, \dots, v_n), \mathbf{w} = (w_1, \dots, w_n) \in \mathbb{Z}_q^n$, we let $< \mathbf{v}, \mathbf{w} >= \sum_i v_i w_i$ the dot product of $\mathbf{v}, \mathbf{w}$. For a given rational number $x \in \mathbb{Q}$, we let $\lfloor x \rceil$, $\lfloor x \rfloor$ and $\lceil x \rceil$ be respectively the rounding function, the floor function and the ceiling function. For a given integer $n \in \mathbb{N}$, $\lfloor n + 1/2 \rceil = n + 1$. To indicate that an element a is chosen uniformly at random from a set A we use notation $a \overset{\$}{\leftarrow} A$. For a set A, we let its cardinality be $|A|$. We consider also the standard basis $\{\mathbf{e}_i\}_{i=1}^n$ of $\mathbb{R}^n$, where the coefficients of $\mathbf{e}_i$ are all 0 except for the i-th coefficient, which is 1.

Unless otherwise specified, λ will always denote the security parameter of the encryption schemes. In the asymmetric schemes we are going to discuss, the secret key will be denoted as sk, and the public key will be pk.

2.1 Homomorphic Encryption

The following definitions are adapted from [Gen09b]. We only assume bit-by-bit public-key encryption, i.e. we only consider encryption schemes that are homomorphic with respect to boolean circuits consisting of gates for addition and multiplication mod 2. Extensions to bigger plaintext spaces and symmetric-key setting are straightforward; we skip the details.

Definition 1 (Homomorphic Encryption). *A public key homomorphic encryption (HE) scheme is a set of four algorithms $\mathcal{E} = (\mathsf{KeyGen}_{\mathcal{E}}, \mathsf{Encrypt}_{\mathcal{E}}, \mathsf{Decrypt}_{\mathcal{E}}, \mathsf{Evaluate}_{\mathcal{E}})$ all of which must run in polynomial time. When the context is clear, we will often omit the index $\mathcal{E}$.*

$\mathsf{KeyGen}(\lambda) = (\mathsf{sk}, \mathsf{pk})$
- *input:* λ
- *output:* sk; pk

$\mathsf{Encrypt}(\mathsf{pk}, m) = c$
- *input:* pk *and plaintext* $m \in \mathbb{F}_2$
- *output: ciphertext* c

$\mathsf{Decrypt}(\mathsf{sk}, c) = m'$
- *input:* sk *and ciphertext* c
- *output:* $m' \in \mathbb{F}_2$

$\mathsf{Evaluate}(\mathsf{pk}, C, (c_1, \dots, c_r)) = c_e$
- *input:* pk, *a circuit* C, *ciphertexts* $c_1, \dots, c_r$, *with* $c_i = \mathsf{Encrypt}(\mathsf{pk}, m_i)$
- *output: ciphertext* c_e

Definition 2 (Correct Homomorphic Decryption). *The public key homomorphic encryption scheme $\mathcal{E} = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt}, \mathsf{Evaluate})$ is correct for a given t-input circuit C if, for any key-pair $(\mathsf{sk}, \mathsf{pk})$ output by $\mathsf{KeyGen}(\lambda)$, any t plaintext bits $m_1, \dots, m_t$, and any ciphertexts $\overline{c} = (c_1, \dots, c_t)$ with $c_i \leftarrow \mathsf{Encrypt}_{\mathcal{E}}(\mathsf{pk}, m_i)$, $\mathsf{Decrypt}(\mathsf{sk}, \mathsf{Evaluate}(\mathsf{pk}, C, \overline{c})) = C(m_1, \dots, m_t)$ holds.*

Definition 3 (Homomorphic Encryption). $\mathcal{E} = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt}, \mathsf{Evaluate})$ *is homomorphic for a class $\mathcal{C}$ of circuits if it is correct for all circuits $C \in \mathcal{C}$. We say that $\mathcal{E}$ is a fully homomorphic encryption (FHE) scheme if it is correct for all boolean circuits.*

Informally, a homomorphic encryption scheme that can perform only a limited number of operations is called a somewhat homomorphic encryption (SHE) scheme.

2.2 Security Definitions

The security of a public-key encryption scheme in terms of indistinguishability is normally presented as a game between a challenger and an adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$. The scheme is considered secure if no adversary can win the game with significantly greater probability than an adversary who must guess randomly. The game runs in two stages:

- $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$
- $(m_0, m_1) \leftarrow \mathcal{A}_1^{(\cdot)}(\mathsf{pk})$ /* Stage 1 */
- $b \leftarrow \{0, 1\}$
- $c^* \leftarrow \mathsf{Encrypt}(m_b, \mathsf{pk})$
- $b' \leftarrow \mathcal{A}_2^{(\cdot)}(c^*)$ /* Stage 2 */

The adversary is said to win the game if $b = b'$, with the advantage of the adversary winning the game being defined by

$$\mathrm{Adv}_{\mathcal{A},\mathcal{E},\lambda}^{\text{IND-atk}} = |\Pr(b = b') - 1/2|$$

A scheme is said to be IND-atk secure if no polynomial time adversary $\mathcal{A}$ can win the above game with non-negligible advantage in the security parameter λ. The precise security notion one obtains depends on the oracle access one gives the adversary in its different stages:

- If $\mathcal{A}$ has access to no oracles in either stage then atk=CPA (indistinguishability under chosen plaintext attack)
- If $\mathcal{A}$ has access to a decryption oracle in stage one then atk=CCA1 (indistinguishability under non-adaptive chosen ciphertext attack)
- If $\mathcal{A}$ has access to a decryption oracle in both stages then atk=CCA2, often now denoted simply CCA (indistinguishability under adaptive chosen ciphertext attack)
- If $\mathcal{A}$ has access to a ciphertext validity oracle in both stages, which on input of a ciphertext determines whether it would output $\perp$ or not on decryption, then atk=CVA.

According to the definition, in order to show that a scheme is not IND-CCA1 secure, we only need to show that an adversary can guess the bit b with a non-negligible advantage given access to the decryption oracle in Stage 1. Formally, in a *key recovery attack*, an adversary can output the private key given access to the decryption oracle in Stage 1. In comparison, a key recovery attack is stronger than a typical IND-CCA1 attack.

3 Key Recovery Attack against the vDGHV10 Scheme

In [ZPS12], Zhang, Plantard and Susilo presented a key recovery attack against the the vDGHV10 scheme from [vDGHV10]. Given $O(\lambda^2)$ decryption oracle queries, an attacker can recover the private key. Let η be the bit-length of the secret key p, $O(\lambda^2) = 3(\eta + 3)$ in the best case.

In this section, we describe a more efficient and conceptually simpler key recovery attack. Our attack is optimal in the sense that it recovers directly the secret key with at most η oracle queries. Note that the decryption oracle outputs one bit at a time.

3.1 The vDGHV SHE Scheme

We start by presenting the (asymmetric) SHE scheme as presented in [vDGHV10]. The message space is $\mathcal{M} = \mathbb{Z}_2$. The scheme is parametrized by γ (bit-length of the integers in the public key), η (bit-length of the secret key), ρ (bit-length of the noise), and τ (the number of integers in the public key). We also consider a secondary noise parameter $\rho' = \rho + \omega(\log\lambda)$. For a specific ($\eta$-bit) odd positive integer p, consider the following distribution over γ-bit integers:

$$\mathcal{D}_{\gamma,\rho}(p) = \{\text{choose } q \xleftarrow{\$} \mathbb{Z} \cap [0, 2^\gamma/p), r \xleftarrow{\$} \mathbb{Z} \cap (-2^\rho, 2^\rho) : \text{output } x = pq + r\}$$

The algorithms of the vDGHV SHE scheme are defined as follows:

$\mathsf{KeyGen}(\lambda)$
- sk: odd η-bit integer $p \xleftarrow{\$} (2\mathbb{Z} + 1) \cap [2^{\eta-1}, 2^\eta)$.
- sample $x_i \xleftarrow{\$} \mathcal{D}_{\gamma,\rho}(p)$ for $i = 0, \ldots, \tau$
- relabel so that x_0 is the largest
- restart unless x_0 odd, $r_p(x_0)$ even ($r_p(x) = x - \lfloor x/p \rceil \cdot p \in (-p/2, p/2]$)
- $\mathsf{pk} = (x_0, x_1, \ldots, x_\tau)$.

$\mathsf{Encrypt}(\mathsf{pk}, m \in \mathcal{M})$
- choose a random subset $S \subseteq \{1, 2, \ldots, \tau\}$
- choose a random integer r in $(-2^{\rho'}, 2^{\rho'})$
- output $c = [m + 2r + 2\sum_{i \in S} x_i]_{x_0}$

$\mathsf{Decrypt}(\mathsf{sk}, c)$
- output $m' = (c \bmod p) \bmod 2$

3.2 The New Key Recovery Attack

Since $\eta = \#\text{bits}(p)$, we immediately obtain odd lower and upper bounds l_p and u_p, respectively, for p:

$$l_p = 2^{\eta-1} + 1 \leq p \leq u_p = 2^\eta - 1$$

Notice explicitly that p can only assume the odd values $2^{\eta-1} + 1, 2^{\eta-1} + 3, \ldots, 2^\eta - 3, 2^\eta - 1$. In particular, between $2^{\eta-1}$ and 2^η there are $2^{\eta-2}$ candidate values for p. We can also argue that between l_p and u_p there are $(u_p - l_p)/2 = 2^{\eta-2} - 1$ even integers. Let $H_{(l_p,u_p)} = \{0, 1, \ldots, 2^{\eta-2} - 2\}$, these integers can be denoted as $l_p + 2h + 1$ for $h \in H_{(l_p,u_p)}$.

Now, the idea of the key-recovery attack is as follows: consider the 'ciphertext' $c = l_p + 2h + 1$ for a given $h \in H_{(l_p,u_p)}$. Submit c to the decryption oracle $\mathcal{O}_D$; we will obtain a bit $b \leftarrow \mathcal{O}_D(c) = (c \bmod p) \bmod 2$. There are two cases to distinguish:

$\mathbf{b = 0}$ 'Decryption is correct' (since c is even); hence $p > c$, i.e. $p \geq l_p + 2h + 2$.
Update $l_p \leftarrow l_p + 2h + 2$.

b = 1 'Decryption is not correct'; hence $p < c$, i.e. $p \leq l_p + 2h$.
Update $u_p \leftarrow l_p + 2h$.

Next, we repeat the decryption query with the updated values for l_p, u_p and with another even 'ciphertext' $c \in [l_p+1, u_p-1]$, and we stop when $u_p = l_p$. In particular, for efficiency we always choose c as the even integer in the middle of the interval $[l_p+1, u_p-1]$. It is easy to see that this attack leads to a full recovery of the secret key p with at most $\log(2^{\eta-2}-2) \approx \eta$ oracle queries.

3.3 Algorithmic Description

Formally, the attack can be described by Algorithm 1. It takes as input an integer $\eta \in \mathbb{N}$ and outputs the secret integer p. Let

$$\mathcal{O}_D(c) = \mathsf{Decrypt}(c, sk) = (c \bmod p) \bmod 2 \quad \text{(decryption oracle)}$$
$$\lfloor x \rfloor_o = \max\{n \in \mathbb{N} \text{ s.t. } n \text{ is odd and } n \leq x\}$$

Algorithm 1 Key Recovery Attack

```
input: $\eta$
$l_p \leftarrow 2^{\eta-1} + 1$
$u_p \leftarrow 2^{\eta} - 1$
while $u_p \neq l_p$ do
    $h \leftarrow (u_p - l_p)/2$ {$h \in \mathbb{N}$ is the number of even values in $[l_p, u_p]$}
    $c \leftarrow l_p + \lfloor h \rfloor_o$
    if $\mathcal{O}_D(c) = 0$ then
        $l_p \leftarrow l_p + \lfloor h \rfloor_o + 1$
    end if
    if $\mathcal{O}_D(c) = 1$ then
        $u_p \leftarrow l_p + \lfloor h \rfloor_o - 1$
    end if
end while
return $u_p$
```

4 Key Recovery Attack against the BV11b Scheme

In this section, we describe a key recovery attack against the SHE scheme from [BV11b].

4.1 The BV11b SHE Scheme

The message space is $\mathcal{M} = \mathbb{Z}_2$. Let f be a polynomial in λ, i.e. $f(\lambda) = \text{poly}(\lambda)$. Consider $n = f(\lambda) \in \mathbb{N}$ and let $\epsilon \in (0,1) \cap \mathbb{R}$. Assume an odd integer $q \in \mathbb{N}$ such that $q \in [2^{n^\epsilon}, 2 \cdot 2^{n^\epsilon})$, and an integer $m \geq n\log q + 2\lambda$. Let χ be a noise distribution over $\mathbb{Z}_q$ (it produces small samples, all of magnitude not greater than n). Finally, let $L \in \mathbb{N}$ be an upper bound on the maximal multiplicative depth that the scheme can homomorphically evaluate, say $L \approx \epsilon\log n$.

$\mathsf{KeyGen}(\lambda)$

- pick $\mathbf{s_0}, \dots, \mathbf{s_L} \xleftarrow{\$} \mathbb{Z}_q^n$
- pick a matrix $A \xleftarrow{\$} \mathbb{Z}_q^{m \times n}$
- pick a vector $\mathbf{e} \leftarrow \chi^m$
- compute $\mathbf{b} = A\mathbf{s_0} + 2\mathbf{e}$
- $\mathsf{sk} = \mathbf{s_L}$
- $\mathsf{pk} = (A, \mathbf{b})$

$\mathsf{Encrypt}(\mathsf{pk}, \mu \in \mathcal{M})$

- pick $\mathbf{r} \xleftarrow{\$} \{0,1\}^m$
- set $\mathbf{v} = A^{\mathrm{T}}\mathbf{r} \in \mathbb{Z}_q^n$
- set $w = \mathbf{b}^{\mathrm{T}}\mathbf{r} + \mu \in \mathbb{Z}_q$
- ciphertext $c = ((\mathbf{v}, w), l)$.

$\mathsf{Decrypt}(\mathsf{sk}, c = ((\mathbf{v}, w), L))$

$$\mu = (w - < \mathbf{v}, \mathbf{s_L} > \bmod\ q) \bmod\ 2$$

Notice that the vectors $\mathbf{s}_1, \dots, \mathbf{s}_{L-1}$ are used in order to compute the evaluation key, which we omit here. We remark that during the homomorphic evaluation, the scheme generates ciphertexts of the form $c = ((\mathbf{v}, w), l)$, where the tag l indicates the multiplicative level at which the ciphertext has been generated (fresh ciphertexts are tagged with $l = 0$). Note that it always holds that $l \leq L$ due to the bound on the multiplicative depth, and that the output of the homomorphic evaluation of the entire circuit is expected to have $l = L$. As described in [BV11b], the SHE scheme is only required to decrypt ciphertexts that are output by the evaluation step (which we omit here), and those will always have level tag L. Therefore, we always expect a ciphertext of the form $c = ((\mathbf{v}, w), L)$ and decryption is correct.

Apparently, we cannot decrypt level l ciphertexts $c = ((\mathbf{v}, w), l)$, for $1 \leq l < L$, since we are only allowed to decrypt level L ciphertexts. However, we can compute $L - l$ fresh encryptions of 1, namely $c_1, \dots, c_{L-l}$. Then, we compute $c^* = \mathsf{Evaluate}(\mathsf{pk}, MUL, c, c_1, \dots, c_{L-l})$ based on the homomorphic property, where MUL is the multiplication circuit. The resulting ciphertext c^* will encrypt the same message as c does, and with a tag level L. In particular, we can decrypt fresh ciphertexts.

4.2 Our Key Recovery Attack

We are going to recover the secret key $\mathbf{s_L} \in \mathbb{Z}_q^n$ component-wise, and bit by bit. For ease of notation, we will write $\mathbf{s}$ instead of $\mathbf{s_L}$. More precisely, we write $\mathbf{s} = (s_1, \dots, s_n) \in \mathbb{Z}_q^n$. For every $1 \leq j \leq n$, we have $s_j \in \mathbb{Z}_q$ and therefore s_j can be written with a maximum number N of bits, where $N = \lfloor \log_2(q-1) \rfloor + 1$. We are going to recover the i-th bit of s_j, for all $1 \leq i \leq N$ and for all $1 \leq j \leq n$.

Intuitively, our attack works as follows. We start by finding the first bit of s_j for every $1 \leq j \leq n$; then we will recover the second bit of s_j for every $1 \leq j \leq n$; and we stop until we reach the N-th bit. In order to do so, we have to choose a 'ciphertext' c to be submitted to the decryption oracle. Instead of submitting $c = (\mathbf{v}, w)$ for honestly-generated $\mathbf{v} \in \mathbb{Z}_q^n$ and $w \in \mathbb{Z}_q$, we submit $c^* = (\mathbf{x}, y)$ for some specifically-picked $\mathbf{x} \in \mathbb{Z}_q^n$ and $y \in \mathbb{Z}_q$. We omit to write the level tag since we can always obtain a level tag L from any $l \leq L$.

For any $1 \leq j \leq n$, let $(s_j)_2 := a_{j,N}a_{j,N-1} \cdots a_{j,1}$ be the binary representation of s_j (bits ordered from most significant to least significant). We have $a_{j,i} \in \{0, 1\}$, for all $1 \leq i \leq N$.

Recovering $\mathbf{a_{j,1}}$

We have to choose $\mathbf{x} \in \mathbb{Z}_q^n$ and $y \in \mathbb{Z}_q$ in such a way that $y - < \mathbf{x}, \mathbf{s} > \bmod\ q = s_j$. To do so, pick $y = 0$ and $\mathbf{x} = (0, \dots, 0, -1, 0, \dots, 0)$ (where -1 is in position j). Then, we have $0-(-1)s_j \bmod q = s_j \bmod q = s_j$. As a result, by modding out with 2, this will return the last bit $a_{j,1}$ of s_j.

Recovering $\mathbf{a_{j,2}}$

Now that we know the last bit $a_{j,1}$ of s_j, we want to obtain $s_j^{(1)} := (s_j - a_{j,1})/2 \in \mathbb{Z}_q$ whose bit decomposition is the same as the bit decomposition of s_j, but with the last bit removed from it. Then, modding out by 2, we will get the desired bit. This translates to the following condition: find $\mathbf{x} \in \mathbb{Z}_q^n$ and $y \in \mathbb{Z}_q$ such that $y - < \mathbf{x}, \mathbf{s} > \bmod q = (s_j - a_{j,1})/2$. Let $\mathbf{x} = (0, \ldots, 0, x_j, 0, \ldots, 0)$ (with x_j in j-th position). We have to find y and x_j such that $2y - s_j(2x_j + 1) = -a_{j,1} \bmod q$. Clearly, the solution is given by $x_j = -2^{-1} \bmod q$ and $y = -2^{-1}a_{j,1} \bmod q$. By querying the decryption oracle with the 'ciphertext' $c^* := (\mathbf{x}, y)$, we obtain the second-to-last bit $a_{j,2}$ of s_j.

Recovering $\mathbf{a_{j,m}}$, for $\mathbf{1 \leq m \leq N}$

Based on the above two cases, we generalize the procedure. Suppose we have found all bits $a_{j,i}$, for $1 \leq i \leq m-1$. In order to recover the bit $a_{j,m}$, we choose $\mathbf{x} := (0, \ldots, 0, x_j, 0, \ldots, 0) \in \mathbb{Z}_q^n$ and $y \in \mathbb{Z}_q$ as follows: $x_j = -(2^{m-1})^{-1} \bmod q$ and $y = -(2^{m-1})^{-1}(\sum_{i=1}^{m-1} 2^{i-1}a_{j,i})$.

4.3 Algorithmic Description and Efficiency Analysis

We denote the decryption oracle $\mathcal{O}_D(c) := \mathsf{Decrypt}(\mathsf{sk}, c)$. The ciphertext c is of the form $c = (\mathbf{x}, y)$ (the level tag is omitted), with $\mathbf{x} \in \mathbb{Z}_q^n$, $y \in \mathbb{Z}_q$. For ease of notation, we have also considered the standard vectors $\mathbf{e}_1, \ldots, \mathbf{e}_n \in \mathbb{Z}_q^n$: for every $i = 1, \ldots, n$, $\mathbf{e}_i$ is the $\mathbf{0}$-vector except in position i, where it has value 1, i.e. $\mathbf{e}_i = (e_{i,1}, \ldots, e_{i,n}) = (0, \ldots, 0, 1, 0, \ldots, 0)$, $e_{i,i} = 1, e_{i,j} = 0$ for $j \neq i$. Formally, the attack from Section 4.2 can be described by Algorithm 2. It takes as input the integers n, q and returns the secret key vector $\mathbf{s} = (s_1, \ldots, s_n) \in \mathbb{Z}_q^n$.

Algorithm 2 Key Recovery Attack

```
input: $n, q \in \mathbb{N}$
$N \leftarrow \lfloor \log_2(q-1) \rfloor + 1$
for $j = 1$ to $n$ do
    for $m = 1$ to $N$ do
        $x_j \leftarrow -(2^{m-1})^{-1} \bmod q$
        $\mathbf{x} \leftarrow x_j \cdot \mathbf{e}_j$
        $y \leftarrow x_j \cdot \sum_{i=1}^{m-1} 2^{i-1} a_{j,i} \bmod q$        $\{\text{if } m = 1,\ y \leftarrow 0\}$
        $a_{j,m} \leftarrow \mathcal{O}_D(\mathbf{x}, y)$
    end for
    $s_j \leftarrow \sum_{m=1}^{N} 2^{m-1} a_{j,m}$
end for
$\mathbf{s} \leftarrow (s_1, \ldots, s_n)$
return $\mathbf{s}$
```

The secret key vector $\mathbf{s} = \mathbf{s_L} \in \mathbb{Z}_q^n$ has n coefficients s_j, and each one of them has length of at most N bits. Now, $n = f(\lambda)$ for a polynomial function $f(\lambda) = \text{poly}(\lambda)$, and

$N = \lfloor \log_2(q-1) \rfloor + 1$. We have that $q \in [2^{n^\epsilon}, 2^{n^\epsilon+1})$, with $\epsilon \in (0,1) \cap \mathbb{R}$ a constant, and

$$\lfloor \log_2(q-1) \rfloor + 1 < \lfloor \log_2 2^{n^\epsilon+1} \rfloor + 1 = \lfloor n^\epsilon + 1 \rfloor + 1 = \lfloor f(\lambda)^\epsilon + 1 \rfloor + 1 = g(\lambda)$$

where $g(\lambda) = \text{poly}(\lambda)^\epsilon$. Therefore, the total number of queries we must perform to recover $\mathbf{s}$ is $n \times N < f(\lambda) \cdot g(\lambda) = \text{poly}(\lambda)^{\epsilon+1}$. Since each query to the decryption oracle reveals one bit of $\mathbf{s}$, our attack is optimal and ends in polynomial time.

5 Key Recovery Attack against the BV11a Scheme

In this section, we describe a key recovery attack against the symmetric-key SHE scheme from [BV11a]. The attack also applies to the asymmetric-key SHE scheme.

5.1 The BV11a SHE Scheme

Consider primes $q = \text{poly}(\lambda) \in \mathbb{N}$, $t = \text{poly}(\lambda) \in \mathbb{Z}_q^*$. Let $n = \text{poly}(\lambda) \in \mathbb{N}$ and consider a polynomial $f(x) \in \mathbb{Z}[x]$ with $\deg(f) = n+1$. The message space is $\mathcal{M} = R_t = \mathbb{Z}[x]/(f(x))$. Namely, a message is encoded as a degree n polynomial with coefficients in $\mathbb{Z}_t$. Let χ be an error distribution over the ring $R_q := \mathbb{Z}_q[x]/(f(x))$ and let $D \in \mathbb{N}$, which is related to the maximal degree of homomorphism allowed (and to the maximal ciphertext length). Parameters n, f, q, χ are public.

```
Keygen($\lambda$)
    – sample $s \leftarrow \chi$
    – $\mathbf{s} = (1, s, s^2, \ldots, s^D) \in R_q^{D+1}$
    – $\mathsf{sk} = s$
```

```
Decrypt($\mathsf{sk}, \mathbf{c} = (c_0, \ldots, c_D) \in R_q^{D+1}$)
    $\mu = (< \mathbf{c}, \mathbf{s} > \text{mod} q) \bmod t$
```

```
Encrypt($\mathsf{sk}, \mu \in \mathcal{M}$)
    – sample $a \overset{\$}{\leftarrow} R_q$ and $e \leftarrow \chi$
    – compute $(a, b := as + te) \in R_q^2$
    – compute $c_0 := b + \mu \in R_q$, $c_1 := -a$
    – output $\mathbf{c} = (c_0, c_1) \in R_q^2$
```

We remark that while the encryption algorithm only generates ciphertexts $\mathbf{c} \in R_q^2$, homomorphic operations (as described in the evaluation algorithm which we omit here) might add more elements to the ciphertext. Thus, the most generic form of a decryptable ciphertext in this scheme is $\mathbf{c} = (c_0, \ldots, c_d) \in R_q^{d+1}$, for $d \leq D$. Notice that 'padding with zeros' does not affect the ciphertext. Namely, $(c_0, \ldots, c_d) \in R_q^{d+1}$ and $(c_0, \ldots, c_d, 0, \ldots, 0) \in R_q^{D+1}$ encrypt the same message $\mu \in R_t$.

5.2 Our Key Recovery Attack

We can write $s = s_0 + s_1 x + \cdots + s_n x^n \in \mathbb{Z}_q[x]/(f(x))$ with coefficients $s_j \in \mathbb{Z}_q$, $\forall 0 \leq j \leq n$. We will recover each coefficient s_j separately. Now, each s_j has at most $N := \lfloor \log_2(q-1) \rfloor + 1$ bits; therefore $\#\text{bits}(s) \leq (n+1) \times N = (n+1) \times (\lfloor \log_2(q-1) \rfloor + 1)$ and each query to the oracle decryption will reveal a polynomial $\mu(x) = \mu_0 + \mu_1 x + \cdots + \mu_n x^n \in \mathbb{Z}_t[x]/(f(x))$;

we have $\#\text{bits}(\mu) \leq (n+1) \times (\lfloor \log_2(t-1) \rfloor + 1)$. Therefore, the minimum number of oracle queries needed is given by

$$\left\lceil \frac{\#(\text{bits}(s))}{\#(\text{bits revealed by an oracle query})} \right\rceil = \left\lceil \frac{(n+1) \times (\lfloor \log_2(q-1) \rfloor + 1)}{(n+1) \times (\lfloor \log_2(t-1) \rfloor + 1)} \right\rceil = \left\lceil \frac{\lfloor \log_2(q-1) \rfloor + 1}{\lfloor \log_2(t-1) \rfloor + 1} \right\rceil$$

We are going to query the decryption oracle with 'ciphertexts' of the form $\mathbf{c}_i^* := (h_i, y_i, 0, \dots, 0) \in R_q^{D+1}$ for some $h_i, y_i \in R_q$. We will describe in detail our attack in the case $t = 2$. An easy generalization for $t \geq 2$ is discussed later.

An easy case: $t = 2$.

We expect to query the decryption oracle at least $\lfloor \log_2(q-1) \rfloor + 1$ times and recover s_j, for all $0 \leq j \leq n$, bit by bit. Let $N = \#\text{bits}(s_j) = \lfloor \log_2(q-1) \rfloor + 1$, $\forall 0 \leq j \leq n$; and let $(s_j)_2 = a_{j,N} a_{j,N-1} \cdots a_{j,1}$ be the binary representation of s_j, $\forall 0 \leq j \leq n$ (i.e., $a_{j,i} \in \{0,1\}, \forall 1 \leq i \leq N$ and bits ordered most significant to least significant). For ease of notation, we write $\mathbf{c}^* = (h, y)$ instead of $\mathbf{c}^* = (h, y, 0, \dots, 0)$.

Recovering $a_{j,1}$, for all $0 \leq j \leq n$

For a submitted 'ciphertext' $\mathbf{c}^* = (h, y)$, decryption works as follows: $< \mathbf{c}^*, \mathbf{s} > \bmod 2 = h + ys \bmod 2$. We choose $h = \sum_{j=0}^{n} 0x^j = 0 \in R_q$ and $y = 1 + \sum_{j=1}^{n} 0x^j = 1 \in R_q$. The decryption oracle outputs

$$\begin{aligned} s \bmod 2 &= (s_0 \bmod 2) + (s_1 \bmod 2)x + \cdots + (s_n \bmod 2)x^n \\ &= a_{0,1} + a_{1,1}x + \cdots + a_{n,1}x^n \end{aligned}$$

Therefore, we obtain the last bits $a_{j,1}$ for all $1 \leq j \leq n$, which are n bits of s.

Recovering $a_{j,2}$, $\forall 0 \leq j \leq n$

With $a_{j,1}$ for all $0 \leq j \leq n$, we are going to recover $a_{j,2}$, $\forall 0 \leq j \leq n$, as follows. We want to obtain

$$\begin{aligned} s^{(1)} &:= \frac{s - (a_{0,1} + a_{1,1}x + \cdots + a_{n,1}x^n)}{2} \\ &= s_0^{(1)} + s_1^{(1)}x + \cdots + s_n^{(1)}x^n \in \frac{\mathbb{Z}_q[x]}{(f(x))} \end{aligned}$$

for which the bit decomposition of the coefficients $s_j^{(1)}$ is the same as the bit decomposition of s_j, but with the last bit removed from it, for all $0 \leq j \leq n$. Then, by modding out with 2, we will get the desired bits. This translates to the following condition: find $\mathbf{c}^* = (h, y) = (h, y, 0, \dots, 0) \in R_q^{D+1}$ such that $< \mathbf{c}^*, \mathbf{s} >= s^{(1)} := \frac{s-(a_{0,1}+a_{1,1}x+\cdots+a_{n,1}x^n)}{2}$, from which we obtain $2h + s(2y-1) = -(a_{0,1} + a_{1,1}x + \cdots + a_{n,1}x^n)$. A solution is given by $y = 2^{-1} \in R_q$ and $h = -2^{-1}(a_{0,1} + a_{1,1}x + \cdots + a_{n,1}x^n) \in R_q$. Then, by modding out with 2 the 'decrypted ciphertext' $\mu =< \mathbf{c}^*, \mathbf{s} >$, we recover the second-to-last bits $a_{j,2}$, for all $0 \leq j \leq n$.

Recovering $a_{j,m}$, for $1 \leq m \leq N$, $0 \leq j \leq n$

Suppose we have found all bits $a_{j,i}$, $\forall 1 \leq i \leq m-1$ and $\forall 0 \leq j \leq n$. We want to recover $a_{j,m}$, $\forall 0 \leq j \leq n$. By a recursive argument, we find that we have to submit a 'ciphertext' $\mathbf{c}^* = (h, y)$ such that $y = (2^{m-1})^{-1} \in R_q$ and $h = -(2^{m-1})^{-1} \left(\sum_{j=0}^{n} d_j x^j\right)$ with $d_j = \sum_{i=1}^{m-1} 2^{i-1} a_{j,i}$.

This concludes the attack for the case $t = 2$. Efficiency-wise, the total number of oracle queries is $N = \lfloor \log_2(q-1) \rfloor + 1$, which is optimal.

The general case: $t \geq 2$.

We consider now the general case in which $t \geq 2$ is a prime number in $\mathbb{Z}_q^*$. We want to find $s = s_0 + s_1 x + \cdots + s_n x^n \in \mathbb{Z}_q[x]/(f(x))$ and expect to query the decryption oracle $\left\lceil \frac{\lfloor \log_2(q-1) \rfloor + 1}{\lfloor \log_2(t-1) \rfloor + 1} \right\rceil$ times. With each query to the decryption oracle, we are going to recover $M = \lfloor \log_2(t-1) \rfloor + 1$ bits of s_j, $\forall 0 \leq j \leq n$. The idea is that we are going to recover s_j, for all $0 \leq j \leq n$. In its representation in base t, s_j can be represented with N figures $a_{j,i} \in \{0, 1, \ldots, t-1\}$: $(s_j)_t = a_{j,N} a_{j,N-1} \cdots a_{j,1}$ where $N = \lfloor \log_t(q-1) \rfloor + 1$; each $a_{j,i}$ is bounded by $t-1$, which explains the value $M = \lfloor \log_2(t-1) \rfloor + 1$.

Recovering $a_{j,1}$, $\forall 0 \leq j \leq n$

For a submitted 'ciphertext' $\mathbf{c}^* = (h, y) = (h, y, 0, \ldots, 0) \in R_q^{D+1}$, decryption works as follows: $< \mathbf{c}^*, \mathbf{s} > \text{mod} t = x + ys \bmod t$. We choose $h = 0 \in R_q$ and $y = 1 \in R_q$. Then, the decryption oracle outputs

$$\begin{aligned} s \bmod t &= (s_0 \bmod t) + (s_1 \bmod t)x + \cdots + (s_n \bmod t)x^n \\ &= a_{0,1} + a_{1,1}x + \cdots + a_{n,1}x^n \end{aligned}$$

as we wanted.

Recovering $a_{j,m}$, $\forall 1 \leq m \leq N$, $\forall 0 \leq j \leq n$

Suppose we know $a_{j,i}$, $\forall 1 \leq i \leq m-1$, $\forall 0 \leq j \leq n$. We want to recover $a_{j,m}$, for all $0 \leq j \leq n$. To do so, we submit to the decryption oracle a 'ciphertext' $\mathbf{c}^* = (h, y)$ such that $y = (t^{m-1})^{-1} \in R_q$, $h = -(t^{m-1})^{-1} \left(\sum_{j=0}^{n} d_j x^j\right)$, $d_j = \sum_{i=1}^{m-1} t^{i-1} a_{j,i}$. It is straightforward to verify that it works and we skip the details here.

5.3 Algorithmic Description

Formally, the attack from Section 5.2 can be described by Algorithm 3. It takes as input integers $q, t, n, D \in \mathbb{N}$ and a polynomial $f(x) \in \mathbb{Z}[x]$ of degree n. It outputs the secret key vector $\mathbf{s} = (1, s, s^2, \ldots, s^D) \in R_q^{D+1}$, where $R_q := \mathbb{Z}_q[x]/(f(x))$. Let $\mathcal{O}_D(c) := \mathsf{Decrypt}(\mathsf{sk}, \mathbf{c})$ be the decryption oracle, where $\mathbf{c} \in R_q^{D+1}$. For given $h, y \in R_q$, let $\mathcal{O}_D(h, y) := \mathcal{O}_D((h, y, 0, \ldots, 0))$.

5.4 Key Recovery Attack against the BGV12 Scheme

The SHE scheme from [BGV12] is closely related to the SHE schemes from [BV11a,BV11b]. This implies that the attacks from Section 5.2 and 4.2 can be directly applied against the SHE scheme from [BGV12].

Algorithm 3 Key Recovery Attack

```
input: $q, t, n, D \in \mathbb{N}$; $f(x) \in \mathbb{Z}[x]$
$N \leftarrow \lfloor \log_t(q-1) \rfloor + 1$
for $m = 1$ to $N$ do
    $y \leftarrow (t^{m-1})^{-1}$ in $R_q$
    $h \leftarrow -y \cdot \sum_{i=1}^{m-1} t^{i-1} r_i$ in $R_q$        $\{$if $m = 1$, $h \leftarrow 0\}$
    $r_m \leftarrow \mathcal{O}_D(h, y)$
end for
$s \leftarrow \sum_{i=1}^{N} t^{i-1} r_i$
$\mathbf{s} \leftarrow (1, s, s^2, \ldots, s^D) \in R_q^{D+1}$
return $\mathbf{s}$
```

We first remark that the LWE and RLWE problems are syntactically equivalent. They only use different rings ($\mathbb{Z}$ for LWE, and a polynomial ring $\mathbb{Z}[x]/(x^d + 1)$ for RLWE), as well as different vector dimensions over these rings ($n = \text{poly}(\lambda)$ for LWE, $n = 1$ for RLWE). For this reason and to simplify the presentation, the authors of [BGV12] introduced the general learning with errors (GLWE) problem, which is a generalized version of LWE and RLWE.

Definition 4 (GLWE problem). *For security parameter λ, let $n = n(\lambda)$ be an integer dimension, let $f(x) = x^d + 1$ where $d = d(\lambda)$ is a power of 2, let $q = q(\lambda) \geq 2$ be a prime integer, let $R = \mathbb{Z}[x] = (f(x))$ and $R_q = R/qR$, and let $\chi = \chi(\lambda)$ be a distribution over R. The $\mathsf{GLWE}_{n,f,q,\chi}$ problem is to distinguish the following two distributions:*

- *one samples $(\mathbf{a}_i, b_i)$ uniformly from R_q^{n+1}.*
- *one first draws $\mathbf{s} \leftarrow R_q^n$ uniformly and then samples $(\mathbf{a}_i, b_i) \in R_q^{n+1}$ by sampling $\mathbf{a}_i \leftarrow R_q^n$ uniformly, $e_i \leftarrow \chi$, and setting $b_i =< \mathbf{a}_i, \mathbf{s} > +e_i$.*

The $\mathsf{GLWE}_{n,f,q,\chi}$ assumption is that the $\mathsf{GLWE}_{n,f,q,\chi}$ problem is infeasible.

LWE is GLWE when $d = 1$, and RLWE is GLWE when $n = 1$. Let's review the GLWE-based encryption scheme.

$\mathsf{Setup}(\lambda)$:

- use bit $b \in \{0, 1\}$ to determine whether we are setting parameters for a LWE-based scheme ($d = 1$) or a RLWE-based scheme ($n = 1$).
- choose μ-bit modulus q and d, n, N, χ (all polynomials in λ, μ, b) in order to have a GLWE-based scheme with 2^λ security against known attacks.
- Let $R = \mathbb{Z}[x]/(x^d + 1)$
- Let $\mathsf{params} = (q, d, n, N, \chi)$.

$\mathsf{SecretKeyGen}(\mathsf{params})$:

- choose $\mathbf{s}' \leftarrow \chi^n$.
- set $\mathsf{sk} = \mathbf{s} \leftarrow (1, \mathbf{s}'[1], \ldots, \mathbf{s}'[n]) \in R_q^{n+1}$.

$\mathsf{PublicKeyGen}(\mathsf{params}, \mathsf{sk})$:

- input: params and $\mathsf{sk} = \mathbf{s} = (1, \mathbf{s})$ with $\mathbf{s}[0] = 1$ and $\mathbf{s}' \in R_q^n$.
- generate matrix $\mathbf{A}' \xleftarrow{\$} R_q^{N \times n}$
- generate a vector $\mathbf{e} \leftarrow \chi^N$
- set $b \leftarrow \mathbf{A}'\mathbf{s}' + 2\mathbf{e}$.
- set $\mathbf{A}$ to be the $(n+1)$-column matrix consisting of $\mathbf{b}$ followed by the n columns of $-\mathbf{A}'$. (Remark: $\mathbf{A} \cdot \mathbf{s} = 2\mathbf{e}$.)
- set $\mathsf{pk} = \mathbf{A}$.

$\mathsf{Enc}(\mathsf{params}, \mathsf{pk}, m)$:

- input: message $m \in R_2$
- set $\mathbf{m} \leftarrow (m, 0, \ldots, 0) \in R_q^{n+1}$
- sample $\mathbf{r} \leftarrow R_2^N$

- output ciphertext $\mathbf{c} \leftarrow \mathbf{m} + \mathbf{A}^T\mathbf{r} \in R_q^{n+1}$.

$\mathsf{Dec}(\mathsf{params}, \mathsf{sk}, c)$**:** Output $m \leftarrow (< \mathbf{c}, \mathbf{s} > \bmod q) \bmod 2$.

The SHE scheme from [BGV12] uses the above GLWE-based encryption scheme as the main building block, and we only need to show attacks against the latter. Depending on which instantiation is chosen (either LWE or RLWE), we can apply one of the key recovery attacks against [BV11b,BV11a] to the basic GLWE-based encryption scheme.

- If $b = 0$, then $d = 1$, $R = \mathbb{Z}[x]/(x+1) \cong \mathbb{Z}$, and
$$\mathbf{c} := \mathbf{m} + A^{\mathrm{T}}\mathbf{r} = \mathbf{m} + (\mathbf{b} \mid -A'^{\mathrm{T}})\mathbf{r} = \begin{pmatrix} m + \mathbf{b}^{\mathrm{T}}\mathbf{r} \\ -A'^{\mathrm{T}}\mathbf{r} \end{pmatrix}$$
which can be written as $\mathbf{c} = \binom{w}{-\mathbf{v}} \in \mathbb{Z}_q^{n+1}$. For decryption we have
$$m := (< \mathbf{c}, \mathbf{s} > \bmod q) \bmod 2 = (m + \mathbf{b}^{\mathrm{T}}\mathbf{r} + < -A'^{\mathrm{T}}\mathbf{r}, \mathbf{s} > \bmod q) \bmod 2$$
which is $(w - < \mathbf{v}, \mathbf{s}_L > \bmod q) \bmod 2$. The secret key can be recovered by directly applying the key recovery attack from Section 4.2.
- If $b = 1$, then $n = 1$, $R = \mathbb{Z}[x]/(x^d + 1)$. The scheme is slightly different from the BV11a SHE scheme just for the encryption part, but the setup, the key generation and the decryption steps are the same. Therefore, our key recovery attack can be applied. Precisely, to recover the secret polynomial $\mathbf{s} := s(x) = s_0 + s_1 x + \cdots + s_{d-1}x^{d-1} \in \mathbb{Z}[x]/(x^d + 1)$, one could directly use our key recovery attack from Section 5.2 with the following settings: $D \leftarrow 1, n \leftarrow d-1, t \leftarrow 2$.

6 Key Recovery Attack against the Bra12 SHE Scheme

In this section, we describe a key recovery attack against the SHE scheme from [Bra12]. The scheme uses, as a building block, Regev's [Reg05] public-key encryption scheme. It is then enough to show a key recovery attack on Regev's scheme since the full [Bra12] can be attacked exactly as the basic Regev's encryption scheme (the only differences between the two schemes are in the evaluation step which is missing in Regev's scheme).

Let's first recall Regev's encryption scheme. In this scheme, let $n := \lambda$ be the security parameter.

6.1 The Bra12 SHE Scheme (Regev's Encryption Scheme)

Let q be a prime number and let $\chi = \chi(n)$ be a distribution ensemble over $\mathbb{Z}$. The message space is $\mathcal{M} = \{0, 1\}$. As claimed in [Reg05], choosing q such that $n^2 \leq q \leq 2n^2$ is enough for security (in particular, $q = \mathrm{poly}(n)$; for other parameters settings, see [Reg05]).

$\mathsf{SecretKeyGen}(n)$:

- Sample $\mathbf{s} := (s_1, \ldots, s_n) \xleftarrow{\$} \mathbb{Z}_q^n$
- Output $\mathsf{sk} = \mathbf{s}$

$\mathsf{PublicKeyGen}(\mathbf{s})$**:**

- Let $N := (n+1) \cdot (\log q + O(1))$
- Sample $\mathbf{A} \xleftarrow{\$} \mathbb{Z}_q^{N \times n}$

- Sample $\mathbf{e} \leftarrow \chi^N$
- Compute $\mathbf{b} := \mathbf{A} \cdot \mathbf{s} + \mathbf{e} \bmod q$
- Define $\mathbf{P} := [\mathbf{b}| - \mathbf{A}] \in \mathbb{Z}_q^{N \times (n+1)}$
- Output $\mathsf{pk} = \mathbf{P}$

$\mathsf{Encrypt}(\mathsf{pk}, m \in \{0,1\})$:

- Sample $\mathbf{r} \in \{0,1\}^N$
- Let $\mathbf{m} := (m, 0, \ldots, 0) \in \{0,1\}^{n+1}$

$$\mathbf{c} := \mathbf{P}^T \cdot \mathbf{r} + \lfloor q/2 \rfloor \cdot \mathbf{m} \in \mathbb{Z}_q^{n+1}$$

$\mathsf{Decrypt}(\mathsf{sk}, \mathbf{c})$:

$$m := \left\lfloor \frac{2}{q} \cdot (< \mathbf{c}, (1, \mathbf{s}) > \mathrm{mod} q) \right\rceil \bmod 2$$

6.2 Our Key Recovery Attack

Recall that we defined the rounding function $\lfloor \cdot \rceil$ such that $\lfloor m + 1/2 \rceil := m + 1$ for every $m \in \mathbb{N}$. The following attack works also, with trivial modifications, in case we define $\lfloor m + 1/2 \rceil := m$. In this section, for a given ciphertext $\mathbf{c}$ we use notation $D(\mathbf{c})$ instead of $\mathsf{Decrypt}(\mathsf{sk}, \mathbf{c})$.

We will start by describing how to recover s_1. An easy generalization will allow to recover $s_j, \forall j = 1, 2, \ldots, n$. We are going to submit to the decryption oracle 'ciphertexts' of the form $\mathbf{c} = (c_1, c_2, \ldots, c_{n+1}) \in \mathbb{Z}_q^{n+1}$. It holds

$$< \mathbf{c}, (1, \mathbf{s}) > \mathrm{mod} q = c_1 + c_2 s_1 + c_3 s_2 + \cdots + c_{n+1} s_n \bmod q$$

Choose $\mathbf{c} = (0, 1, 0, \ldots, 0)$, i.e. $c_2 = 1$ and $c_i = 0$, for $i = 1, \ldots, n+1$ and $i \neq 2$. Then $< \mathbf{c}, (1, \mathbf{s}) > \mathrm{mod} q = s_1 \bmod q = s_1$. (Recall that $s_j \leq q - 1$, $\forall j = 1, 2, \ldots, n$.) Then

$$D(\mathbf{c}) = \left\lfloor \frac{2}{q} s_1 \right\rceil \bmod 2$$

Now, since $0 \leq s_1 < q$, we have $0 \leq \frac{2}{q} s_1 < \frac{2}{q} q = 2$. Let $u = \left\lfloor \frac{2}{q} s_1 \right\rceil$; then we have $u \in \{0, 1, 2\}$. In particular, it is easy to see that

- $u = 0 \Leftrightarrow 0 \leq s_1 < \frac{q}{4}$
- $u = 1 \Leftrightarrow \frac{q}{4} \leq s_1 < \frac{3q}{4}$
- $u = 2 \Leftrightarrow \frac{3q}{4} \leq s_1 \leq q - 1$

Remember that q is prime, so in particular $\frac{q}{4}, \frac{3q}{4} \notin \mathbb{N}$. Since $D(\mathbf{c}) = u \bmod 2$, we have

- $D(\mathbf{c}) = 1 \Leftrightarrow \frac{q}{4} < s_1 < \frac{3q}{4}$
- $D(\mathbf{c}) = 0 \Leftrightarrow 0 \leq s_1 < \frac{q}{4}$ or $\frac{3q}{4} < s_1 \leq q - 1$

Having these considerations in mind, we recover s_1 like follows.

Recovering s_1.
Select $\mathbf{c} = (0, 1, 0, \ldots, 0) \in \mathbb{Z}_q^{n+1}$, i.e. $c_2 = 1$ and $c_i = 0$, $\forall i = 1, \ldots, n+1$, $i \neq 2$. Submit $\mathbf{c}$ to the decryption oracle. We have two case to consider.

Case 1: $D(\mathbf{c}) = 1$. Then we know that

$$\frac{q}{4} < s_1 < \frac{3q}{4} \tag{1}$$

Now select $\mathbf{c} = (1, 1, 0, \dots, 0) \in \mathbb{Z}_q^{n+1}$. Then $< \mathbf{c}, (1, \mathbf{s}) >= 1 + s_1 \bmod q$. There are two cases to consider:

1.1 if $D(\mathbf{c}) = 0$, then it must be

$$\frac{3q}{4} < 1 + s_1 \tag{2}$$

Conditions 1 and 2 together imply that s_1 is the biggest integer smaller than $\frac{3q}{4}$, i.e. $s_1 = \lfloor \frac{3q}{4} \rfloor$.

1.2 if $D(\mathbf{c}) = 1$, then we still have

$$\frac{q}{4} < 1 + s_1 < \frac{3q}{4} \tag{3}$$

We then select $\mathbf{c} = (2, 1, 0, \dots, 0) \in \mathbb{Z}_q^{n+1}$ and submit it to the decryption oracle. Similarly as above, we have $< \mathbf{c}, (1, \mathbf{s}) >= 2 + s_1 \bmod q$. Again, there are two cases to consider:

1.2.1 if $D(\mathbf{c}) = 0$, then it must be

$$\frac{3q}{4} < 2 + s_1 \tag{4}$$

Conditions 3 and 4 together imply that $1 + s_1 = \lfloor \frac{3q}{4} \rfloor$, i.e. $s_1 = \lfloor \frac{3q}{4} \rfloor - 1$.

1.2.2 if $D(\mathbf{c}) = 1$, then we still have

$$\frac{q}{4} < 2 + s_1 < \frac{3q}{4}$$

We keep reasoning this way, submitting to the decryption oracle 'ciphertexts' $\mathbf{c}_i = (c_{1,i}, 1, 0, \dots, 0) \in \mathbb{Z}_q^{n+1}$, for increasing values $c_{1,i} = 1, 2, 3, \dots$ until we obtain $D(\mathbf{c}_i) = 0$. Then we will have

$$s_1 = \lfloor \frac{3q}{4} \rfloor - c_{1,i} + 1$$

We notice that, in the worst case, i.e. when $s_1 = \lceil \frac{q}{4} \rceil$, we have to query the decryption oracle at most $M_1 := \lceil \frac{3q}{4} \rceil - \lceil \frac{q}{4} \rceil$ times. Therefore, in the worst case the total number of oracle queries is

$$T_1 := 1 + M_1 = 1 + \lceil \frac{3q}{4} \rceil - \lceil \frac{q}{4} \rceil \approx \frac{q}{2}$$

Case 2: $D(\mathbf{c}) = 0$. Then we know that s_1 is such that

(2.1) $0 \leq s_1 < \frac{q}{4}$ or

(2.2) $\frac{3q}{4} < s_1 \leq q - 1$

We use techniques as before, but we have to be more careful since now we have to understand in which case we are among (2.1) or (2.2). As before, we select $\mathbf{c} = (1, 1, 0, \dots, 0) \in \mathbb{Z}_q^{n+1}$. Then $< \mathbf{c}, (1, \mathbf{s}) >= 1 + s_1 \bmod q$.

The idea is similar to case 1: we keep submitting to the decryption oracle 'ciphertexts' $\mathbf{c}_i = (c_{1,i}, 1, 0, \dots, 0) \in \mathbb{Z}_q^{n+1}$, for increasing values $c_{1,i} = 1, 2, 3, \dots$, until $D(\mathbf{c}_i) = 1$. When we will receive $D(\mathbf{c}_i) = 1$, we will know that $s_1 + c_{1,i} > \frac{q}{4}$. The exact value $c_{1,i}$ will tell us in which of the cases (2.1) or (2.2) we were at the beginning. In fact,

– in case (2.1) we will get $D(\mathbf{c}_i) = 1$ after a number of oracle queries M_2' such that

$$1 \leq M_2' \leq \left\lceil \frac{q}{4} \right\rceil$$

where $M_2' = 1$ when $s_1 = \lfloor \frac{q}{4} \rfloor$ and $M_2' = \lceil \frac{q}{4} \rceil$ when $s_1 = 0$.

- in case (2.2) the number M_2'' of oracle queries needed in order to obtain $D(\mathbf{c}_i) = 1$ is such that

$$1 + \left\lceil \frac{q}{4} \right\rceil \leq M_2'' \leq q - \left\lceil \frac{3q}{4} \right\rceil + \left\lceil \frac{q}{4} \right\rceil$$

  where $M_2'' = 1 + \lceil \frac{q}{4} \rceil$ when $s_1 = q - 1$ and $M_2'' = q - \lceil \frac{3q}{4} \rceil + \lceil \frac{q}{4} \rceil$ when $s_1 = \lceil \frac{3q}{4} \rceil$.

Therefore, consider the first value $c_{1,i}$ such that $D(\mathbf{c}_i) = 1$.

- if $1 \leq c_{1,i} \leq \lceil \frac{q}{4} \rceil$, we are in case (2.1) and

$$s_1 = \left\lfloor \frac{q}{4} \right\rfloor - c_{1,i} + 1$$

- if $1 + \lceil \frac{q}{4} \rceil \leq c_{1,i} \leq q - \lceil \frac{3q}{4} \rceil + \lceil \frac{q}{4} \rceil$ we are in case (2.2) and

$$s_1 = q - c_{1,i} + \left\lceil \frac{q}{4} \right\rceil$$

We notice that, in the worst case (i.e., when $s_1 = \lceil \frac{3q}{4} \rceil$) we need to query the decryption oracle $M_0 := q - \lceil \frac{3q}{4} \rceil + \lceil \frac{q}{4} \rceil$ times. (Notice that $M_0 = q - M_1$.) Therefore, in case 2, in the worst case the total number of oracle queries is

$$T_2 := 1 + M_0 = 1 + q - \left\lceil \frac{3q}{4} \right\rceil + \left\lceil \frac{q}{4} \right\rceil \approx \frac{q}{2}$$

So in both cases 1 and 2, the total number of oracle queries needed to recover s_1 is $\approx \frac{q}{2}$.

Remark 1. We can provide an exact simpler formula for T_1 and T_2. Recall that $q \geq 2$ is prime; we can reasonably assume q odd. Then one can check that

- if $q \equiv 1 \bmod 4$

$$M_1 = \frac{q-1}{2}, M_0 = \frac{q+1}{2}, T_1 = \frac{q+1}{2}, T_2 = \frac{q+3}{2}$$

- if $q \equiv 3 \bmod 4$

$$M_1 = \frac{q+1}{2}, M_0 = \frac{q-1}{2}, T_1 = \frac{q+3}{2}, T_2 = \frac{q+1}{2}$$

In particular, the total number T_{tot} of oracle queries needed to recover s_1 is $T_{\text{tot}} \leq \frac{q+3}{2}$.

An optimization. We could optimize the previous algorithm like follows. Let $b := D(\mathbf{c}) \in \{0, 1\}$, where $\mathbf{c} = (0, 1, 0, \ldots, 0) \in \mathbb{Z}_q^{n+1}$. Our previous strategy was to submit 'ciphertexts' $\mathbf{c}_i := (c_{1,i}, 1, 0, \ldots, 0) \in \mathbb{Z}_q^{n+1}$ for *increasing* values $c_{1,i} = i$, for $i = 1, 2, \ldots, M_b$.

We modify our strategy and choose the first value $c_{1,1}$ in the middle of the interval $[1, M_b]$. Then, if $D(\mathbf{c}_1) = 1 + b \bmod 2$ we choose $c_{1,2}$ in the middle of the interval $[1, c_{1,1}]$; otherwise, if $D(\mathbf{c}_1) = b \bmod 2$, we choose $c_{1,2}$ in the middle of the interval $[c_{1,1} + 1, M_b]$. Keep reasoning this way, we will obtain s_1 in $\lfloor \log_2(M_b) + 1 \rfloor \approx \log_2(q/2)$ oracle queries.

Recovering s_j, for $j = 1, \ldots, n$.

Similarly, and more in general, we can recover s_j, for $j = 1, 2, \ldots, n$. In this case, the 'ciphertext' to submit is $\mathbf{c} = (c_1, c_2, \ldots, c_{n+1}) \in \mathbb{Z}_q^{n+1}$ with

$$c_k = \begin{cases} 0 & \text{if } k = 1 \text{ and it is the first query to the decryption oracle} \\ c_{1,i} & \text{if } k = 1 \text{ and it is not the first query, } 1 \leq c_{1,i} \leq M_b \\ 1 & \text{if } k = j + 1 \\ 0 & \text{if } k \notin \{1, j + 1\} \end{cases}$$

6.3 Algorithmic Description and Efficiency

For a given vector $\mathbf{c} = (c_1, \ldots, c_{n+1}) \in \mathbb{Z}_q^{n+1}$, we denote the decryption oracle as $\mathcal{O}_D(\mathbf{c}) := \mathsf{Decrypt}(\mathsf{sk}, \mathbf{c})$. For ease of notation, we define the following function. Let $j \in \{1, 2, \ldots, n\}$, and $a, b \in \mathbb{Z}_q$; define the function $f_j : \mathbb{Z}_q^2 \to \mathbb{Z}_q^{n+1}$ such that $f_j(b_1, b_2) = (a_1, \ldots, a_{n+1})$ with $a_1 = b_1$, $a_{j+1} = b_2$ and $a_k = 0$ for $k \neq 1, j+1$. Algorithm 4 takes as input the integers n, q and returns the secret key $\mathbf{s} = (s_1, \ldots, s_n) \in \mathbb{Z}_q^n$. Algorithm 5 is the optimized version of it.

Algorithm 4 Key-Recovery Attack

```
input: $q, n \in \mathbb{N}$
for $j = 1$ to $n$ do
  $\mathbf{c} \leftarrow f_j(0, 1)$, $b \leftarrow \mathcal{O}_D(\mathbf{c})$, $b' \leftarrow b$, $i \leftarrow 0$
  while $b' = b$ do
    $i = i + 1$
    $\mathbf{c} \leftarrow f_j(i, 1)$
    $b' \leftarrow \mathcal{O}_D(\mathbf{c})$
  end while
  if $b = 1$ then
    $s_j \leftarrow \left\lfloor \frac{3q}{4} \right\rfloor - i + 1$
  else if $b = 0$ then
    if $1 \leq i \leq \left\lceil \frac{q}{4} \right\rceil$ then
      $s_j \leftarrow \left\lfloor \frac{q}{4} \right\rfloor - i + 1$
    else if $1 + \left\lceil \frac{q}{4} \right\rceil \leq i \leq q - \left\lceil \frac{3q}{4} \right\rceil + \left\lceil \frac{q}{4} \right\rceil$ then
      $s_j \leftarrow q - i + \left\lceil \frac{q}{4} \right\rceil$
    end if
  end if
end for
return $\mathbf{s} := (s_1, \ldots, s_n)$
```

Notice that $\max(M_0, M_1) = (q+1)/2 =: M$. Therefore, in the worst case the total number T_{tot} of oracle queries needed to recover $\mathbf{s} = (s_1, \ldots, s_n) \in \mathbb{Z}_q^n$ is

$$T_{\text{tot}} \leq n \cdot (1 + M) = n \cdot \frac{q+3}{2} \approx n \cdot \frac{q}{2}$$

In the optimized version, the total number $T_{\text{tot}}^{\text{opt}}$ of oracle queries is

$$T_{\text{tot}}^{\text{opt}} \leq n \cdot (1 + \lfloor \log_2(M) + 1 \rfloor) \approx n \cdot (1 + \log_2(q/2))$$

Therefore, our optimized key recovery algorithm is indeed optimal since the number of oracle queries needed to recover the secret key is not greater than the bits of the secret key (and one oracle query reveals on bit at a time). In fact:

$$\#\text{bits in } \mathsf{sk} = n \cdot (1 + \lfloor \log_2(q-1) \rfloor)$$

$$T_{\text{tot}}^{\text{opt}} \leq n \cdot \left(1 + \left\lfloor \log_2\left(\frac{q+1}{2}\right) + 1 \right\rfloor\right) = \left(1 + \left\lfloor \log_2\left(\frac{q+1}{2} \cdot 2\right) \right\rfloor\right) = n \cdot (1 + \lfloor \log_2(q+1) \rfloor)$$

In particular the above algorithm is polynomial in the security parameter n: recall that the suggested parameters in Regev's encryption scheme [Reg05] for q are $n^2 \leq q \leq 2n^2$; therefore we have $T_{\text{tot}}^{\text{opt}} = O(n\log\, n)$.

Algorithm 5 Optimized Key-Recovery Attack

```
input: $q, n \in \mathbb{N}$
if $q \equiv 1 \bmod 4$ then
    $M_1 \leftarrow \frac{q-1}{2}$, $M_0 \leftarrow \frac{q+1}{2}$
else if $q \equiv 3 \bmod 4$ then
    $M_1 \leftarrow \frac{q+1}{2}$, $M_0 \leftarrow \frac{q-1}{2}$
end if
for $j = 1$ to $n$ do
    $\mathbf{c} \leftarrow f_j(0,1)$, $b \leftarrow \mathcal{O}_D(\mathbf{c})$, $L \leftarrow 1$, $U \leftarrow M_b$
    while $L \neq U$ do
        $i \leftarrow \left\lfloor \frac{L+U}{2} \right\rfloor$
        $\mathbf{c} \leftarrow f_j(i,1)$
        $b' \leftarrow \mathcal{O}_D(\mathbf{c})$
        if $b' \neq b$ then
            $U \leftarrow i$
        else if $b' = b$ then
            $L \leftarrow i + 1$
        end if
    end while
    if $b = 1$ then
        $s_j \leftarrow \left\lfloor \frac{3q}{4} \right\rfloor - L + 1$
    else if $b = 0$ then
        if $1 \leq L \leq \lceil \frac{q}{4} \rceil$ then
            $s_j \leftarrow \lfloor \frac{q}{4} \rfloor - L + 1$
        else if $1 + \lceil \frac{q}{4} \rceil \leq L \leq q - \left\lceil \frac{3q}{4} \right\rceil + \lceil \frac{q}{4} \rceil$ then
            $s_j \leftarrow q - L + \lceil \frac{q}{4} \rceil$
        end if
    end if
end for
return $\mathbf{s} := (s_1, \ldots, s_n)$
```

7 Key Recovery Attack against the GSW13 SHE Scheme

In this section, we describe a key recovery attack against the SHE scheme from [GSW13]. We first give some useful preliminary definitions. Let $q, k \in \mathbb{N}$. Let l be the bit-length of q, i.e. $l = \lfloor \log_2 q \rfloor + 1$, and let $N = k \cdot l$. Consider a vector $\mathbf{a} := (a_1, \ldots, a_k) \in \mathbb{Z}_q^k$, and let $(a_i)_2 := a_{i,0}a_{i,1} \ldots a_{i,l-1}$ be the binary decomposition of a_i (bit ordered least to most significant), for every $i = 1, \ldots, k$. We define

$$\mathsf{BitDecomp}(\mathbf{a}) := (a_{1,0}, \ldots, a_{1,l-1}, \ldots, a_{k,0}, \ldots, a_{k,l-1}) \in \mathbb{Z}_q^N$$

For a given $\mathbf{a}' := (a_{1,0}, \ldots, a_{1,l-1}, \ldots, a_{k,0}, \ldots, a_{k,l-1}) \in \mathbb{Z}_q^N$, let

$$\mathsf{BitDecomp}^{-1}(\mathbf{a}') := (\sum_{j=0}^{l-1} 2^j \cdot a_{1,j}, \ldots, \sum_{j=0}^{l-1} 2^j \cdot a_{k,j}) \in \mathbb{Z}_q^k$$

We notice explicitly that $\mathbf{a}'$ does not necessarily lie in $\{0,1\}^N$, but when it does then $\mathsf{BitDecomp}^{-1}$ is the inverse of $\mathsf{BitDecomp}$. For $\mathbf{a}' \in \mathbb{Z}_q^N$, we define

$$\mathsf{Flatten}(\mathbf{a}') := \mathsf{BitDecomp}(\mathsf{BitDecomp}^{-1}(\mathbf{a}')) \in \mathbb{Z}_q^N$$

When A is a matrix, let $\mathsf{BitDecomp}(A), \mathsf{BitDecomp}^{-1}(A), \mathsf{Flatten}(A)$ be the matrix formed by applying the operation to each row of A separately. Finally, for $\mathbf{b} := (b_1, \ldots, b_k) \in \mathbb{Z}_q$ let

$$\mathsf{PowersOf2}(\mathbf{b}) := (b_1, 2b_1, \ldots, 2^{l-1}b_1, \ldots, b_k, 2b_k, \ldots, 2^{l-1}b_k) \in \mathbb{Z}_q^N$$

It is easy to see that, for $\mathbf{a}, \mathbf{b} \in \mathbb{Z}_q^k$ and for $\mathbf{a}' \in \mathbb{Z}_q^N$,

- $< \mathsf{BitDecomp}(\mathbf{a}), \mathsf{Powersof2}(\mathbf{b}) >=< \mathbf{a}, \mathbf{b} >$
- $< \mathbf{a}', \mathsf{Powersof2}(\mathbf{b}) >=< \mathsf{BitDecomp}^{-1}(\mathbf{a}'), \mathbf{b} >=< \mathsf{Flatten}(\mathbf{a}'), \mathsf{Powersof2}(\mathbf{b}) >$

7.1 The GSW13 SHE Scheme

The message space is $\mathcal{M} = \mathbb{Z}_q$ for a given modulus q with # bits$(q) = \kappa = \kappa(\lambda, L)$. Let $n = n(\lambda)$ be the lattice dimension and let $\chi = \chi(\lambda)$ be the error distribution over $\mathbb{Z}_q$ (chosen appropriately for LWE: it must achieve at least 2^λ security against known attacks). Choose $m = m(\lambda) = O(n\log q)$. So the parameters used in all algorithms are n, q, χ, m. We have that $l = \lfloor \log q \rfloor + 1$ is the number of bits of q, and we let $N = (n+1) \cdot l$.

$\mathsf{Keygen}(\lambda)$**:**

- sample $\mathbf{t} := (t_1, \ldots, t_n) \leftarrow \mathbb{Z}_q^n$
- $\mathsf{sk} := \mathbf{s} \leftarrow (1, -t_1, \ldots, -t_n) \in \mathbb{Z}_q^{n+1}$
- let $\mathbf{v} = \mathsf{Powersof2}(\mathbf{s}) \in \mathbb{Z}_q^N$; see [1]
- sample a matrix $B \overset{\$}{\leftarrow} \mathbb{Z}_q^{m \times n}$
- sample a vector $\mathbf{e} \leftarrow \chi$, $\mathbf{e} \in \mathbb{Z}_q^m$
- set $\mathbf{b} := B \cdot \mathbf{t} + \mathbf{e} =: (b_1, \ldots, b_m) \in \mathbb{Z}_q^m$.
- set A to be the $(n+1)$-column matrix consisting of $\mathbf{b}$ followed by the n columns of B

$$A = (\mathbf{b} \mid B) \in \mathbb{Z}_q^{m \times (n+1)}$$

- $\mathsf{pk} := A$.

We remark that $A \cdot \mathbf{s} = \mathbf{e}$.

$\mathsf{Encrypt}(\mathsf{pk}, \mu \in \mathcal{M})$**:**

- sample a matrix $R \overset{\$}{\leftarrow} \{0,1\}^{N \times m}$
- output the ciphertext

$$C = \mathsf{Flatten}(\mu \cdot I_N + \mathsf{BitDecomp}(R \cdot A)) \in \mathbb{Z}_q^{N \times N}$$

$\mathsf{Decrypt}(\mathsf{sk}, C)$**:**

- observe that the first l coefficients of $\mathbf{v}$ are $1, 2, \ldots, 2^{l-2}$
- among these coefficients, let $v_i = 2^i$ be in $(q/4, q/2]$
- let C_i be the i-th row of C
- compute $x_i :=< C_i, \mathbf{v} >$
- output $\mu' := \lfloor x_i / v_i \rceil$

The $\mathsf{Decrypt}$ algorithm can recover the message μ when it is in a 'small space' ($q = 2$, i.e. $\mathcal{M} = \mathbb{Z}_2$). For an algorithm that can recover any $\mu \in \mathbb{Z}_q$, we refer to the MPDec algorithm as described (as a special case) in [GSW13] and in [MP11]. If the ciphertext is generated correctly, it is not difficult to show that $C \cdot \mathbf{v} = \mu \cdot \mathbf{v} + R \cdot A \cdot \mathbf{s} = \mu \cdot \mathbf{v} + R \cdot \mathbf{e} \in \mathbb{Z}_q^N$.

Now, the $\mathsf{Decrypt}$ algorithm uses only the i-th coefficient of the vector $C \cdot \mathbf{v} \in \mathbb{Z}_q^N$, i.e. $< C_i, \mathbf{v} >= \mu \cdot v_i + < R_i, \mathbf{e} > \in \mathbb{Z}_q$. Moreover, in the $\mathsf{Decrypt}$ step, i has to be such that $v_i := 2^i \in (q/4, q/2]$, with $i \in [1, 2, \ldots, 2^{l-1}]$. Now remember that $l = \lfloor \log q \rfloor + 1$ equals the number of bits of q. Hence we have

$$2^{l-3} \leq \frac{q}{4} < 2^{l-2} \leq \frac{q}{2} < 2^{l-1} \leq q < 2^l$$

Therefore the only possible value for $2^i \in (q/4, q/2]$ is 2^{l-2}. For this reason, $\mathsf{Decrypt}$ can be simply rewritten as

$\mathsf{Decrypt}(\mathsf{sk}, C)$**:**

[1] $\mathbf{v} = \mathsf{Powersof2}(\mathbf{s}) = (s_1, 2s_1, \ldots, 2^{l-1}s_1, s_2, \ldots, 2^{l-1}s_2, \ldots, s_{n+1}, 2s_{n+1}, \ldots, 2^{l-1}s_{n+1})$
$= (1, 2, \ldots, 2^{l-1}, -t_1, -2t_1, \ldots, -2^{l-1}t_1, \ldots, -t_n, -2t_n, \ldots, -2^{l-1}t_n) \in \mathbb{Z}_q^{(n+1)l} = \mathbb{Z}_q^N$

– let C_{l-2} be the $(l-2)$-th row of C
– compute $x_{l-2} :=< C_{l-2}, \mathbf{v} >$
– output $\mu' := \lfloor x_{l-2}/2^{l-2} \rceil$

One could think of outputting as ciphertext only the $(l-2)$-th row C_{l-2} of the matrix C; this is actually not possible since the full matrix is still needed in order to perform the homomorphic operations (in particular, the multiplication of two ciphertexts). We will not discuss them here; see [GSW13].

7.2 Our Key Recovery Attack

We are going to recover bit by bit each coefficient t_i of the secret vector $\mathbf{t} := (t_1, \dots, t_n) \in \mathbb{Z}_q^n$. For every $1 \le i \le n$, let $\mathsf{BitDecomp}(t_i) := (t_{i,0}, t_{i,1}, \dots, t_{i,l-1}) \in \mathbb{Z}_q^l$ bits ordered from least to most significant. We explicitly remark that $t_i = \sum_{j=0}^{l-1} 2^j t_{i,j}$. We will proceed as follows: start with $i = 1$ and recover, in this order, the bits from most to least significant. Then continue with $i = 2$, and so on until $i = n$. Let $x \in \mathbb{Z}_q$. Since $\#\text{bits}(q) = l$, we have $x \le q - 1 \le 2^l - 2$. Moreover, we have $\#\text{bits}(x) \le \lfloor \log_2(q-1) \rfloor + 1 := l^*$. We have $l^* = l$ if q is not a power of 2, i.e. if $q \ne 2^h$, for any $h \in \{1, 2, \dots, l-1\}$. Otherwise, $l^* = l - 1$. We will not distinguish between these two cases: just remark that if $l^* = l - 1$, then $t_{i,l-1} = 0$ for all $i \in \{1, 2, \dots, n\}$.

<u>**Recovering** $\mathsf{BitDecomp(t_1)}$</u>

We start by recovering $\mathsf{BitDecomp(t_1)}$. The trickiest part is to recover the most significant bit. We start by recovering $t_{1,l-1}, t_{1,l-2}, t_{1,l-3}$. We have to choose, and submit to the decryption oracle, a matrix $C \in \mathbb{Z}_q^{N \times N}$. Then the oracle will compute $x =< C_{l-2}, \mathbf{v} >$ and will output the rounded value $\mu = \lfloor x/2^{l-2} \rceil$. Our attack works also, with a trivial modification, in the case we define the rounding function such that $\lfloor n + 1/2 \rceil := n$, for every $n \in \mathbb{N}$. Our strategy is to submit a matrix C whose entries are all 0 except for the $(l-2)$-th row C_{l-2}. Let $\mathbf{y} = (y_1, \dots, y_N) \in \mathbb{Z}_q^N$ be the vector representing C_{l-2}.

We select $\mathbf{y} = (0, \dots, 0, -1, 0, \dots, 0) \in \mathbb{Z}_q^N$ where -1 is in $l+1$-th position, i.e.

$$y_i = \begin{cases} -1 & \text{if } i = l+1 \\ 0 & \text{otherwise} \end{cases}$$

Through the decryption oracle, we have $x =< \mathbf{y}, \mathbf{v} >= -v_{l+1} = t_1 \in \mathbb{Z}_q$ and $\mu = \lfloor t_1/2^{l-2} \rceil$. There are two cases.

1. $\mu = 0$. In this case, we have $0 \le \frac{t_1}{2^{l-2}} < \frac{1}{2}$ i.e. $t_1 < 2^{l-3} = \sum_{j=0}^{l-4} 2^j + 1$. Then it must be $\boxed{t_{1,l-1} = t_{1,l-2} = t_{1,l-3} = 0}$.
2. $1 \le \mu \le 4$. In particular, $2^{l-3} \le t_1 \le 2^l - 2$. Then we have

$$(t_{1,l-1}, t_{1,l-2}, t_{1,l-3}) \in \{0,1\}^3 \backslash \{(0,0,0)\} \tag{5}$$

Next, query the decryption oracle with $\mathbf{y} = (0, \dots, 0, -1, 0, 0, -1, 0, \dots, 0) \in \mathbb{Z}_q^N$ with -1 in $(l-2)$-th and $(l+1)$-th positions:

$$y_i = \begin{cases} -1 & \text{if } i = l-2 \text{ or } i = l+1 \\ 0 & \text{otherwise} \end{cases}$$

Through the decryption oracle, we have $x =< \mathbf{y}, \mathbf{v} >= t_1 - 2^{l-3} \geq 0$ and $\mu = \left\lfloor \frac{t_1 - 2^{l-3}}{2^{l-2}} \right\rceil$. There are two cases:

2.1. $\mu = 0$. In this case, we have $0 \leq \frac{t_1 - 2^{l-3}}{2^{l-2}} < \frac{1}{2}$ i.e. $2^{l-3} \leq t_1 < 2^{l-2} = \sum_{j=0}^{l-3} 2^j + 1$. Then it must be $\boxed{t_{1,l-1} = t_{1,l-2} = 0}$. Condition (5) implies that $\boxed{t_{1,l-3} = 1}$.

2.2. $1 \leq \mu \leq 3$. In particular, $2^{l-2} \leq t_1 \leq 2^l - 2$. Then we have

$$(t_{1,l-1}, t_{1,l-2}) \in \{0,1\}^2 \backslash \{(0,0)\} \tag{6}$$

Next, query the decryption oracle with $\mathbf{y} = (0, \ldots, 0, -1, 0, -1, 0, \ldots, 0) \in \mathbb{Z}_q^N$, with -1 in $(l-1)$-th and $(l+1)$-th positions:

$$y_i = \begin{cases} -1 & \text{if } i = l-1 \text{ or } i = l+1 \\ 0 & \text{otherwise} \end{cases}$$

Through the decryption oracle, we have $x =< \mathbf{y}, \mathbf{v} >= t_1 - 2^{l-2} \geq 0$ and $\mu = \left\lfloor \frac{t_1 - 2^{l-2}}{2^{l-2}} \right\rceil$. There are two cases:

2.2.1. $\mu = 0$. In this case, we have $0 \leq \frac{t_1 - 2^{l-2}}{2^{l-2}} < \frac{1}{2}$ and $2^{l-2} \leq t_1 < 2^{l-2} + 2^{l-3} < 2^{l-1}$. This means that $\boxed{t_{1,l-1} = 0}$. Therefore, condition (6) implies that $\boxed{t_{1,l-2} = 1}$. Moreover, since we have $0 \leq t_1 - 2^{l-2} < 2^{l-3}$, we have that $\boxed{t_{1,l-3} = 0}$.

2.2.2. $1 \leq \mu \leq 2$. In particular, $2^{l-3} + 2^{l-2} \leq t_1$.
Next, query the decryption oracle with $\mathbf{y} = (0, \ldots, 0, -1, -1, 0, -1, 0, \ldots, 0) \in \mathbb{Z}_q^N$, with -1 in $(l-2)$-th, $(l-1)$-th and $(l+1)$-th positions:

$$y_i = \begin{cases} -1 & \text{if } i = l-2, i = l-1 \text{ or } i = l+1 \\ 0 & \text{otherwise} \end{cases}$$

Through the decryption oracle, we have $x =< \mathbf{y}, \mathbf{v} >= t_1 - (2^{l-3} + 2^{l-2}) \geq 0$ and $\mu = \left\lfloor \frac{t_1 - (2^{l-3} + 2^{l-2})}{2^{l-2}} \right\rceil$. There are two cases:

2.2.2.1. $\mu = 0$. In this case, we have $0 \leq \frac{t_1 - 2^{l-3} - 2^{l-2}}{2^{l-2}} < \frac{1}{2}$, i.e. $2^{l-3} + 2^{l-2} \leq t_1 < 2^{l-1}$. This implies $\boxed{t_{1,l-1} = 0}$. Therefore, condition (6) gives $\boxed{t_{1,l-2} = 1}$. Moreover, we have $2^{l-3} \leq t_1 - 2^{l-2} < 2^{l-2}$; hence $\boxed{t_{1,l-3} = 1}$.

2.2.2.2. $\mu = 1$. We have $2^{l-1} \leq t_1 \leq 2^l - 2$. This implies $\boxed{t_{1,l-1} = 1}$. We now have to recover $t_{1,l-2}, t_{1,l-3}$.
Next, query the decryption oracle with $\mathbf{y} = (0, \ldots, 0, -1, -1, 0, \ldots, 0) \in \mathbb{Z}_q^N$, with -1 in l-th and $(l+1)$-th positions:

$$y_i = \begin{cases} -1 & \text{if } i = l \text{ or } i = l+1 \\ 0 & \text{otherwise} \end{cases}$$

Through the decryption oracle, we have $x =< \mathbf{y}, \mathbf{v} >= t_1 - 2^{l-1} \geq 0$ and $\mu = \left\lfloor \frac{t_1 - 2^{l-1}}{2^{l-2}} \right\rceil$. There are two cases:

2.2.2.2.1. $\mu = 0$. In this case, we have $0 \leq \frac{t_1 - 2^{l-1}}{2^{l-2}} < \frac{1}{2}$, i.e. $0 \leq t_1 - 2^{l-1} < 2^{l-3} = \sum_{j=0}^{l-4} 2^j + 1$. This implies $\boxed{t_{1,l-2} = t_{1,l-3} = 0}$.

2.2.2.2.2. $1 \leq \mu \leq 3$. In particular, $2^{l-3} \leq t_1 - 2^{l-1}$. Then we have

$$(t_{1,l-2}, t_{1,l-3}) \in \{0,1\}^2 \backslash \{(0,0)\} \tag{7}$$

Next, query the decryption oracle with $\mathbf{y} = (0, \ldots, 0, -1, 0, -1, -1, 0, \ldots, 0) \in \mathbb{Z}_q^N$, with -1 in $(l-2)$-th, l-th and $(l+1)$-th positions:

$$y_i = \begin{cases} -1 & \text{if } i = l-2, i = l \text{ or } i = l+1 \\ 0 & \text{otherwise} \end{cases}$$

Through the decryption oracle, we have $x =< \mathbf{y}, \mathbf{v} >= t_1 - (2^{l-1} + 2^{l-3}) \geq 0$ and $\mu = \left\lfloor \frac{t_1 - 2^{l-1} - 2^{l-3}}{2^{l-2}} \right\rceil$. There are two cases:

2.2.2.2.2.1. $\mu = 0$. In this case, we have $0 \leq \frac{t_1 - 2^{l-1} - 2^{l-3}}{2^{l-2}} < \frac{1}{2}$, i.e. $2^{l-3} \leq t_1 - 2^{l-1} < 2^{l-2}$. This means that $\boxed{t_{1,l-2} = 0}$. Condition (7) then implies $\boxed{t_{1,l-3} = 1}$.

2.2.2.2.2.2. $1 \leq \mu \leq 2$. In particular, $2^{l-2} \leq t_1 - 2^{l-1} \leq 2^l - 2 - 2^{l-1} = 2^{l-1} - 2$. Then, we have $\boxed{t_{1,l-2} = 1}$. We still have to find $t_{1,l-3}$. Next, query the decryption oracle with $\mathbf{y} = (0, \ldots, 0, -1, -1, -1, 0, \ldots, 0) \in \mathbb{Z}_q^N$, where -1 is in $(l-1)$-th, l-th and $(l+1)$-th positions:

$$y_i = \begin{cases} -1 & \text{if } i = l-1, i = l \text{ or } i = l+1 \\ 0 & \text{otherwise} \end{cases}$$

Through the decryption oracle, we have $x =< \mathbf{y}, \mathbf{v} >= t_1 - (2^{l-1} + 2^{l-2}) \geq 0$ and $\mu = \left\lfloor \frac{t_1 - 2^{l-1} - 2^{l-2}}{2^{l-2}} \right\rceil$. There are two cases:

2.2.2.2.2.2.1. $\mu = 0$. In this case,

$$0 \leq \frac{t_1 - 2^{l-1} - 2^{l-2}}{2^{l-2}} < \frac{1}{2}, \quad \text{i.e.} \quad 0 \leq t_1 - 2^{l-1} - 2^{l-2} < 2^{l-3}$$

This implies that $\boxed{t_{1,l-3} = 0}$.

2.2.2.2.2.2.2. $\mu = 1$. Then $2^{l-3} \leq t_1 - 2^{l-1} - 2^{l-2}$. This implies that $\boxed{t_{1,l-3} = 1}$.

At this point, we know the first three significant bits $t_{1,l-1}, t_{1,l-2}, t_{1,l-3}$ of t_1. Notice that we have recovered the first three most significant bits with at most 7 oracle queries. Next, we are going to recover $t_{1,l-4}$. Query the decryption oracle with

$$\mathbf{y} = (0, \ldots, 0, -t_{1,l-3}, -t_{1,l-2}, -t_{1,l-1}, -1, 0, \ldots, 0) \in \mathbb{Z}_q^N$$

where $-t_{1,i}$ is in $(i+1)$-th position. Then

$$x =< \mathbf{y}, \mathbf{v} >= t_1 - (t_{1,l-1}2^{l-1} + t_{1,l-2}2^{l-2} + t_{1,l-3}2^{l-3})$$

Now, we have $0 \leq x < 2^{l-3}$. Therefore, $\mu = \lfloor x/2^{l-2} \rceil = 0$, and so not useful at all to learn $t_{1,l-4}$. The idea is to 'shift' the bits 'to the left', i.e. towards the most significant. So, let us instead choose

$$\mathbf{y} = 2 \cdot (0, \ldots, 0, -t_{1,l-3}, -t_{1,l-2}, -t_{1,l-1}, -1, 0, \ldots, 0) \in \mathbb{Z}_q^N$$

So now $x =< \mathbf{y}, \mathbf{v} >$ is such that $0 \leq x < 2^{l-2}$. After submitting $\mathbf{y}$ to the decryption oracle, it will compute and output $\mu = \lfloor x/2^{l-2} \rceil$. Then $\boxed{t_{1,l-4} = \mu}$.

Now we can generalize and recover $t_{1,k}$, for all $k = l-4, l-5, \ldots, 1, 0$. This will complete the recovery of t_1. Suppose that, for a given k, we recovered already $t_{1,m}$, $\forall m \in [k+1, \ldots, l-1]$. We then recover $t_{1,k}$ by recurrence. Choose

$$\mathbf{y} = 2^{l-k-3}(0, \ldots, 0, -t_{1,k+1}, -t_{1,k+2}, \ldots, -t_{1,l-1}, -1, 0, \ldots, 0) \in \mathbb{Z}_q^N$$

with $-t_{1,i}$ in $(i+1)$-th position; i.e.

$$y_i = \begin{cases} -2^{l-k-3} t_{1,i-1} & \text{for } i \in [k+2, \ldots, l] \\ -2^{l-k-3} & \text{for } i = l+1 \\ 0 & \text{otherwise} \end{cases}$$

Then we have $x =< \mathbf{y}, \mathbf{v} >= 2^{l-k-3} \left(t_1 - \sum_{j=k+1}^{l-1} t_{1,j} 2^j \right)$ with $0 \leq x < 2^{l-2}$. Then, $\boxed{t_{1,k} = \mu}$.

We recover completely t_1 after at most $7 + (l-3) = l+4$ oracle queries.

Recovering $\mathsf{BitDecomp}(\mathsf{t_r})$, for every $r \in [1, 2, \ldots, n]$

We can now generalize and recover $\mathsf{BitDecomp}(\mathsf{t_r})$, for every $r \in [1, 2, \ldots, n]$, in a way analogous to what has been done for the case $r = 1$. The only difference is that, when choosing $\mathbf{y} \in \mathbb{Z}_q^N$, we set -1 in position $rl + 1$. So, for a given $r \in [1, 2, \ldots, n]$, we have the following.

- Recovering the first three most significant bits $t_{r,l-1}, t_{r,l-2}, t_{r,l-3}$. This is done exactly as in the case of t_1, with the only modification $y_{l+1} = 0$ and $y_{rl+1} = -1$ always.
- Recovering $t_{r,k}$, for all $k = l-4, l-5, \ldots, 1, 0$. Suppose that, for a given k, we recovered already $t_{r,m}$, $\forall m \in [k+1, \ldots, l-1]$. We then recover $t_{r,k}$ by recurrence. Choose

$$\mathbf{y} = 2^{l-k-3}(0, \ldots, 0, -t_{r,k+1}, -t_{r,k+2}, \ldots, -t_{r,l-1}, 0, \ldots, 0, -1, 0, \ldots, 0) \in \mathbb{Z}_q^N$$

with $-t_{r,i}$ in $(i+1)$-th position and -1 in $(rl+1)$-th position; i.e.

$$y_i = \begin{cases} -2^{l-k-3} t_{r,i-1} & \text{for } i \in [k+2, \ldots, l] \\ -2^{l-k-3} & \text{for } i = rl+1 \\ 0 & \text{otherwise} \end{cases}$$

Then we have $x =< \mathbf{y}, \mathbf{v} >= 2^{l-k-3} \left(t_r - \sum_{j=k+1}^{l-1} t_{r,j} 2^j \right)$ with $0 \leq x < 2^{l-2}$. Then, $\boxed{t_{r,k} = \mu}$.

In summary, we can recover the secret key $\mathbf{t} \in \mathbb{Z}_q^n$ with at most $(l+4) \cdot n$ oracle queries.

7.3 Algorithmic Description

Formally, the attack from Section 7.2 can be described by Algorithm 6. For a given vector $\mathbf{y} = (y_1, \ldots, y_N) \in \mathbb{Z}_q^N$, we let $C_{\mathbf{y}} \in \mathcal{M}_{N \times N}(\mathbb{Z}_q)$ be the square $N \times N$ matrix whose entries are all 0 except for the $(l-2)$-th row C_{l-2}, which is $\mathbf{y}$. We have denoted the decryption oracle $\mathcal{O}_D(C_{\mathbf{y}}) := \mathsf{Decrypt}(\mathsf{sk}, C_{\mathbf{y}}) = \left\lfloor \frac{<\mathbf{y},\mathbf{v}>}{2^{l-2}} \right\rceil$. For ease of notation, we have also considered the standard vectors $\mathbf{e}_1, \ldots, \mathbf{e}_N \in \mathbb{Z}_q^N$: for every $i = 1, \ldots, N$, $\mathbf{e}_i$ is the $\mathbf{0}$-vector except in position i, where it has value 1:

$$\mathbf{e}_i = (e_{i,1}, \ldots, e_{i,N}) = (0, \ldots, 0, 1, 0, \ldots, 0), \quad e_{i,i} = 1, e_{i,j} = 0 \text{ for } j \neq i$$

We put $\mathbf{d}_i := -\mathbf{e}_i$, for all i.

8 Conclusion

In this paper, we showed that the SHE schemes from [BV11b,BV11a,BGV12,Bra12,GSW13] suffer from key recovery attacks when the attacker is given access to the decryption oracle. Combining the results from [LMSV12,ZPS12], we now know that most existing SHE schemes suffer from key recovery attacks, and so they are not IND-CCA1 secure. As such, a natural next step is to investigate whether it is possible to enhance these SHE schemes to avoid key recovery attacks and make them IND-CCA1 secure. One thing we should keep in mind is to preserve their homomorphic properties. Following the work of [LMSV12], one could think of tweaking the decryption step of a SHE scheme by including a ciphertext validity check in order to make sure that, with some high probability, the ciphertext is honestly generated by the attacker and not specifically chosen for the purpose of recovering a given bit (or bits) of the secret key. Unfortunately, we cannot directly apply the techniques from [LMSV12] due to the fact that the SHE scheme from [LMSV12] enjoys some particular algebraic properties which do not exist in other schemes. So, we need to treat each SHE scheme individually.

Acknowledgements

Massimo Chenal is supported by an AFR PhD grant from the National Research Fund, Luxembourg. Qiang Tang is partially supported by a CORE (junior track) grant from the National Research Fund, Luxembourg.

References

[BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, ITCS '12, pages 309–325. ACM, 2012.

[Bra12] Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical gapsvp. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology - CRYPTO 2012*, volume 7417 of *LNCS*, pages 868–886. 2012.

[BV11a] Zvika Brakerski and Vinod Vaikuntanathan. Fully homomorphic encryption from ring-lwe and security for key dependent messages. In *Advances in Cryptology - CRYPTO 2011*, pages 505–524, 2011.

Algorithm 6 Key Recovery Attack against GSW13 SHE

```
input: $q, n$
$l \leftarrow \lfloor \log_2 q \rfloor + 1$
$N \leftarrow (n+1) \cdot l$
for $r = 1$ to $n$ do
  $\mathbf{y} \leftarrow \mathbf{d}_{rl+1}$
  if $\mathcal{O}_D(C_{\mathbf{y}}) = 0$ then
    $t_{r,l-1}, t_{r,l-2}, t_{r,l-3} \leftarrow 0$
  else
    $\mathbf{y} \leftarrow \mathbf{d}_{l-2} + \mathbf{d}_{rl+1}$
    if $\mathcal{O}_D(C_{\mathbf{y}}) = 0$ then
      $t_{r,l-1}, t_{r,l-2} \leftarrow 0$
      $t_{r,l-3} \leftarrow 1$
    else
      $\mathbf{y} \leftarrow \mathbf{d}_{l-1} + \mathbf{d}_{rl+1}$
      if $\mathcal{O}_D(C_{\mathbf{y}}) = 0$ then
        $t_{r,l-1}, t_{r,l-3} \leftarrow 0$
        $t_{r,l-2} \leftarrow 1$
      else
        $\mathbf{y} \leftarrow \mathbf{d}_{l-2} + \mathbf{d}_{l-1} + \mathbf{d}_{rl+1}$
        if $\mathcal{O}_D(C_{\mathbf{y}}) = 0$ then
          $t_{r,l-1} \leftarrow 0$
          $t_{r,l-2}, t_{r,l-3} \leftarrow 1$
        else
          $t_{r,l-1} \leftarrow 1$
          $\mathbf{y} \leftarrow \mathbf{d}_{l} + \mathbf{d}_{rl+1}$
          if $\mathcal{O}_D(C_{\mathbf{y}}) = 0$ then
            $t_{r,l-2}, t_{r,l-3} \leftarrow 0$
          else
            $\mathbf{y} \leftarrow \mathbf{d}_{l-2} + \mathbf{d}_{l} + \mathbf{d}_{rl+1}$
            if $\mathcal{O}_D(C_{\mathbf{y}}) = 0$ then
              $t_{r,l-2} \leftarrow 0$
              $t_{r,l-3} \leftarrow 1$
            else
              $t_{r,l-2} \leftarrow 1$
              $\mathbf{y} \leftarrow \mathbf{d}_{l-1} + \mathbf{d}_{l} + \mathbf{d}_{rl+1}$
              if $\mathcal{O}_D(C_{\mathbf{y}}) = 0$ then
                $t_{r,l-3} \leftarrow 0$
              else
                $t_{r,l-3} \leftarrow 1$
              end if
            end if
          end if
        end if
      end if
    end if
  end if
  for $k = l-4$ to $0$ do
    $\mathbf{y} \leftarrow 2^{l-k-3} \cdot (\mathbf{d}_{rl+1} + \sum_{i=k+2}^{l} t_{r,i-1}\mathbf{d}_i)$
    $t_{r,k} \leftarrow \mathcal{O}_D(C_{\mathbf{y}})$
  end for
end for
for $i = 1$ to $n$ do
  $t_i \leftarrow \mathsf{BitDecomp}^{-1}(t_{i,0}, t_{i,1}, \ldots, t_{i,l-1})$
end for
$\mathbf{t} \leftarrow (t_1, \ldots, t_n)$
return $\mathbf{t}$
```

[BV11b] Zvika Brakerski and Vinod Vaikuntanathan. efficient fully homomorphic encryption from (standard) lwe. In *Proceedings of the 2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, FOCS '11, pages 97–106, 2011.

[CCK$^+$13] JungHee Cheon, Jean-Sbastien Coron, Jinsu Kim, MoonSung Lee, Tancrde Lepoint, Mehdi Tibouchi, and Aaram Yun. Batch fully homomorphic encryption over the integers. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology - EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 315–335. 2013.

[CMNT11] Jean-Sébastien Coron, Avradip Mandal, David Naccache, and Mehdi Tibouchi. Fully homomorphic encryption over the integers with shorter public keys. In *Advances in Cryptology - CRYPTO 2011*, pages 487–504, 2011.

[CNT12] Jean-Sébastien Coron, David Naccache, and Mehdi Tibouchi. Public key compression and modulus switching for fully homomorphic encryption over the integers. In *Advances in Cryptology - EUROCRYPT 2012*, pages 446–464, 2012.

[DT14] Jintai Ding and Chengdong Tao. A new algorithm for solving the approximate common divisor problem and cryptanalysis of the fhe based on gacd. IACR Cryptology ePrint Archive, Report 2014/042, 2014.

[Gen09a] Craig Gentry. *A Fully Homomorphic Encryption Scheme*. PhD thesis, Stanford, CA, USA, 2009.

[Gen09b] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the Forty-first Annual ACM Symposium on Theory of Computing*, STOC '09, pages 169–178. ACM, 2009.

[Gen10] Craig Gentry. Computing arbitrary functions of encrypted data. *Commun. ACM*, 53(3):97–105, March 2010.

[GH11a] Craig Gentry and Shai Halevi. Fully homomorphic encryption without squashing using depth-3 arithmetic circuits. In *Proceedings of the 2011 IEEE 52Nd Annual Symposium on Foundations of Computer Science*, FOCS '11, pages 107–109, 2011.

[GH11b] Craig Gentry and Shai Halevi. Implementing gentry's fully-homomorphic encryption scheme. In *Advances in Cryptology - EUROCRYPT 2011*, pages 129–148, 2011.

[GHS12a] Craig Gentry, Shai Halevi, and Nigel P. Smart. Better bootstrapping in fully homomorphic encryption. In *Proceedings of the 15th International Conference on Practice and Theory in Public Key Cryptography*, PKC'12, pages 1–16, 2012.

[GHS12b] Craig Gentry, Shai Halevi, and Nigel P. Smart. Fully homomorphic encryption with polylog overhead. In *Advances in Cryptology - EUROCRYPT 2012*, pages 465–482, 2012.

[GSW13] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In Ran Canetti and JuanA. Garay, editors, *Advances in Cryptology - CRYPTO 2013*, volume 8042 of *LNCS*, pages 75–92. 2013.

[HPS98] Jeffrey Hoffstein, Jill Pipher, and JosephH. 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.

[LMSV12] Jake Loftus, Alexander May, Nigel P. Smart, and Frederik Vercauteren. On cca-secure somewhat homomorphic encryption. In *Proceedings of the 18th International Conference on Selected Areas in Cryptography*, SAC'11, pages 55–72, 2012.

[MP11] Daniele Micciancio and Chris Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. IACR Cryptology ePrint Archive, Report 2011/501, 2011.

[NLV11] Michael Naehrig, Kristin Lauter, and Vinod Vaikuntanathan. Can homomorphic encryption be practical? In *Proceedings of the 3rd ACM Workshop on Cloud Computing Security Workshop*, CCSW '11, pages 113–124, 2011.

[Nui14] Koji Nuida. A simple framework for noise-free construction of fully homomorphic encryption from a special class of non-commutative groups. IACR Cryptology ePrint Archive, Report 2014/097, 2014.

[RAD78] Ronald L. Rivest, Len Adleman, and Michael L. Dertouzos. On data banks and privacy homomorphisms. *Foundations of Secure Computation, Academia Press*, pages 169–179, 1978.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In *Proceedings of the Thirty-seventh Annual ACM Symposium on Theory of Computing*, STOC '05, pages 84–93, 2005.

[SS10] Damien Stehle and Ron Steinfeld. Faster fully homomorphic encryption. In Masayuki Abe, editor, *Advances in Cryptology - ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 377–394. 2010.

[SV10] N. P. Smart and F. Vercauteren. Fully homomorphic encryption with relatively small key and ciphertext sizes. In *Proceedings of the 13th International Conference on Practice and Theory in Public Key Cryptography*, PKC'10, pages 420–443, 2010.

[vDGHV10] Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully homomorphic encryption over the integers. In *Advances in Cryptology - EUROCRYPT 2010*, pages 24–43, 2010.

[ZPS12] Zhenfei Zhang, Thomas Plantard, and Willy Susilo. On the cca-1 security of somewhat homomorphic encryption over the integers. In *Proceedings of the 8th International Conference on Information Security Practice and Experience*, ISPEC'12, pages 353–368, 2012.