Reaction Attack on Outsourced Computing with Fully Homomorphic Encryption Schemes[⋆]

Zhenfei Zhang, Thomas Plantard, and Willy Susilo

Centre for Computer and Information Security Research
School of Computer Science & Software Engineering (SCSSE)
University Of Wollongong, Australia
{zz920, thomaspl, wsusilo}@uow.edu.au

Abstract. Outsourced computations enable more efficient solutions towards practical problems that require major computations. Nevertheless, users' privacy remains as a major challenge, as the service provider can access users' data freely. It has been shown that fully homomorphic encryption schemes might be the perfect solution, as it allows one party to process users' data homomorphically, without the necessity of knowing the corresponding secret keys. In this paper, we show a reaction attack against full homomorphic schemes, when they are used for securing outsourced computation. Essentially, our attack is based on the users' reaction towards the output generated by the cloud. Our attack enables us to retrieve the associated secret key of the system. This secret key attack takes $O(\lambda \log \lambda)$ time for both Gentry's original scheme and the fully homomorphic encryption scheme over integers, and $O(\lambda)$ for the implementation of Gentry's fully homomorphic encryption scheme.

Keywords: Cloud Computing, Fully Homomorphic Encryption, Reaction Attack, CCA security, Secured Outsource Computation

1 Introduction

Cloud computing has changed the phenomena in the Information Technology (IT) industry completely. It allows access to highly scalable, inexpensive, on-demand computing resources that can execute the code and store the data that are provided to them. This aspect, known as data outsourced computation, is very attractive, as it alleviates most of the burden on IT services from the consumer (or data owner). Nevertheless, the adoption of data outsourced computation by business has a major obstacle, since the data owner does not want to allow the untrusted cloud provider to have access to the data being outsourced. Merely encrypting the data prior to storing it on the cloud is not a viable solution, since encrypted data cannot be further manipulated. This means that if the data owner would like to search for particular information, then the data

⋆ This work is supported by ARC Future Fellowship FT0991397.

would need to be retrieved and decrypted - a very costly operation, which limits the usability of the cloud to merely be used as a data storage centre.

In [1], van Dijk and Juels argued that, cryptography tools alone is not sufficient for providing privacy for cloud computing. Yet, they found that in a single-client scenario, fully homomorphic encryption schemes deliver the required security.

Indeed, a fully homomorphic encryption is a solution for enabling operations on the encrypted data. Essentially, fully homomorphic encryption schemes enable one to apply homomorphic operations over an arbitrary number (n) of given ciphertexts $c_1, c_2, \ldots, c_n$ without the need to know the corresponding plaintexts $m_1, m_2, \ldots, m_n$.

This feature is useful in the outsourced computation scenario, where one can upload encrypted data to the cloud and enable the cloud to process the data *without* the need for decryption. Nevertheless, whether a fully homomorphic encryption by itself is sufficient enough to secure the outsourced computation in practice remains unclear. This will require some further research.

Our Contribution
In this paper, we show a negative result to the above question. Before addressing our contribution, we should highlight some important factors to motivate our work. In fact, one can categorize an outsourced computation into the following models:

1. The user possesses the data and the computation circuit, and the service provider provides the computation power;
   Example: a stock share holder buys/sells his/her stocks via the cloud, and then retrieve the receipt from the cloud to obtain his/her updated financial status.
2. The user possesses the data, and the service provider provides its computational power, while the computation circuit can be made available publicly to both of the entities;
   Example: a hospital outsources its patients' information to a research institute for acquiring further analysis from the institute (such as the result of the prostate cancer), as the institute has more computational power compared to the hospital.
3. The user possesses the data, and the service provider provides its computational power, only the cloud has access to the computation circuit;
   Example: a company outsources its financial status to an auditing company, however, the auditing algorithm is auditing the company's private property.

In all of the above models, the users' data privacy has to be ensured. The difference among them lies on the privacy of the computational circuit.

In this paper, we present a practical reaction attack that can be applied to all of the above models, in which every time a user interacts with the cloud, he/she is under the risk of leaking some information. As a result, known approaches, i.e., secured outsourced computation (SOC) [2], becomes essential in the first two models, while for the third model, even SOC technique will not be sufficient
.

Further, using our attack, one can construct a probabilistic decryption oracle. Consequently, we argue that for any fully homomorphic encryption schemes, the CCA-1 security is essential. When applying our attack to Gentry's framework [3, 4], our attack recovers the secret key for *all* published fully homomorphic encryption schemes. This secret key attack take $O(\lambda \log \lambda)$ time for both Gentry's original scheme and the fully homomorphic encryption scheme over integers. Furthermore, for the implementation of Gentry's fully homomorphic encryption scheme, this secret key attack requires $O(\lambda)$ time.

Related Work
In [5], Hall et al. presented a reaction attack against several public key cryptosystems, mainly on lattice based cryptosystems and coding based cryptosystems. By observing the reaction of the decryption procedure, one obtains information about the secret key and/or message. Compared to the result of [5], where one does not observe the reaction of the decryption procedure, our attack relies on the difference of users' reactions on receiving different results.

In [2], Gennaro et al. presented a non-interactive verifiable computing protocol, that allows one to outsource its computation to untrusted workers. This method can be applied to the first two models, where the users can check if the cloud has modified the demanded computation circuit. However, it is impossible for this method to be applied to the third model. In addition, the main obstacle of using this technique is its computation efficiency. Initially, this protocol requires a one time pre-processing to generate a minimum garbled circuit, which takes $O(C) \times \mathit{poly}(\lambda)$ time, where C is in function of the computation circuit, and λ is the security parameter. Then, for each new computation circuit, one is required to modify the minimum garbled circuit, using a fully homomorphic encryption scheme. Thus, we argue that this method is impractical.

Other related work of this paper can be found in [6].

2 Background

2.1 Fully Homomorphic Encryptions

The idea of fully homomorphic encryption was raised by Rivest, Adleman and Dertouzos [7], shortly after the invention of RSA [8]. A fully homomorphic encryption scheme consists of following four algorithms:

- $\textsc{KeyGen}(\lambda)$: Input a security parameter λ, it outputs public key $\mathbf{pk}$, secret key $\mathbf{sk}$.
- $\textsc{Encrypt}(m, \mathbf{pk})$: Input a message m and the public key $\mathbf{pk}$, it outputs a corresponding ciphertext c.
- $\textsc{Decrypt}(c, \mathbf{sk})$: Input a ciphertext c and the secret key $\mathbf{sk}$, it outputs a corresponding message m.
- $\textsc{Eval}(\mathbf{pk}, c_1, c_2, \dots, c_n, \mathcal{C}^n)$: Input a public key $\mathbf{pk}$, n ciphertext $c_1, c_2, \dots, c_n$ and a permitted circuit $\mathcal{C}^n$, it outputs $\mathcal{C}^n(c_1, c_2, \dots, c_n)$.

Following this notion, schemes that support partial homomorphism have been proposed. Recently, Gentry [3, 4] successfully provided a framework for constructing homomorphic encryption schemes (referred to as the Gentry scheme) and, further, he provided a concrete construction. In addition, subsequent works based on his framework have been proposed recently (such as [9–11]). For instance, in [12] (referred to as Gentry-Halevi scheme), the author optimized the performance of Gentry scheme, while in [11] (referred to as vDGHV scheme), the author proposed an integer variant of Gentry scheme. In the following, for clarity, we will review Gentry's framework.

2.2 Gentry's Framework

Gentry's framework for constructing fully homomorphic encryption schemes is based on creating a function to perform two atomic operations which will allow the user to build any kind of circuit. Effectively, any circuit can be built with two atomic functions, namely addition $+$ and multiplication $\times$ over $\mathbb{F}_2$ (see [3, 4]). Therefore, to evaluate any circuit, we are only required to be able to add and multiply over $\mathbb{F}_2$ two encrypted bits.

We note that, to ensure security, such an encryption function is required to be indistinguishable, namely $Enc(m_0) \neq Enc(m_1) \nRightarrow m_0 \neq m_1$. To build such a function, $\oplus$ and $\otimes$, Gentry used a simple model. Gentry defined the two functions f_+ and $f_\times$ which are equivalent to decrypting both encrypted bits, adding or multiplying such decrypted bits and then encrypting the resulting bits (See Figure 1).

However, if f_+ and $f_\times$ return the desired result for $\oplus$ and $\otimes$, the bits are clearly readable and therefore they do not maintain the intended security requirement.

To achieve this required property, Gentry used an encryption scheme which allows evaluation of short circuits. Therefore, it encrypts the ciphertext with a second cryptosystem. Hence, it can remove the first encryption securely to perform the addition or the multiplication (See Figure 2).
Using such a technique, Gentry simplified the quest of constructing a fully homomorphic encryption that can evaluate any circuit on encrypted data by finding an encryption system that can evaluate only some short circuits, namely f_+ and $f_\times$. In [4, 3], Gentry built such an encryption scheme using ideal lattices. This work was followed by other fully homomorphic encryption schemes based also on ideal lattices [10, 9]. Another type of encryption scheme respecting Gentry's model requirement was also proposed in [11] using integers.

2.3 The vDGHV Fully Homomorphic Encryption Scheme

In this subsection, we describe the fully homomorphic encryption scheme over integers (vDGHV scheme), instead of Gentry scheme, since this scheme uses integers rather than ideal lattice, and therefore, it is easier to demonstrate and explain, and later incorporate our idea into.

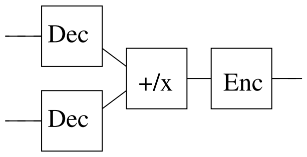

Fig. 1: $f_{+,\times}$

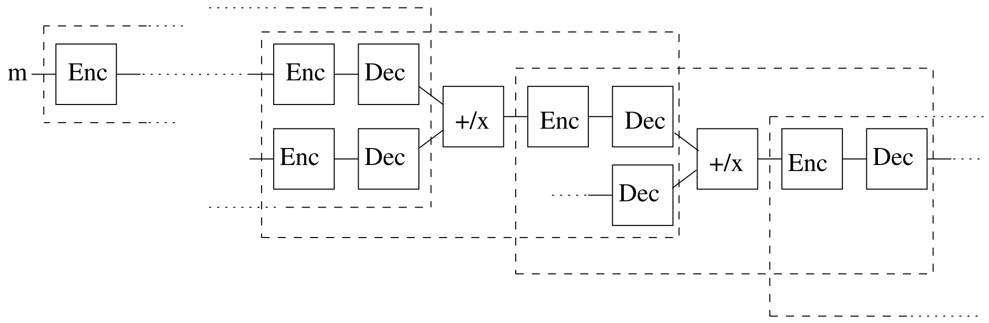

Fig. 2: Gentry's fully homomorphic encryption model

vDGHV scheme consists of a somewhat homomorphic encryption scheme (SHE) that supports limited additions and multiplications, and the bootstrapping technique to break such limitation.

The somewhat homomorphic encryption scheme consists of four algorithms:

- KeyGen(λ): Input a security parameter λ, it firstly generates parameters $\{\alpha, \beta, \gamma, t, n\}$ in function of λ. It then generates a secret *odd* integer $p \in (2^{\beta}, 2^{\beta+1})$, n different integers $\{r_i \in [-2^{\alpha}, 2^{\alpha})\}$ and another n different integers $\{g_i \in [0, 2^{\gamma-\beta})\}$, respectively. It finally outputs the public key $\mathbf{pk} = \{x_i = g_i p + 2r_i\}$ and secret key $\mathbf{sk} = \{p\}$.
- Encrypt(m, $\mathbf{pk}$): Input the public key $\mathbf{pk}$ and a message $m \in \{0, 1\}$, it chooses a random subset $\mathbf{s} \subseteq \mathbf{pk}$ and output $c = m + 2r + \sum_{x_i \in \mathbf{s}} x_i \bmod x_0$, where x_0 is the smallest in $\{x_i\}$, $r \in [-2^{\alpha}, 2^{\alpha})$ is a random noise.
- Decrypt(c, $\mathbf{sk}$): Input the secret key $\mathbf{sk} = \{p\}$ and a ciphertext c, it outputs $m = (c \bmod 2) \oplus (\lfloor c/p \rceil \bmod 2)$, where $\lfloor c/p \rceil$ returns the closest integer of c/p.
- Evaluate(c_1, c_2, ..., c_k, $\mathcal{P}$, $\mathbf{pk}$). It outputs $\mathcal{P}(c_1, c_2, ..., c_k)$, where $\mathcal{P}$ is a k-inputs evaluation polynomial whose circuit depth is lower than the maximum circuit depth allowed by this SHE.

This SHE supports homomorphic additions and multiplications, when $\alpha \ll \beta$. For instance, suppose $c_1 = m_1 + g_1 p + 2r_1$ and $c_2 = m_2 + g_2 p + 2r_2$ for certain g_1, r_1, g_2, r_2, the product of two ciphertext $c_1 c_2 = m_1 m_2 + 2(r_1 m_2 + r_2 m_1 +$

$2r_1r_2) + p(g_1m_2 + 2g_1r_2 + g_2m_1 + 2g_2r_1 + g_1g_2p)$. One can observe that the decryption of c_1c_2 is m_1m_2, as long as $2(r_1m_2 + r_2m_1 + 2r_1r_2) \in (-p/2, p/2]$. Therefore, the above SHE is somewhat homomorphic.

However, the homomorphic circuit depth is limited, *i.e.*, the noise grows after each operation, and eventually it is possible that the absolute value of the noise will be greater than $p/2$ and a decryption error is then being generated.

Suppose we want to evaluate a circuit whose depth is greater than this SHE permits, we break the circuit into several sub-circuits. For each sub-circuit, the absolute value of resulted noise is less than the threshold ($p/2$). Then we refresh the resulted ciphertext using the bootstrapping technique. We describe the bootstrapping technique in general. We refer the readers to their original scheme[11] for more details.

To bootstrap, firstly, they modify the decryption circuit. As we have shown earlier, the noise grows significantly faster in a multiplication than in an addition. Therefore, vDGHV scheme used a squashing method that breaks the decryption circuit from one multiplication into several additions. The squashing technique is as follows:

- Generate $x = \lfloor 2^\kappa/p \rceil$, where κ is a parameter in λ that is greater than $\beta + 1$.
- Build a bit sequence $S =< s_1, s_2, \ldots, s_\eta >$, $s_i \in \{0, 1\}$, with $\sum s_i = \theta$. S becomes the new secret key.
- Choose n random integers u_i between 0 and $2^{\kappa+1}$, such that $\sum_i^n s_iu_i = x \bmod 2^{\kappa+1}$.
- Set $y_i = u_i/2^\kappa$. Then $\sum s_iy_i = 1/p + \epsilon$, where ϵ is negligible compared with $1/p$.
- New ciphertext is a vector $z =< z_1, z_2, \ldots, z_\eta >$, generated by $z_i = [c \times y_i]_2$.
- New decryption circuit becomes $m = [c - \lfloor \sum s_i \times z_i \rceil]_2$

As a result, the decryption circuit now consists only additions, while the growth of noise in additions is extremely slow. Then, because the modified decryption circuit depth is relatively low, now it is possible to carry out the decryption circuit homomorphically, through the proposed SHE.

In practice, they encrypt ciphertexts, denoted by $\{Enc(z_i)\}$ and the secret keys, denoted by $\{Enc(s_i)\}$. Denote $\mathcal{C}_D$ the decryption circuit, then

$$\textsc{Decrypt}(\mathcal{C}_D, \{Enc(z_i)\}, \{Enc(s_i)\}) = Enc(m).$$

This is because firstly $\mathcal{C}_D(\{z_i\}, \{s_i\}) = m$ and secondly, $\mathcal{C}_D$ can be carried out homomorphically. Therefore, we obtain a new ciphertext $Enc(m)$.

The new ciphertext, $Enc(m)$ has a refreshed noise level (less than 2^α), which means $Enc(m)$ can be evaluated again. By doing this repeatedly, we can evaluate circuit with any depth homomorphically. Therefore, a fully homomorphic encryption scheme is achieved.

2.4 Security Models

In the following, we describe briefly both Chosen-Plaintext Attack (CPA) [13] and Chosen-Ciphertext Attack (CCA) [14] attacks for completeness.

The IND-CPA security game is defined as follows:

1. The challenger runs KeyGen algorithm and outputs a secret key **sk** and a public key **pk**;
2. The attacker is given an encryption oracle that computes the functionality Encrypt(m, **pk**);
3. The attacker then generates two ciphertexts m_0 and m_1;
4. The challenger generates $c = \text{Encrypt}(m_b, \mathbf{sk})$, where $b \in \{0, 1\}$;
5. The attacker outputs b'.

We say that an encryption scheme is CPA secure if the advantage of the attacker to win the game $(Pr[b = b'] - 1/2)$ is negligible.

The IND-CCA-1/2 security game is defined as follows:

1. The challenger runs KeyGen algorithm and outputs a secret key **sk** and a public key **pk**;
2. The attacker is given two oracles, an encryption oracle and a decryption oracle;
3. The attacker then generates two ciphertexts m_0 and m_1;
4. The challenger generates $c = \text{Encrypt}(m_b, \mathbf{sk})$, where $b \in \{0, 1\}$;
5. (Only for CCA-2) The attacker is given the two oracles again, but it can not query on c;
6. The attacker outputs b'.

We say that an encryption scheme is CCA-1/2 secure if the advantage of the attacker to win the game $(Pr[b = b'] - 1/2)$ is negligible.

3 Our Reaction Attack

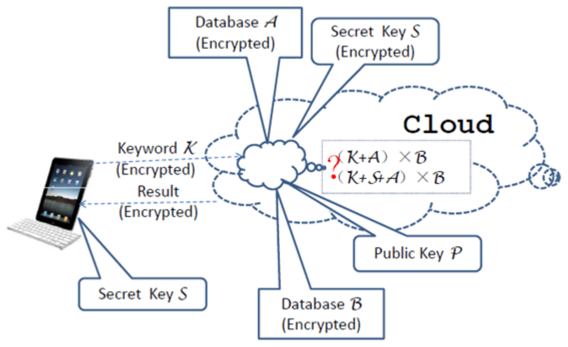
Database $\mathcal{A}$ (Encrypted)
Secret Key $\mathcal{S}$ (Encrypted)
Cloud
Keyword $\mathcal{K}$ (Encrypted)
Result (Encrypted)
?
$(\mathcal{K}+\mathcal{A}) \times \mathcal{B}$
$(\mathcal{K}+\mathcal{S}+\mathcal{A}) \times \mathcal{B}$
Public Key $\mathcal{P}$
Secret Key $\mathcal{S}$
Database $\mathcal{B}$ (Encrypted)

Fig. 3: Using Fully Homomorphic Encryption in a Cloud Search Scenario

In this section, we will firstly introduce our message attack that recovers a message of any kind of fully homomorphic systems used in the outsourced computation, in probabilistic manner. Then, we show that by adapting our attack in Gentry's framework, we can recover the secret key. We note that our attack described in this section is applicable to all three models in the first section. For simplicity, we uses the first model.

3.1 A Message Attack

The Idea In this subsection, we describe our message attack. For any given ciphertext, our attack recovers the message with provability ε.

We will first illustrate our high level construction for clarity. To use fully homomorphic encryption schemes in outsourced computation scenarios, the users firstly upload their encrypted data to the cloud. Then, they submit their demanded circuits to the cloud, in an on-demand fashion. The demanded circuit consists either some data and an evaluation function, or merely an evaluation function only. The cloud processes users' data through the requested circuits, and returns the result.

Ideally, all the data, including the results, are encrypted, and hence, a malicious cloud provider cannot gain information from the users, *i.e.*, let ε_1 be the possibility of $m = 1$, then for any ciphertext, $\mid \varepsilon_1 - 1/2 \mid$ is negligible from the cloud provider's point of view.

Nevertheless, we notice that the attacker can modify the encrypted circuits/results by adding some random ciphertext c that encrypts a message m. Because homomorphism is enabled, modifying the demanded circuits/results will affect the plaintext eventually. To be more precise, if the added random ciphertext encrypts a 0, the returned result remains the same; while if the added random ciphertext encrypts a 1, the returned result is modified. By observing the users' reactions, the service provider can increase or decrease ε_1 accordingly, and eventually recover m.

Generally speaking, the cloud provider can compare users' reaction with their former reaction, if the users are acting "unexpectedly", then the cloud can expect $m = 1$. For completeness, we list some (*but not* all) possible reactions that can be defined as "unexpected" behaviors.

- The users set up a new task much sooner than usual, after they acquire the result sent by the cloud;
- The circuit of a new task is identical from a former one;
- The number of tasks is significantly higher than average - this occurs when the cloud provider feeds same faulty information for a certain period.

We note that these users' reactions are very natural and practical. To anticipate the reactions is even easier, when the users use a certain software, instead of expecting the results themselves, to communicate with the cloud. We argue that this is very common in practice as nobody will conduct this process manually.

However, the success of our attack relies highly on the actions performed by the users after receiving valid or error results. Hence, if the users act completely randomly, then our attack will be unsuccessful. Nevertheless, we argue that the latter usually will not happen in practice, as it is the users' interest to acquire the results that they would like to obtain.

An Example In this subsection, we demonstrate an example of our attack. Suppose we have a cloud search engine (see Figure 3), which looks up keywords from database A, and outputs the corresponding results in database B. Database A consists of names of stocks, while database B shows corresponding price for each stock.

To distinguish from a traditional search engine, the databases of the cloud search engine are all encrypted, and the search circuit is a homomorphic circuit. Without losing generality, we use vDGHV scheme to demonstrate our attack.

Entry	A	B
1	AAPL	335
2	GOOG	494
3	MSFT	027
4	SPRD	013
5	NDAQ	024
...	...	...

Table 1: Databases in plaintext

Table 1 shows the databases in plaintext for our example. We note that this is not exactly the databases stored in the cloud. The cloud maintains multiple copies of the databases for different users, each copy is encrypted under different users' FHE secret key.

Let K, A and B denote the binary form of the keyword, database A and database B, respectively. Let k_i, a_i and b_i be the i-th digit of K, A and B. Then, the database in the cloud consists of $\{Enc(a_i)\}$ and $\{Enc(b_i)\}$. Also, let $\bigotimes_1^n c_i$ be $c_1 \otimes c_2 \otimes \cdots \otimes c_n$. A basic search algorithm is defined in Algorithm 1, and we achieve a fully homomorphic searching algorithm in Algorithm 2 using the fully homomorphic encryption scheme over integers.

Algorithm 1 Basic Search (K, A, B)

```
$l_a \leftarrow$ LEN_OF_WORD_A //$l_a = 32$
$l_b \leftarrow$ LEN_OF_WORD_B //$l_b = 24$
for $j = 0 \rightarrow$ END_OF_ENTRY$-1$ do
  for $i = 1 \rightarrow l_b$ do
    $r_i \leftarrow b_{i+jl_b} \otimes (\bigotimes_{t=1}^{l_a}(a_{t+jl_a} \oplus k_{t+jl_a})) \oplus r_i$
  end for
end for
```

As shown in Table 2, suppose we want to look for the price of GOOG, with the basic search algorithm, we obtain $52, 57, 52$ in ASCII code, which is 494. While with the homomorphic search algorithm, we obtain the third column of

Algorithm 2 Homomorphical Search $(\{Enc(k_i)\}, \{Enc(a_i)\}, \{Enc(b_i)\})$

```
$l_a \leftarrow$ LEN_OF_WORD_A //$l_a = 24$
$l_b \leftarrow$ LEN_OF_WORD_B //$l_b = 32$
for $j = 0 \rightarrow$END_OF_ENTRY$-1$ do
  for $i = 1 \rightarrow l_a$ do
    $r_i \leftarrow Enc(b_{i+jl_b}) \times \prod_{t=1}^{l_a}(Enc(a_{t+jl_a}) + Enc(k_{t+jl_a})) + r_i$
  end for
end for
```

Database A	Basic Search	Homomorphic Search	Homo Search + Faulty Info
AAPL	0,0,0	Enc(0), Enc(0), Enc(0)	Enc(0),Enc(0),Enc(0)
GOOG	52,57,52	Enc(52), Enc(57), Enc(52)	Enc(0),Enc(0),Enc(0)
MSFT	52,57,52	Enc(52), Enc(57), Enc(52)	Enc(0),Enc(0),Enc(0)
SPRD	52,57,52	Enc(52), Enc(57), Enc(52)	Enc(0),Enc(0),Enc(0)
NDAQ	52,57,52	Enc(52), Enc(57), Enc(52)	Enc(0),Enc(0),Enc(0)
...	...	...	...
result	494	Enc("494")	error

Table 2: Searching Results in ASC II

Table 2. The cloud cannot decrypt $Enc(52)/Enc(57)/Enc(52)$, hence, the users' privacy is guaranteed. The user holds the secret key, therefore, he/she is the only one who knows the searching results, while the cloud cannot even distinguish the difference between the two $Enc(52)$.

However, if the cloud acts maliciously, it can recover one bit of message from c through our attack model, since it adds a value to the keyword. Hence, if $Dec(c) = 0$, the algorithm will search for Enc(GOOG) as before, and therefore, no error will occur, and the user will most likely do nothing. However, if $Dec(c) = 1$, instead of searching for Enc(GOOG), the input of the algorithm is actually Enc(GOOH). Therefore, no match will be found. It is reasonable to believe that the user will start to execute another search, in which case the cloud increases ε_1.

As we have stated, a malicious cloud can also modify the circuit/result accordingly. However, we notice that in the above example, modifying the result merely helps our attack, as if the cloud induces an $Enc(1)$, the user will receive 495, which will be recognized as a valid result.

Formal Construction In this section we show a formal construction of evaluating users' reactions in terms of timing. We note that our formal construction is only a function of the time the user responses. In practice, with the aid of other parameters, the malicious service provider can further increase the successful rate of this attack.

Let t_{res} be the time period when the user starts a new task after receiving the result from the cloud. Let t_n be the average time when the user sets up his/her tasks when no error is induced, and t_e be the minimum time when the user starts a new task if some error occurs. Assume that t_{res} follows a certain distribution $\mathcal{D}_0 = f_0(t)$ depending on the average time t_n when no error occurs, and another distribution $\mathcal{D}_1 = f_1(t)$ depending on the t_e when there exists an error. As a service provider, we assume the cloud is aware of above information.

We note that the success of the message attack relies on the difference between $\mathcal{D}_0$ and $\mathcal{D}_1$. For users whose reactions are completely random, i.e., $\mathcal{D}_0 = \mathcal{D}_1$, our attack will not be successful. However, we argue that, in practice, a reasonable user will have different distributions for different results.

For any time frame t, $\mathcal{D}_0$ has a $f_0(t)$ probability to distribute a new task, while $\mathcal{D}_1$ has a $f_1(t)$ probability to distribute a new task. As a result, if the cloud receives a new task at time t, the confidence of $m = 0$ and $m = 1$ can be determined by $g_0(x) = \frac{f_0(t)}{f_0(t)+f_1(t)}$ and $g_1(x) = \frac{f_1(t)}{f_0(t)+f_1(t)}$, respectively.

Later, we can anticipate the successful rate of our attack as follows. Let $\psi = g_1(t_1) = g_2(t_2)$ $(t_2 > t_1)$, when the cloud wants to build a minimum ψ confidence. Therefore, any task delivered prior to t_1 will give the cloud at least ψ confidence of $m = 1$, while any task delivered after t_2 will give the cloud at least ψ confidence of $m = 0$. Then we can evaluate the successful rate ε. Generally, it can be expressed by the volume of $f_1(x)$ in $[0, t_1]$ and $[t_2, \infty)$ over the total volume. More formally,

$$\varepsilon = \frac{\displaystyle\int_0^{t_1} f_1(x)\,dx + \int_{t_2}^{\infty} f_1(x)\,dx}{\displaystyle\int_0^{\infty} f_1(x)\,dx}.$$

In the case the cloud has not developed sufficient confidence for a certain message (*i.e.*, it receives a new task at time between t_1 and t_2), it will induce the same faulty information to the next several tasks. Suppose the cloud takes n tasks to develop the confidence, and the corresponding response time are $t_1, t_2, \ldots, t_n$, then the confidence can be determined by $\psi' = \max(\psi'_0, \psi'_1)$, where

$$\psi'_0 = \frac{\prod_i^n g_0(t_i)}{\prod_i^n g_0(t_i) + \prod_i^n g_1(t_i)}, \quad \psi'_1 = \frac{\prod_i^n g_1(t_i)}{\prod_i^n g_0(t_i) + \prod_i^n g_1(t_i)}.$$

For instance, if the adversary sends an identical message twice, with response time t_1 and t_2, then the possibility of two continuous 0-s and 1-s are $g_0(t_1)g_0(t_2)$ and $g_1(t_1)g_0(t_2)$, respectively. We note that it is not possible to have 10 or 01, as we are using the same ciphertext. As a result, the possibility of 10 and 01 are eliminated. Therefore, we have

$$\psi'_0 = \frac{g_0(t_1)g_0(t_2)}{g_0(t_1)g_0(t_2) + g_1(t_1)g_1(t_2)}, \quad \psi'_1 = \frac{g_1(t_1)g_1(t_2)}{g_0(t_1)g_0(t_2) + g_1(t_1)g_1(t_2)}.$$

Meanwhile, the new successful rate can be determined by $\varepsilon' = \frac{\varepsilon^n}{\varepsilon^n + (1-\varepsilon)^n}$. This guarantees that we can achieve a very high confidence/successful rate by attacking the same ciphertext repeatedly.

To exemplify our construction, without losing generality, we show a example where users' reaction follows a normal distribution (also known as Gaussian distribution) [15] with the distribution factor $\sigma = 2$ (see Figure 4). We note that for simplicity we uses smooth curves to illustrate the distributions, while in practise, the actual distribution will be more discrete.

In this example, $t_n = 10$ minute. Meanwhile, we assume the transmission time/evaluation time is negligible, which implies $t_e = 0$. From this figure, we obtain a figure that illustrates the confidence of messages in Figure 5. Hence, if the adversary requires 96% confidence in a single round, the corresponding t_1 and t_2 are 4 and 6.5, respectively. Therefore, the successful rate by one single round can be determined by $\frac{V_1+V_0}{V_1+V_0+V_e}$, which is essentially $\frac{\int_0^4 f_1(x)\,dx+\int_{6.5}^{\infty} f_1(x)\,dx}{\int_0^{\infty} f_1(x)\,dx}$.

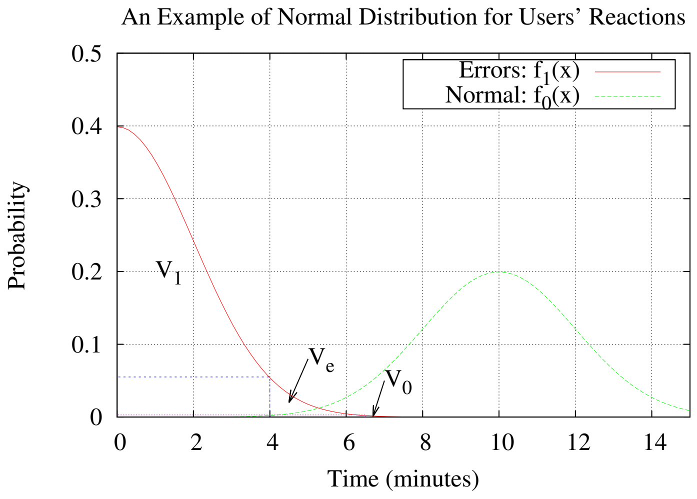

Fig. 4: An example of Users' Reactions Using Normal Distribution

On CCA Security It is well known that for any public key encryption schemes, the CPA security is essential. Meanwhile, constructing a CCA-2 secure fully homomorphic encryption scheme is impossible, since homomorphic operations on ciphertexts are enabled (and also it is due to the "malleability" of the ciphertext). Moreover, unfortunately, fully homomorphic encryption schemes that follow Gentry's framework cannot be CCA-1 secure due to the bootstrapping technique. In fact, if a somewhat homomorphic encryption scheme is CCA-1 secure is questionable. Indeed, in [16], the authors presented a CCA-1 attack against Gentry-Halevi SHE scheme.

We note that the consequence of such an attack might be severe. A CCA-1 attack is an attack model that assumes there exists a decryption oracle. People may

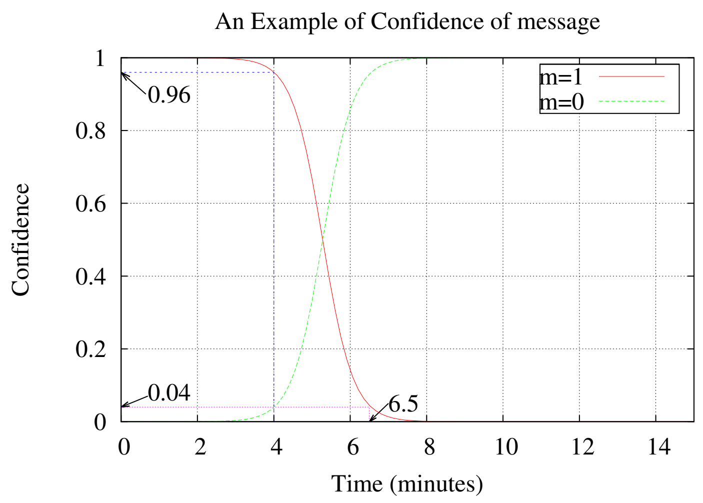

Fig. 5: An example of Confidence Curve

argue that this is just merely an attack model, since in practice, having such an oracle (or an honest user who is helping the attacker during the learning phase) is impractical. Further, constructing such an oracle in general is not really feasible. Nevertheless, with our message attack, it is possible to construct a probabilistic decryption oracle in practice. Consequently, if for a certain fully homomorphic encryption scheme, a CCA-1 attack is successful with a non-negligible advantage of ϕ using a hypothetical deterministic oracle, then one can achieve this attack with an advantage of $\phi\varepsilon^n$ using our message attack, where n is the number of ciphertext required for the CCA-1 attack. Hence, the CCA-1 attack becomes really practical.

To sum up, we argue that in practice, if a fully homomorphic encryption scheme is not CCA-1 secure, then it alone cannot deliver secured outsourced computation.

3.2 The Secret Key Attack

In this subsection, we propose our secret key attack. We show that for *any* FHE scheme, even its SHE schem is CCA-1 secure, as long as it uses Gentry's bootstrapping technique, it is vulnerable to our attack.

Recall that in Gentry's framework, binary operations ($i.e.$, $\oplus, \otimes$) of messages are eventually $+$ and $\times$ of ciphertext in a FHE scheme; also, to enable "bootstrappability", one has to publish an encryption of its secret key. Therefore, for any FHE schemes that follows Gentry's framework, no matter what kind of circuit the user demands, with our message attack, a malicious cloud recovers one bit of the secret key through each attack. Eventually, it will recover the whole secret key.

For instance, as in vDGHV scheme, one publishes $\{Enc(s_i)\}$. The cloud can recover s_i through each message attack, and recover S as a result. As another example, the secret key in Gentry-Halevi scheme [17] is a bit sequence of 1024, while the number of encrypted 1-s is 15. Hence, it takes a maximum 1024 message attacks to recover the secret key.

One may argue that the secret key may contain too many bits, and therefore, it is impractical to recover them all. For instance, S in vDGHV scheme contains λ^5 digits, *i.e.*, $\eta \sim O(\lambda^5)$. Consequently, to recover all bits becomes impractical.

However, we note that, firstly, our attack can be launched with other attacks. For instance, with the existence of a decryption oracle, the attack in [16] recovers the secret key of Gentry-Halevi scheme within $O(\lambda^2)$ queries, consequently, a CCA attack can break the system with an ε^{λ^2} advantage.

Secondly, we notice that the secret keys of Gentry scheme, and that of all other schemes following Gentry's framework, are sparse sequences with a significantly smaller Hamming weight. As an example, in vDGHV scheme, the secret key $S =< s_1, s_2, \ldots, s_\eta >$ only has θ 1-s, while the rest are 0-s and $\theta \ll \eta$ (*i.e.*, $\theta = \lambda$, $\eta = \lambda^5$). Hence, inducing $Enc(s_i)$ will not result into any error, in most cases. In practice, this means using our attack merely increases the error rate by $1/\lambda^4$. This enables a much higher attacking rate for the cloud. The cloud can induce encrypted secret key bits at all time without being detected.

Finally, because of the special structure of the secret key, we propose an optimized secret key attack that improves the performance of our secret key attack significantly. As a result, it only requires $O(\lambda \log \lambda)$ operations to recover all λ^5 bits. We will describe this optimized attack in the following section.

Optimized Secret Key Attack In Gentry's framework, the secret key $S =< s_1, s_2, \ldots, s_\eta >$ contains η bits, while the Hamming weight $\theta = \sum_i^\eta s_i$ is significantly smaller than η. Therefore, we propose two optimizations, using fully Dichotomy search and block Dichotomy search algorithm.

For the block Dichotomy search algorithm, if we cut the whole secret key into k blocks, with each block $l = \eta/k$ bits, with a very high probability the block will only consist of 0-s. Now, instead of testing the parity of each bit, we test if a block contains only 0-s. To perform such a task, one generates

$$b = Enc(1) + \prod_i^l (Enc(1) + Enc(s_i)),$$

and test the parity of b. Because homomorphic operations are enabled, if all the bits are 0-s, then decrypting b will give us 0, and vice versa. As a result, the adversary recovers l bits in one message attack.

There is one exception, where there are at least one 1 in the block. Hence, a Dichotomy search algorithm is required. The average case takes place when there are no more than one encrypted 1 in a block. Therefore, for each encrypted 1, it requires additional $\log(l)$ message attacks.

To sum up, this optimization reduces the number of message attacks from η to $k + \theta \log(\eta/k)$. With the configuration of vDGHV scheme/Gentry scheme

($\theta \sim \lambda$ and $\eta \gg \theta$), the minimum number for message attacks is $O(\lambda \log \lambda)$. While with the configuration of Gentry-Halevi scheme ($\theta = \lambda / \log \lambda$ and $\eta = 2^{\lceil 2 \log \lambda \rceil}$), the minimum number for message attacks is $O(\lambda)$.

While for the fully Dichotomy search algorithm, one cuts the secret key into two halves for each round. For each half, one generates $b' = Enc(1) + \prod_i (Enc(1) + Enc(s_i))$. If one or more 1-s is anticipated in any piece, the adversary feeds the user with the inverse of b', and vice versa. This method is to ensure that the induced ciphertexts always have a higher probability of encrypting 0-s, and consequently, the error rate will be minimized. For instance, for the first round, any half has $1 - \binom{\eta/2}{\theta} / \binom{\eta}{\theta}$ possibility of having at least a 1. Therefore, the cloud sends $b' + 1$.

In the worst case, the fully Dichotomy search algorithm requires $\theta \log(\eta/\theta) + \theta$ enquiries. Applying over the FHE schemes, we observe a similar result with the previous optimization, *i.e.*, $O(\lambda \log \lambda)$ for vDGHV scheme/Gentry scheme and $O(\lambda)$ for Gentry-Halevi scheme. However, we note that in practice, this optimization might work better, because we will have a higher probability to eliminate a big block of 0-s.

We use vDGHV scheme to exemplify our secret key attack. The recommended configuration for this scheme is $\theta = 80$ and $\eta = 80^5$. Our optimization reduces the number of messages attacks from 80^5 to approximately 2100. Meanwhile, the actual successful rate for the secret key attack depends on the confidence from the message attack. Suppose through the message attack the cloud develops 0.999 confidence on the message (this ratio can be achieved by launching the message attack repeatedly on a same message), then the cloud can recover the secret key with a probability of $0.999^{2100} = 0.122$. That is to say, in average case, the cloud needs to use the secret key attack for 8 times to recover the secret key, which is unacceptable by the users.

4 Discussion

4.1 Practicality of Our Attack

In practice, errors cannot be eliminated in the outsourced computation scenario. The users cannot distinguish if it is caused by a malicious service provider, or by some connection/transmission error, or even by the algorithm itself. As we have shown before, most of our message attacks in the (optimized) secret key attack will not cause errors. Even with our optimization, the error rate is still significantly small, compared with other causes. If $\lambda = 80$, the error rate can be as small as 2.44×10^{-8} for the original secret key attack. Meanwhile, the cloud can manipulate the attacking rate, *i.e.*, it attacks only when the current error rate of the network is lower than normal. Therefore, the user is incapable of detecting this attack.

As a result, most of the decryption errors is generated by other reasons. The user cannot afford to transfer its data to another cloud every time an error occurs, since transferring encrypted data is too costly.

One may also argue that when the attacker induces $Enc(0)$ and it might also be so coincidence that a user is proceeding another search subsequently. Consequently, the attacker would recognize the corresponding bit to be 1. Will this mislead the attacker? Possibly. This is the reason why we stated that our attack is probabilistic, and we have shown that our attack can minimize the impact of this event to occur. The attacker performs one attack during a period, instead of one interaction. During this period, ε_1 may have already been decreased a lot times for other reasons. Therefore, one increase will *not* mislead the cloud after all.

4.2 Protecting FHE with Verifiable Computation

In theory, using FHE with verifiable computation will stop our attack in model 1 and 2. Since the user is able to verify the computation, the cloud will not be able to modify the computation circuit without be detected. As a result, our attack will be unsuccessful. However, we argue that, in order to use FHE in those models, one *must* use verifiable computation all the time.

Moreover, we also observe that, although the cost of verification is low, to generate the minimum circuit and to homomorphically modify it is costly. Thus, whether this technique can be used in practise is doubtful.

Finally, as far as model 3 is concerned, the computation circuit is private to the cloud, hence, verifiable computation protocols are not applicable in this scenario.

4.3 Other Possible Protections

A possible solution to the secret key attack could be removing the necessity of bootstrapping technique. If there is such a FHE scheme that supports arbitrary circuit depth without the bootstrapping technique, then the users can avoid publishing his/her encrypted secret keys. Unfortunately, so far all FHE schemes except [18] follow Gentry's framework, and to bootstrap is essential for them to achieve fully homomorphic. Meanwhile, even if there exists a FHE scheme without bootstrapping, by incorporating our attack, the attacker is still capable of recovering data (*but not* the secret key) from the users.

To stop the message attack is much more difficult, as one cannot determine where an error is actually from. A possible solution could be setting up certain protocols between the service provider and the users. This protocol has to minimize the error rate. Further, whenever an error occurs, the provider has to show the users' full details of the evaluation circuit, in order to convince the users.

Another possible but expensive solution would be letting the users to generate random "meaningless" tasks (or "stubs") periodically. Whenever the user needs to use the service, he/she replaces a stub with the one he/she really requires. Even though the user receives some errors and needs to set up the same task again, he/she does not process immediately. Instead, he/she will wait till the next period of sending tasks. As a result, the attacker will not be able to distinguish if a task is a valid one, or a repeated one (due to message attack), or merely a random one. Also, the overall average task rate remains the same.

However, we note that this solution is very expensive, as it requires periodic communications between the users and the cloud. Meanwhile, the user sometimes has to wait for several periods when he/she requires multiple tasks. As a result, the availability of the cloud is not always ensured. Further, in practice, the service providers charge users based on their tasks, and additional random meaningless tasks will significantly increase the cost of using cloud computing.

5 Conclusion

It is widely believed that cloud computing has become the next stage of the Internet, as it enables outsourced computation. However, how to ensure information security and users' privacy remains a challenging open problem. In this paper, we showed that, fully homomorphic encryption schemes, although seem to be a promising candidate, have some problems when they are used in the context of cloud computing.

Subsequently, we presented a practical message attack against *all* fully homomorphic encryption schemes, in that a malicious cloud can recover the messages by observing users reactions. With several examples, we showed that in practice, our message attack has a very high probability to be successful.

In addition, this message attack can be extended to construct a probabilistic decryption oracle. This brings CCA-1 security as an essential requirement for constructing a secure fully homomorphic encryption scheme.

Further, because of the bootstrapping technique that is used in Gentry's framework, we obtain a secret key attack against *all* fully homomorphic encryption schemes that follow this framework [11, 4, 3, 9, 17, 19, 10, 12, 16], and this secret key attack is very practical that only takes a maximum $O(\lambda \log \lambda)$ time.

Finally, we argued that CCA-1 security and *no* bootstrappability are the two essential requirements for fully homomorphic encryption schemes that can be used to secure cloud computing scenarios.

References

1. van Dijk, M., Juels, A.: On the impossibility of cryptography alone for privacy-preserving cloud computing. Cryptology ePrint Archive, Report 2010/305 (2010) `http://eprint.iacr.org/`.
2. Gennaro, R., Gentry, C., Parno, B.: Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In: CRYPTO. (2010) 465–482
3. Gentry, C.: A Fully Homomorphic Encyrption Scheme. PhD thesis, Stanford University (2009)
4. Gentry, C.: Fully homomorphic encryption using ideal lattices. In Mitzenmacher, M., ed.: STOC, ACM (2009) 169–178
5. Hall, C., Goldberg, I., Schneier, B.: Reaction attacks against several public-key cryptosystems. In Varadharajan, V., Mu, Y., eds.: ICICS. Volume 1726 of Lecture Notes in Computer Science., Springer (1999) 2–12

6. Myers, S., Shelat, A.: Bit encryption is complete. In: FOCS, IEEE Computer Society (2009) 607–616
7. Rivest, R., Adleman, L., Dertouzos, M.: On data banks and privacy homomorphisms. In: Foundations of Secure Computation, Academic Press (1978) 169–177
8. Rivest, R.L., Shamir, A., Adleman, L.M.: A method for obtaining digital signatures and public-key cryptosystems. Commun. ACM **21**(2) (1978) 120–126
9. Smart, N.P., Vercauteren, F.: Fully homomorphic encryption with relatively small key and ciphertext sizes. In Nguyen, P.Q., Pointcheval, D., eds.: Public Key Cryptography. Volume 6056 of Lecture Notes in Computer Science., Springer (2010) 420–443
10. Stehlé, D., Steinfeld, R.: Faster fully homomorphic encryption. In Abe, M., ed.: ASIACRYPT. Volume 6477 of Lecture Notes in Computer Science., Springer (2010) 377–394
11. van Dijk, M., Gentry, C., Halevi, S., Vaikuntanathan, V.: Fully homomorphic encryption over the integers. In Gilbert, H., ed.: EUROCRYPT. Volume 6110 of Lecture Notes in Computer Science., Springer (2010) 24–43
12. Gentry, C., Halevi, S.: Implementing gentry's fully-homomorphic encryption scheme. In Paterson, K.G., ed.: EUROCRYPT. Volume 6632 of Lecture Notes in Computer Science., Springer (2011) 129–148
13. Goldwasser, S., Micali, S.: Probabilistic encryption. J. Comput. Syst. Sci. **28**(2) (1984) 270–299
14. Bellare, M., Desai, A., Pointcheval, D., Rogaway, P.: Relations among notions of security for public-key encryption schemes. In Krawczyk, H., ed.: CRYPTO. Volume 1462 of Lecture Notes in Computer Science., Springer (1998) 26–45
15. Georgii, H.O.: Stochastics: Introduction to Probability and Statistics (de Gruyter Textbook). 1 edn. Walter de Gruyter (2008)
16. Loftus, J., May, A., Smart, N., Vercauteren, F.: On cca-secure fully homomorphic encryption. In: Selected Areas in Cryptography. (2011)
17. Gentry, C., Halevi, S., Vaikuntanathan, V.: i-Hop Homomorphic Encryption and Rerandomizable Yao Circuits. In Rabin, T., ed.: CRYPTO. Volume 6223 of Lecture Notes in Computer Science., Springer (2010) 155–172
18. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: Fully homomorphic encryption without bootstrapping. Electronic Colloquium on Computational Complexity (ECCC) **18** (2011) 111
19. Gentry, C.: Computing arbitrary functions of encrypted data. Commun. ACM **53**(3) (2010) 97–105