A Study on Relinearize Problem in Fully Homomorphic Encryption

A Thesis Submitted to the Department of Computer Science and Communications Engineering, the Graduate School of Fundamental Science and Engineering of Waseda University in Partial Fulfillment of the Requirements for the Degree of Master of Engineering

Submission Date: February 1st, 2019

Hiroki Sato

(5117F041-9)

Advisor: Prof. Hayato Yamana

Research Guidance: Research on Parallel and Distributed Architecture

Abstract

Fully homomorphic encryption (FHE) allows arbitrary computation on encrypted data without any decryption key. However, ring-LWE-based FHE schemes have a problematic feature: ciphertext size increases with every homomorphic multiplication. Furthermore, the computation cost of homomorphic multiplication linearly increases with increasing input sizes. To overcome this, these FHE schemes support a special operation called *relinearization*, which can reduce the ciphertext size. Relinearization requires almost the same amount of computation cost as that of the homomorphic multiplication, which takes a few to hundreds of milliseconds. Thus, determining when and the number of times to relinearize a ciphertext in a given arithmetic circuit to evaluate in order to minimize the total computation cost is an important task. This problem has been proved to be an NP-hard problem, and is called the *relinearize problem*.

In this study, we design an approximation algorithm to address the relinearize problem. The algorithm runs in polynomial time, and we experimentally confirmed that the output of the algorithm is nearly the same as the optimal solution. In particular, we show that the output is exactly equal to the optimal one in a specific case.

Contents

Chapter 1

Introduction

Fully homomorphic encryption (FHE) allows evaluation of any circuit on encrypted data without decryption. FHE enables us to securely delegate computation on sensitive data to a third party while retaining data confidentiality. Starting from Gentry's first construction [20], various FHE schemes have been proposed [5, 6, 21, 15, 19]. FHE has a wide range of applications, including machine learning on encrypted data [4, 22, 18], private genomic analysis [23], and private database query [3].

Though several years of research have improved the efficiency of FHE, homomorphic operations are still heavy compared to operations on plaintexts. Therefore, speeding up computations over FHE should be addressed not only from the cryptographic aspect but also from the application aspect, i.e., how to use it. Typically, a function to be executed can be represented as an arithmetic circuit. The circuit forms a directed acyclic graph (DAG), allowing optimizations to be performed on the graph. Although we can apply a general optimization strategy used in traditional compilers to the graph, the same optimization strategy is not sufficient when the circuit is evaluated over FHE because some FHE-specific operations cannot be represented on the graph.

In this study, we focus on the B/FV scheme [5, 19], one of the most efficient and widely used FHE schemes. For example, various implementations [28, 27] and applications to secure computation [18, 1] with the B/FV scheme are published. In the B/FV scheme, a ciphertext is a vector of polynomials; the length of a ciphertext just after encryption is 2. When homomorphic multiplication is performed on the ciphertext, the length of the ciphertext increases. The computation time of homomorphic addition and multiplication is proportional to the ciphertext length. As a result, both time and space complexity grow as the computation with multiplication proceeds because the ciphertext length continues to increase as well. Thus, controlling the ciphertext length is critical. To control the length of the ciphertext, the B/FV scheme supports an operation called *relinearization*, which reduces

the length of the ciphertext. Relinearization approximately involves the same amount of computation cost as that of homomorphic multiplication. A simple strategy to handle the ciphertext length is to relinearize the ciphertext after every multiplication; however, this strategy is not always optimal. Given a circuit, we can reduce the total computation time by adjusting both when and the number of times to relinearize a ciphertext. This optimization problem, called *relinearize problem*, was first introduced by Chen in 2018 [10] and proved to be NP-hard.

In this study, we design a polynomial-time approximation algorithm to address the relinearize problem. We show that our method outputs nearly the same result as the optimal one, or equal to in a specific case. More precisely, our contributions are as follows:

- We propose a polynomial-time approximation algorithm to the relinearize problem, adapting the idea used in the bootstrap problem [26].
- We experimentally show that our algorithm outputs nearly the same result as the optimal solution.
- We prove that in a specific FHE parameter setting, our algorithm outputs exactly the same result as the optimal one.

The remainder of this thesis is organized as follows. We first describe the related work in Chapter 2, followed by a formulation of the *relinearize problem* in Chapter 3. Then, we present our proposed algorithm and explain the case where the result is equal to the optimal one in Chapter 4 and provide evaluation results in Chapter 5. Finally, we conclude the study in Chapter 6.

Chapter 2

Background and Related Work

Homomorphic encryption is a useful tool to realize secure computation. However, there are some differences with the computation on the plaintext, which makes it difficult to build an application over FHE as follows:

- We cannot use any branching based on an encrypted value.
- We cannot use traditional table-lookup from an encrypted input.
- We can encrypt multiple plaintexts as a batch into a single ciphertext and evaluate them in a single-instruction multiple-data manner.
- Operations such as bootstrapping and relinearization, described in Section 2.1 and in Section 2.2, are essential but are not usually mentioned in a high-level algorithm description because these operations are used for the maintenance of a ciphertext and do not modify the underlying plaintext.

Thus, devising a well-optimized algorithm on encrypted data is more difficult than on the plaintexts. It is important to bridge the gap between an algorithm written in a high-level programming language and a sequence of low-level operations on the encrypted data. This translation can be seen as a compiler for FHE.

There exist several compiler-like studies that translate computation from a user-friendly language to algorithms that can be directly handled by FHE [8, 9, 17, 16]. The main focus of these studies was to reduce the complexity of homomorphic operations and simplify a process to write applications for secure computation. The authors of these studies mentioned some optimization capabilities but did not discuss optimization techniques in detail.

A few research studies focused on optimizations considering characteristics of FHE. In FHE, the smaller the multiplicative depth of the circuit[1],

[1] The multiplicative depth of the circuit is defined as the largest number of multiplications an input ciphertext will be applied to through the circuit.

the smaller the parameters we can use and the faster the evaluation of the circuit. Carpov et al. [7] proposed a heuristic optimization technique to reduce the multiplicative circuit depth. They used an associative rule and a distribution rule to modify the evaluation order while keeping the same final output, and lowered the multiplicative depth of the circuit, though the total number of arithmetic operations including both additions and multiplications increased.

Addition and multiplication are explicitly shown in a data-flow graph. In contrast, bootstrapping and relinearization operations are used only for the maintenance of a ciphertext and do not modify the underlying plaintext. Thus, both operations are not shown in the data-flow graph. When we optimize the usage of these operations, it is difficult to adopt tools or strategies used in other research areas like circuit synthesis or compiler for CPUs. Examples of circuit optimizations specific to the FHE are the *bootstrap problem* and the *relinearize problem*; we explain them below.

2.1 Bootstrap Problem

All the existing FHE schemes have the same property; every ciphertext includes some amount of noise that grows with every arithmetic operation. If the amount of noise reached the predetermined threshold, the decryption result is incorrect. To keep the amount of noise within the threshold, we need an operation called bootstrapping to reduce the noise. However, the bootstrapping takes huge computation time. Given a circuit to evaluate, minimizing the number of bootstrapping operations directly results in reducing the entire computation time. The minimization problem is called the *bootstrap problem*.

Previous research on the bootstrap problem focused on leveled FHE. In the leveled FHE, the parameter *level* $L \in \mathbb{N}_+$ is decided upon key generation, and this parameter indicates the number of times that the homomorphic multiplication can be applied to each ciphertext between the bootstrapping.

The bootstrap problem was first pointed out by Lepoint and Paillier [24] in 2013, and they modeled the problem based on a boolean satisfiability problem. Later, in 2015, Paindavoine and Vialla [26] analyzed the bootstrap problem using graph theory. The bootstrap problem is NP-hard due to a reduction from the vertex cover problem. In particular, Paindavoine and Vialla showed that in the case where level $L = 2$, the problem is solved in polynomial time by a reduction to (s, t) min-cut. Here, we define a data-flow graph $G = \{V, E\}$ with vertices V and edges E. The maximum-flow algorithm runs in $O(|V||E|)$ [25]; thus, the bootstrap problem under $L = 2$ can be solved in $O(|V||E|)$. In 2017, Benehamouda et al. [2] proposed a L-approximation algorithm for the bootstrap problem using linear programming and dynamic programming, and showed this approximation is a theoretical bound.

2.2 Relinearize Problem

Bootstrapping involves heavy computation, which takes a few seconds to a few minutes, in ring-based state-of-the-art schemes [12, 13, 11]. Owing to this heavy computation, many researchers are trying to build a practical application with a low depth circuit in which the homomorphic operations are completed without invoking the bootstrapping. In such cases, the bottleneck of the homomorphic computation is not the bootstrapping but homomorphic multiplication and relinearization.[2] Therefore, the bootstrap problem is no longer a concern.

If we consider a ciphertext as a vector of polynomials and denote the length of a ciphertext c as $l(c)$ (≥ 2), multiplication over ciphertexts as $\otimes$, and addition as $\oplus$, then $l(c_1 \otimes c_2) = l(c_1) + l(c_2) - 1$ and $l(c_1 \oplus c_2) = \max(l(c_1), l(c_2))$. By performing homomorphic multiplication, the length of the ciphertext increases and the computation time and memory cost of multiplication also increases. The relinearization can reduce the length of the ciphertext but incurs almost the same amount of the cost as that of the homomorphic multiplication. Here, the question is when and the number of times a ciphertext should be relinearized in a given circuit in order to minimize the total computation time while keeping both the data-flow graph and the underlying plaintext the same. This is called the *relinearize problem*.

The relinearize problem was first observed by Chen [10] in 2018. Chen proved that the problem is an NP-hard and provided a polynomial-time algorithm for the special case where each vertex has at most 1 out-degree. However, there are many circuits that have a vertex with 2 or more out-degrees, and thus the applicable circuits are limited in real-world applications. There is no other practical approach to the relinearize problem to the best of our knowledge.

[2] This is because homomorphic additions are less time-consuming compared to multiplications or relinearizations.

Chapter 3

Problem Formulation

In this chapter, we formally describe the relinearize problem, following Chen's definition [10].

Definition 1. An arithmetic circuit is a directed acyclic graph (DAG) $G = (V, E)$, where there are three kinds of vertices:

- input vertices that have in-degree 0 and out-degree 1.
- output vertices that have in-degree 1 and out-degree 0.
- add/multiply operation vertices that have in-degree 2 and out-degree 1.

We denote $e = (v_1, v_2) \in E$ as an edge from vertex v_1 to vertex v_2 ($\{v_1, v_2\} \in V$).

Definition 2. The *relinearize problem* is an integer programming problem on an arithmetic circuit. For every vertex i, we maintain three integer variables x_i, $l(i)$ and $l^{new}(i)$ during the homomorphic evaluation of G. x_i (≥ 0) denotes the number of relinearizations executed just after the evaluation of add or multiply operation at vertex i. $l(i)$ (≥ 2) denotes the length of the ciphertext at vertex i after the evaluation but before performing relinearization. $l^{new}(i)$ (≥ 2) denotes the length of the ciphertext at vertex i after executing both evaluation and relinearizations.[1] Thus, $l^{new}(i) = l(i) - x_i$ holds. We denote the two parents of vertex i by $p_1(i)$ and $p_2(i)$. We denote the kind of vertex i by $t(i)$. $t(i)$ is one of $\{\otimes, \oplus, \texttt{in}, \texttt{out}\}$, where $\otimes$ is a multiply operation over ciphertexts, $\oplus$ is an add operation, `in` is an input, and `out` is an output. We denote the computation time of a single relinearization by k_r (> 0). We denote the computation time of a multiplication with input lengths of $l^{new}(p_1(i))$ and $l^{new}(p_2(i))$ by $k_m l(i) = k_m(l^{new}(p_1(i)) + l^{new}(p_2(i)) - 1)$,

[1] When we do not relinearize at vertex i, $l(i) = l^{new}(i)$ holds.

Table 3.1: Notations and Definitions

Notation	Definition
G	a graph
E	a set of edges in a graph
V	a set of vertices
x_i	the number of relinearizations executed at vertex i
$l(i)$	the length of the ciphertext at vertex i after the evaluation but before performing relinearization
$l^{new}(i)$	the length of the ciphertext at vertex i after executing both evaluation and relinearizations
$t(i)$	the kind of vertex i
$\otimes$	homomorphic multiply operation veretx
$\oplus$	homomorphic add operation vertex
`in`	input vertex
`out`	output vertex
k_r	the computation time of a single relinearization
K_m	a factor of proportionality of a computation time of a homomorphic multiplication

where k_m (> 0) is a factor of proportionality.[2] Then, the *relinearize problem* on G is
minimize

$$k_r \sum_{i \in V} x_i + k_m \sum_{t(i)=\otimes, i \in V} (l^{new}(i) + x_i) \tag{3.1}$$

subject to

$$\left.\begin{array}{ll} l^{new}(i) \geq 2 & \text{for all } i \\ l^{new}(i) \geq l^{new}(p_1(i)) + l^{new}(p_2(i)) - 1 - x_i & \text{if } t(i) = \otimes \\ l^{new}(i) \geq l^{new}(p_1(i)) - x_i & \text{if } t(i) = \oplus \\ l^{new}(i) \geq l^{new}(p_2(i)) - x_i & \text{if } t(i) = \oplus \\ l^{new}(i) = 2, x_i = 0 & \text{if } t(i) = \texttt{in} \\ l^{new}(p_1(i)) - x_i = 2 & \text{if } t(i) = \texttt{out} \\ x_i, l^{new}(i) \in \mathbb{Z} & \text{for all } i \end{array}\right\} \tag{3.2}$$

The definitions of variables we used is summarized in Table 3.1.

The second to the fourth constraints in Equation (3.2) are relations of input and output ciphertext lengths at each operation vertex. We use $\geq$ instead of $=$ because we can increase a ciphertext length by homomorphically adding a constant ciphertext encrypting 0 whose ciphertext length is $l^{new}(i)$.

[2] In this work, k_r and k_m are given parameters, measured beforehand. We ignore the time of the homomorphic addition because it is several magnitudes faster than both multiplication and relinearization.

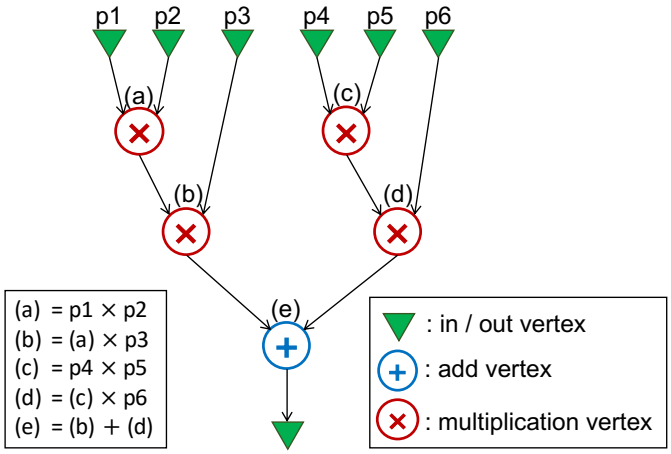

Figure 3.1: A circuit example.

Note that the original definition by Chen [10] consists of four kinds of vertices: the same three kinds of vertices in Definition 1 and square operation vertices. We omit square operation vertices because they can be replaced by a multiply operation vertex with a multi-edge input. Moreover, we add the constraints that the lengths of the ciphertext at both input and output vertices should be 2. This is because we followed what general compilers do, that is, they divide a given code into multiple blocks and later concatenate these blocks of inputs and outputs so that every input and output should satisfy the same condition.

The definition above assumes the B/FV scheme. This problem can be considered for other ring-based FHE schemes such as the BGV scheme [6] and the approximate HE [14].

3.1 An Example of Relinearize Problem

For example, we consider the circuit shown in Figure 3.1. In this circuit, there are six input variables (vertices from `p1` to `p6`), one output variable, four multiply operation vertices and one add operation vertex.

Firstly, we consider a simple strategy in which we relinearize at every multiply operation vertex. From the constraints in Equation (3.2), length of each ciphertext at every input vertex from `p1` to `p6` is 2. At the multiply operation vertices from `(a)` to `(d)`, the length of each output ciphertext $l(i)$ is 3. They becomes 2 after applying relinearizations, i.e. $x_i = 1, l^{new}(i) = 2$. Thus, the objective value defined in Equation (3.1) is $4k_r + 12k_m$. Secondly, instead of relinearizing at vertex `(b)` and `(d)`, we can relinearize at `(e)` while maintaining the constraints. In this case, the objective value becomes $3k_r + 12k_m$, which is smaller than the previous one. Lastly, even when we

relinearize twice at the vertex `(e)` and not at the other vertices, it satisfies the constraints and the objective value is now $2k_r + 14k_m$. As long as $k_r > 2k_m$, the last strategy is preferred; otherwise, the second strategy is preferable.

As observed here, the optimal case depends on both the circuit and parameters k_r and k_m.

3.2 Naïve Solution

The simplest solution is that we relinearize at every multiply operation vertex. However, this is not optimal in some cases such as the example shown in Section 3.1.

From Definition 2, the problem is integer linear programming (ILP), which is an NP-hard problem. However, we can use a solver to obtain the optimal result, though the time and space complexity are not bounded by a polynomial.

Chapter 4

Proposed Method

In this chapter, we propose an approximation algorithm for the relinearize problem, followed by a proof of the special case where our method outputs exactly the same result as the optimal one.

In our proposed method, we add the following constraint to the previous formulation in Equation (3.2):

$$l(i) = l^{new}(i) + x_i \leq 3 \quad \text{for all } i \tag{4.1}$$

This constraint restricts the length of every ciphertext to 2 or 3. With this constraint, the problem can be reduced to (s, t) min-cut problem and be solved in polynomial time.

4.1 Polynomial-time Algorithm

The relinearize problem with the additional constraint in Equation (4.1) can be solved in polynomial time by reduction to a (s, t) min-cut problem. This reduction method is inspired by Paindavoine and Vialla's approach [26] proposed for a special case of the bootstrap problem.

Before describing our algorithm, we define an *interesting-path* to support our explanation.

Definition 3. An *interesting-path* in a data-flow graph is a path $v_1, \ldots, v_k$ where $t(v_1) = \otimes$ and $t(v_i) = \oplus, i \in [2, k]$ holds and there exists an edge (v_k, v_{k+1}) such that $t(v_{k+1}) \in \{\otimes, \texttt{out}\}$ holds.

Our proposed algorithm is shown in Algorithm 1. Here, we will provide the explanation and proofs in detail below.

Lemma 1. The relinearize problem restricted by the additional constraint Equation (4.1) can be solved by minimizing the number of vertices to perform a relinearization in the graph.

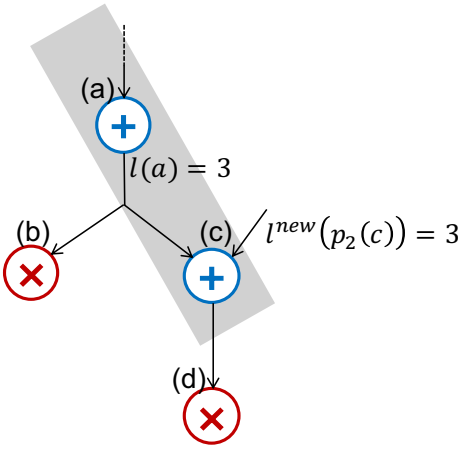

Figure 4.1: An example of *interesting-path* including multiple relinearizations.

Proof. If some multiply operation vertex v accepts a ciphertext of length 3 as its input, then the length of the resulting ciphertext $l(v)$ is greater than 3, which breaks Equation (4.1). Thus, every multiply operation vertex should accept only ciphertexts of length 2 as its inputs, i.e., the term $(l^{new}(i) + x_i)$ of Equation (3.1) is always 3, a fixed value. As a result, we only need to minimize the term $k_r \sum_{i \in V} x_i$ of Equation (3.1). Since k_r is a fixed parameter, it is enough to minimize $\sum_{i \in V} x_i$, which is the number of relinearizations in the graph, to solve the relinearize problem under the constraints in Equation (4.1). Furthermore, because the maximum length of the ciphertext is 3, we relinearize a ciphertext at most once in each vertex, i.e., $x_i = \{0, 1\}$. □

Now, the situation is similar to the special case of the bootstrap problem under $L = 2$ discussed in [26]. In both situations, we have to minimize the number of vertices at which perform bootstrapping or relinearization. In the bootstrap problem, we have to perform *bootstrapping* one or more times in every *interesting-path*. In our case, we have to perform *relinearization* one or more times in every *interesting-path*. Note that in both cases, we perform bootstrapping or relinearization one **or more** times. This is because, for example in the relinearize problem case, even if the length of a ciphertext becomes 2 by performing relinearization in the *interesting-path*, a subsequent add operation in the *interesting-path* might have another input in which the ciphertext length is 3, which reverts it to a length 3 ciphertext. In this case, we need multiple relinearizations in one *interesting-path*. For example in Figure 4.1, let us cosider the *interesting-path* that ends vertex `(c)`. As $l(a) = 3$ and `(a)` outputs to multiply operation vertex `(b)`, we must perform a relinearization at `(a)`. In additon, because `(c)` receives a length 3 ciphertext as its input and outputs to multiply operation vertex `(d)`, a relinearization

Algorithm 1 Determining the minimum set of vertices to perform a relinearization

Require: $G = (V, E)$ ▷ data-flow graph representing the circuit
Ensure: V_{relin} ▷ set of vertices to relinearize

$G \leftarrow G \cup \{s, t\}$
for all $\{(u, v) \mid \forall e \in E, e = (u, v), \{u, v\} \in V, t(v) = \otimes\}$ **do**
    $E \leftarrow E \setminus e$
    $E \leftarrow E \cup \{(u, t)\}$
end for
for all $\{v \mid \forall v \in V, t(v) = \otimes\}$ **do**
    $E \leftarrow E \cup \{(s, v)\}$
end for
for all $\{v \mid \forall v \in V, t(v) =$`out`$\}$ **do**
    $E \leftarrow E \cup \{(v, t)\}$
end for
return $V_{relin} \leftarrow$ minimal (s, t) separator

at `(c)` is also required. Thus, the *interesting-path* includes relinearizations at `(a)` and `(c)`.

On a DAG $G = \{V, E\}$ with vertices $s, t \in V$, (s, t) separator W is a subset of V in which every path from s to t has at least one vertex in W. Here, we will construct a new DAG from the given data-flow graph G such that s is connected to the first vertices of every *interesting-path* to and the last vertices of each *interesting-path* are connected to t. Then, by calculating (s, t) separator, every *interesting-path* includes at least one separator vertex. Here, we can split the new graph into two subgraphs by the separator nodes; in one subgraph, the length of each output ciphertext is 2, and in the other, the length of each output ciphertext is 3. Relinearizations are performed at the nodes selected as separators. Such a graph is constructed by the procedure as shown in Algorithm 1.

In the algorithm, given a data-flow graph G, we firstly add vertices s and t to V. Then, we replace every edge (u, v), where $t(v) = \otimes$, to (u, t). Next, we add an edge from s to all of the multiply operation vertices. Lastly, we add an edge from all of the output vertices to t.

In the example as shown in Figure 4.2, the size of the minimum (s, t) separator is 5, which is a set of every multiply operation vertices for instance.

Lemma 2. The minimum set of vertices to perform a relinearization in the graph is obtained by using a minimal (s, t) separator problem.

Proof. By the graph construction in Algorithm 1, all of the (s, t) paths first pass through a multiply operation vertex, followed by passing through only add operation vertices. All of the *interesting-path* in the original graph are enumerated by all of the (s, t) paths in the new graph. Now, as we have

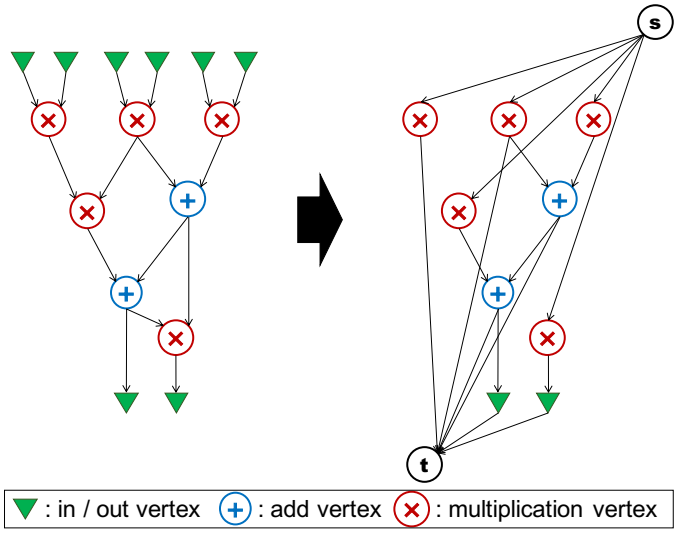

Figure 4.2: An example of graph conversion.

to pick up one or more vertices from each *interesting-path* as a vertex to perform relinearization, (s, t) separator is a set of vertices that satisfies our constraints. Thus, minimum (s, t) separators in the new graph coincides with the minimum set of vertices to perform a relinearization. □

Theorem 1. The relinearize problem restricted by the additional constraint Equation (4.1) is solved in $O(|V||E|)$ time.

Proof. From Lemma 1 and Lemma 2, performing Algorithm 1 solves the problem. The graph construction runs in $O(|V| + |E|)$ time. The number of vertices and edges in the new graph are still $O(|V|)$ and $O(|E|)$, respectively. The minimal (s, t) separator can be solved by the (s, t) min-cut, which runs is $O(|V||E|)$ time. Overall, this algorithm runs in $O(|V||E|)$ time and outputs the minimum number of vertices to perform a relinearization. □

Note that our proposed algorithm depends only on the data-flow graph, and not on the FHE parameters k_r, k_m. This feature enables us to divide the optimization process into circuit optimization and FHE parameter selection. For example, we can first optimize the circuit and the relinearization scheduling, followed by a selection of the FHE parameters. In contrast, the naïve ILP solution described in Section 3.2 requires k_r and k_m as given parameters. Thus, our method simplifies the optimization process.

4.2 The Optimal Case

Our proposed method is an approximation algorithm and thus not always optimal. However, in the case where $k_r \leq k_m$ is satisfied, the algorithm

outputs exactly the same objective value as the optimal one.

For simplicity, we define the variables R and M as

$$R = \sum_{i \in V} x_i, \quad M = \sum_{t(i)=\otimes, i \in V} (l^{new}(i) + x_i) \tag{4.2}$$

, where R is the total number of relinearizations in a given graph and M is the number of multiplications weighted by $l(i)$ in the graph. We note that $l(i) = l^{new}(i) + x_i$ holds. Then, we are able to rewrite our objective function of Equation (3.1) as follows.

$$k_r R + k_m M \tag{4.3}$$

Before we prove the optimality of our proposed method, we consider an extreme case where $k_m \to \infty$. In this case, the best strategy is to minimize M. In our method, as we discussed in the proof of Lemma 1, every multiplication only accepts ciphertexts of length 2 as its input, i.e., M is the minimum value. Thus, the proposed algorithm outputs the optimal result in this extreme case.

First, we define an operation $AddRelin(G, S, v)$.

Definition 4. $AddRelin(G, S, v)$ is an operation to output a feasible solution of the relineaize problem for a data-flow graph $G = \{V, E\}$, where S is also a feasible solution of the relinearize problem and $v \in V$ is a vertex in which both $l(v) \geq 4$ and $t(v) = \otimes$ holds. The operation outputs S' by the procedure as follows; As $l(v) \geq 4$ is satisfied, $l^{new}(p_1(v)) \geq 3$ or $l^{new}(p_2(v)) \geq 3$ is satisfied. Without loss of generality, let us assume $l^{new}(p_1(v)) \geq 3$. Let S' be the same as S but with the additional relinearization of the ciphertext at vertex $p_1(v)$, i.e., $x_{p_1(v)}$ increases by one and both $l^{new}(p_1(v))$ and $l(v)$ decreases by one. Though $l^{new}(p_1(v))$ and $l^{new}(v)$ decrease, they are still greater than or equal to 2 and satisfy all the constraints.

We denote $obj(S)$ as the objective value, i.e. the resulting value of Equation (3.1), of the relinearize problem for G under a feasible solution S. Now, by denoting $S' = AddRelin(G, S, v)$, we will compare $obj(S')$ against $obj(S)$. By adding relinearization at vertex $p_1(v)$, R increases by 1 and M decreases by 1. Inserting add operation vertices does not have any influence on the objective value. Thus, the following equation holds.

$$obj(S') = obj(S) + k_r - k_m \tag{4.4}$$

Furthermore, we note that $l(v)$ in S' is decreased by 1 compared to S.

Now, we are ready to prove the optimality of our algorithm under $k_r \leq k_m$.

Lemma 3. When $k_r < k_m$ is satisfied, $l(i) \leq 3$ for all $i \in V$ holds in the optimal solution of the relinearize problem.

Proof. We assume that in the optimal solution S_{OPT} of the relinearize problem on the graph G, there exists a vertex $u \in V$, which satisfies $l(u) \geq 4$. Here, we only consider u as a multiply operation vertex. This is because the length of the ciphertext at an add operation vertex is the same as one of its parent vertices. Thus, if there exists a vertex u with $l(u) \geq 4$ and $t(u) = \oplus$, there also exists its ancestor vertex $u' \in V$ that satisfies $l(u') \geq l(u)$ and $t(u') = \otimes$. We denote $S' = \mathit{AddRelin}(G, S_{OPT}, u)$. From Equation (4.4), we have $\mathit{obj}(S') = \mathit{obj}(S_{OPT}) + k_r - k_m < \mathit{obj}(S_{OPT})$. However, S_{OPT} is already the minimized solution. This contradiction proves that $l(i) \leq 3$ for all i under $k_r < k_m$. □

Lemma 4. When $k_r = k_m$ is satisfied, any optimal solution of the relinearize problem S_{OPT} can be converted into another optimal solution S'_{OPT} in which $l(i) \leq 3$ for all $i \in V$ holds.

Proof. Let S_{OPT} be the optimal solution of the relinearize problem in graph G and with the same discussion in the proof of Lemma 3 there exists a vertex $u \in V$ that satisfies $l(u) \geq 4$, $t(u) = \otimes$. We denote $S'_{OPT} = \mathit{AddRelin}(G, S_{OPT}, u)$. By applying $k_r = k_m$ to Equation (4.4), we have $\mathit{obj}(S'_{OPT}) = \mathit{obj}(S_{OPT})$. Thus, by comparing S'_{OPT} with S_{OPT}, we are able to reduce $l(u)$ by one without affecting $\mathit{obj}(S_{OPT})$. By repeating this procedure, we can replace all $l(u) \geq 4$ by $l(u) = 3$. □

Theorem 2. When $k_r \leq k_m$ is satisfied, there exists the optimal solution of the relinearize problem that satisfies $l(i) \leq 3$ for all $i \in V$.

Proof. A feasible solution always exists because a simple strategy that relinearizes ciphertext at every multiply operation vertex is always a feasible solution of the problem. Thus, the theorem follows Lemma 3 and Lemma 4. □

In summary, our method outputs the optimal solution under $k_r \leq k_m$ because even if we add a constraint Equation (4.1), we can obtain the same objective value from the optimal solution as one without the constraint.

4.3 Comparison of Methods

Now, we have the following four methods to the relinearize problem.

1. **Simple method** that relinearizes ciphertext at every multiply operation vertex.

2. **Naïve method** that optimally solves ILP described in Section 3.2.

Table 4.1: Comparison of methods to solve relinearize problem

	Applicable Circuits	Result	Complexity
Simple	Any	Heuristic	$O(\|V\|)$
Naïve	Any	Optimal	Exponential
Proposed	Any	Near-optimal	$O(\|V\|\|E\|)$
Chen's [10]	Limited	Optimal	$O(\|V\|^4)$

3. **Our approximation method** proposed in this paper.

4. **Chen's metthod [10]** for the special case in which every out-degree is one, mentioned in Section 2.2.

Table 4.1 shows the summary of the above four methods.

Note that in a general graph $O(|E|) = O(|V|^2)$ holds, but in a data-flow graph, every vertex's in-degree is at most 2, thus $O(|E|) = O(|V|)$ holds.

Chapter 5

Experimental Result

In this chapter, we discuss the practicality of our method based on the experiments with several circuits from [29]. Their characteristics are summarized in Table 5.1.

To obtain the optimal objective value, we used the Gurobi Optimizer[1] as an ILP solver. In order to obtain the result of our method, we have to calculate the (s, t) min-cut of the graph, which we implemented from scratch in C++. Both experiments were run on a desktop computer with Intel Core i7-4790 CPU @ 3.60 GHz and 16 GB of RAM running Ubuntu 18.04.1 LTS and gcc 7.3.0.

We applied both our method and the naïve method described in Section 3.2 to the circuits with various FHE parameters k_r, k_m. We calculated the ratio of the objective value $k_r R + k_m M$ obtained from our method to the optimal one from the naïve method to summarize in Table 5.2. We omitted the result under the condition $k_r < k_m$, in which the ratio should be always 1 because our method outputs the optimal solution as proved in Section 4.2. The experimental result shows that even if $k_r > k_m$, our approximation method is nearly optimal and all results are within 1% approximation error.

Table 5.3 compares the optimization run-time of our method and the naïve method. Each run-time was measured 20 times then averaged. Since the naïve method with the ILP solver depends on the FHE parameters k_r, k_m, we first measured with each parameter and took the average. Our method, on the other hand, does not depend on k_r, k_m and thus put only a single column for our method. The table shows that the proposed method runs 10 or more times faster than the naïve method. It can be easily estimated from the complexities of Table 4.1 that with a larger circuit, our method becomes more efficient than the naïve method.

In summary, our method decreases the optimization time from exponential to polynomial time, only with slight sacrifice of the objective value.

[1] http://www.gurobi.com/

Table 5.1: Circuits used for our experiment

Circuit File Name	$\|V\|$	$\|E\|$	# add	# mul
adder_64bit	759	887	265	494
AES-non-expanded	33,616	33,872	68,00	26,816
comparator_32bit_signed_lt	300	364	150	150
DES-non-expanded	30,313	30,441	18,124	12,189
md5	77,861	78,373	29,084	48,777
mult_32x32	12,374	12,438	5,926	6,448
sha-1	106,601	107,113	37,300	69,301
sha-256	236,112	236,624	90,825	145,287

Table 5.2: Ratio of the approximated objective value to the optimal value

Circuit, (k_r, k_m)	$(10,1)$	$(5,1)$	$(3,1)$	$(2,1)$	$(1,1)$
adder_64bit	1.0000	1.0000	1.0000	1.0000	1.0000
AES-non-expanded	1.0000	1.0000	1.0000	1.0000	1.0000
comparator_32bit_signed_lt	1.0000	1.0000	1.0000	1.0000	1.0000
DES-non-expanded	1.0000	1.0000	1.0000	1.0000	1.0000
md5	1.0058	1.0034	1.0016	1.0005	1.0000
mult_32x32	1.0002	1.0002	1.0001	1.0000	1.0000
sha-1	1.0087	1.0053	1.0025	1.0010	1.0000
sha-256	1.0080	1.0047	1.0022	1.0009	1.0000

Note: A value of 1.0000 indicates that the result of our method is optimal. A smaller value indicates better approximation.

Table 5.3: Optimization run-time

Circuit File, (k_r, k_m)	Naïves [ms]						Proposed	Speed Up[a]
	$(10,1)$	$(5,1)$	$(3,1)$	$(2,1)$	$(1,1)$	Avg.	[ms]	
adder_64bit	3.817	3.643	3.637	3.666	3.662	3.685	0.078	47.2
AES-non-expanded	892.729	835.516	802.444	694.378	691.067	783.227	72.980	10.7
comparator_32bit_signed_lt	2.262	2.174	2.173	2.176	2.173	2.192	0.072	30.4
DES-non-expanded	276.059	276.376	271.879	271.747	255.942	270.401	12.205	22.2
md5	1985.374	1387.921	1685.531	1298.942	1147.082	1500.970	77.300	19.4
mult_32x32	141.867	109.668	115.414	112.689	107.150	117.358	4.589	15.6
sha-1	3420.731	2381.683	2376.709	1837.956	1709.175	2345.251	105.012	22.3
sha-256	9686.367	5613.053	5481.531	5077.008	4630.607	6097.713	394.893	15.4

[a]Speed Up is the average optimization time of naive method divided by the optimization time of proposed method.

Chapter 6

Conclusion

In this paper, we proposed an approximation algorithm for the relinearize problem by adding one strong constraint and reducting to (s, t) min-cut problem. Though our proposed algorithm is approximated, the experimental results demonstrated that it is nearly optimal in practical circuits. Furthermore, we proved that under a specific FHE parameter, our algorithm outoputs optimal solution. Moreover, because our algorithm does not depend on the FHE parameters, users can adjust the parameters after optimizing the relinearization scheduling. However, selecting optimal FHE parameters is not an easy task and remains to be addressed in a future study.

Acknowledgement

Upon completion of this paper, I would like to thank Professor Yamana for continuing support of my research. Also, I would like to thank members in Secure Computation Group in the laboratory, especially Yu Ishimaki for wide-ranging discussion and sharing his knowledge of cryptography, Yoshiko Yasumura for constructive discussion and helping me with English writing, and Takuya Suzuki for checking this paper and providing valuable feedback. I would like to express my sincere gratitude to all the lab members.

Bibliography

[1] S. Angel, H. Chen, K. Laine, and S. Setty. Pir with compressed queries and amortized query processing. In *2018 IEEE Symposium on Security and Privacy (S&P)*, pages 962–979, May 2018.

[2] F. Benhamouda, T. Lepoint, C. Mathieu, and H. Zhou. Optimization of bootstrapping in circuits. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '17)*, pages 2423–2433, 2017.

[3] D. Boneh, C. Gentry, S. Halevi, F. Wang, and D. J. Wu. Private database queries using somewhat homomorphic encryption. In *International Conference on Applied Cryptography and Network Security*, volume 7954, LNCS, pages 102–118, 2013.

[4] R. Bost, R. A. Popa, S. Tu, and S. Goldwasser. Machine learning classification over encrypted data. In *22nd Annual Network and Distributed System Security Symposium, NDSS*, 2015.

[5] Z. Brakerski. Fully homomorphic encryption without modulus switching from classical gapsvp. In *Advances in Cryptology–CRYPTO 2012*, volume 7417, LNCS, pages 868–886, 2012.

[6] Z. Brakerski, C. Gentry, and V. Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, pages 309–325. ACM, 2012.

[7] S. Carpov, P. Aubry, and R. Sirdey. A multi-start heuristic for multiplicative depth minimization of boolean circuits. In *International Workshop on Combinatorial Algorithms, IWOCA 2017*, volume 10765, LNCS, pages 275–286, 2018.

[8] S. Carpov, P. Dubrulle, and R. Sirdey. Armadillo: a compilation chain for privacy preserving applications. In *Proceedings of the 3rd International Workshop on Security in Cloud Computing*, pages 13–19. ACM, 2015.

[9] A. Chatterjee and I. Sengupta. Translating algorithms to handle fully homomorphic encrypted data on the cloud. *IEEE Transactions on Cloud Computing*, 6(1):287–300, Jan 2015.

[10] H. Chen. Optimizing relinearization in circuits for homomorphic encryption. arXiv e-prints, Oct. 2017.

[11] H. Chen, I. Chillotti, and Y. Song. Improved bootstrapping for approximate homomorphic encryption. Cryptology ePrint Archive, Report 2018/1043, 2018.

[12] H. Chen and K. Han. Homomorphic lower digits removal and improved fhe bootstrapping. In *Advances in Cryptology – EUROCRYPT 2018*, volume 1082, LNCS, pages 315–337, 2018.

[13] J. H. Cheon, K. Han, and M. Hhan. Faster homomorphic discrete fourier transforms and improved fhe bootstrapping. Cryptology ePrint Archive, Report 2018/1073, 2018.

[14] J. H. Cheon, A. Kim, M. Kim, and Y. Song. Homomorphic encryption for arithmetic of approximate numbers. In *Advances in Cryptology – ASIACRYPT 2017*, volume 10624, LNCS, pages 409–437, 2017.

[15] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène. Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In *Advances in Cryptology–ASIACRYPT 2016*, volume 10031, LNCS, pages 3–33, 2016.

[16] E. Crockett, C. Peikert, and C. Sharp. Alchemy: A language and compiler for homomorphic encryption made easy. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018*, pages 1020–1037. ACM, 2018.

[17] R. Dathathri, O. Saarikivi, H. Chen, K. Laine, K. Lauter, S. Maleki, M. Musuvathi, and T. Mytkowicz. Chet: Compiler and runtime for homomorphic evaluation of tensor programs. In *Privacy Preserving Machine Learning, NeurIPS 2018 Workshop*, 2018.

[18] N. Dowlin, R. Gilad-Bachrach, K. Laine, K. Lauter, M. Naehrig, and J. Wernsing. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *Proceedings of International Conference on Machine Learning ICML*, volume 48, pages 201–210, 2016.

[19] J. Fan and F. Vercauteren. Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Report 2012/144, 2012.

[20] C. 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.

[21] C. Gentry, A. Sahai, and B. Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In *Advances in Cryptology–CRYPTO 2013*, volume 8042, LNCS, pages 75–92. 2013.

[22] T. Graepel, K. Lauter, and M. Naehrig. Ml confidential: Machine learning on encrypted data. In *International Conference on Information Security and Cryptology - ICISC 2012*, volume 7839 pf LNCS, pages 1–21, 2012.

[23] K. Lauter, A. López-Alt, and M. Naehrig. Private computation on encrypted genomic data. In *Progress in Cryptology - LATINCRYPT 2014*, volume 8895, LNCS, pages 3–27, 2015.

[24] T. Lepoint and P. Paillier. On the minimal number of bootstrappings in homomorphic circuits. In *Financial Cryptography and Data Security: FC 2013 Workshops*, volume 7862, LNCS, pages 189–200, 2013.

[25] J. B. Orlin. Max flows in o(nm) time, or better. In *Proceedings of the Forty-fifth Annual ACM Symposium on Theory of Computing*, STOC '13, pages 765–774. ACM, 2013.

[26] M. Paindavoine and B. Vialla. Minimizing the number of bootstrappings in fully homomorphic encryption. In *International Conference on Selected Areas in Cryptography*, volume 9566, LNCS, pages 25–43, 2015.

[27] Palisade lattice cryptography library. `https://git.njit.edu/palisade/PALISADE`, Nov. 2018.

[28] Simple Encrypted Arithmetic Library (release 3.1.0). `https://github.com/Microsoft/SEAL`, Dec. 2018. Microsoft Research, Redmond, WA.

[29] N. P. Smart and S. Tillich. Circuits of basic functions suitable for mpc and fhe. `https://homes.esat.kuleuven.be/~nsmart/MPC/`.

Publications

[1] **Hiroki Sato**, Akira Umayabara, Yu Ishimaki and Hayato Yamana. Poster: Loop Circuit Optimization with Bootstrapping over Fully Homomorphic Encryption, In *Proceedings of 2nd IEEE European Symposium on Security and Privacy (Euro S&P 2017)*, April, 2017.

[2] 馬屋原昂, **佐藤宏樹**, 石巻優, 今林広樹, 山名早人. FCMalloc: 完全準同型暗号の高速化に向けたメモリアロケータ, 第 *141* 回システムソフトウェアとオペレーティング・システム研究会, 情処研報, Vol. 2017-OS-141, No. 6, July, 2017.

[3] Arisa Tajima, **Hiroki Sato**, and Hayato Yamana. Privacy-Preserving Join Processing over outsourced private datasets with Fully Homomorphic Encryption and Bloom Filters, 第 *10* 回データ工学と情報マネジメントに関するフォーラム *2018 (DEIM2018)*, F7-2, March, 2018.

[4] **佐藤宏樹**, 馬屋原昂, 石巻優, 山名早人. 完全準同型暗号による秘密計算回路のループ最適化と最近傍法への適用, 日本データベース学会和文論文誌 *DBSJ Japanese Journal*, Vol.16-J, Article No.12, March, 2018.

[5] Arisa Tajima, **Hiroki Sato**, and Hayato Yamana. Outsourced Private Set Intersection Cardinality with Fully Homomorphic Encryption. *2018 6th International Conference on Multimedia Computing and Systems (ICMCS)*, May, 2018.

[6] **佐藤宏樹**, 石巻優, 山名早人. 完全準同型暗号における bootstrap problem 及び relinearize problem の高速化, 第 *11* 回データ工学と情報マネジメントに関するフォーラム *2019 (DEIM2019)*, March, 2018.

[7] 鈴木拓也, **佐藤宏樹**, 山名早人. Knights Landing 上での準同型暗号データ高速処理手法の検討, 第 *11* 回データ工学と情報マネジメントに関するフォーラム *2019 (DEIM2019)*, March, 2018.

Appendix A

How much we can speed-up by the Relinearize Problem

Here, for reference, we show how much computation time can be reduced by solving the relinearize problem. Now, we have four solutions to the relinearize problem listed in Section 4.3. We examined the estimated total computation time $k_r R + k_m M$, the objective value of the ILP, for each method. Of course, the **Simple** solution outputs the longest computation time, the **Naïve** outputs the shortest and the optimal, and the **Proposed** outputs a result between the two. We summarized the ratio of the **Naïve** to the **Simple** and the **Proposed** to the **Simple** in Table A.1. Though solving the relinearize problem does not reduce the computation time in some cases, it generally reduces a certain amount of the computation time. Here, we cannot apply Chen's method [10] because this method has a strong limitation to the applicable circuit described in Section 2.2.

Table A.1: Ratio of the objective values. The upper row is the ratio of the result from the optimal strategy to the simple strategy, and the lower row is the ratio of the result from our approximation method to the simple strategy.

Circuit	(k_r, k_m)	$(10,1)$	$(5,1)$	$(3,1)$	$(2,1)$	$(1,1)$	$(1,2)$	$(1,3)$	$(1,5)$	$(1,10)$
adder_64bit	Optimal	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
	Approx.	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
AES-non-expanded	Optimal	0.6570	0.7213	0.7771	0.8216	0.8885	0.9363	0.9554	0.9721	0.9856
	Approx.	0.6570	0.7213	0.7771	0.8216	0.8885	0.9363	0.9554	0.9721	0.9856
comparator_32bit_signed_lt	Optimal	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
	Approx.	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
DES-non-expanded	Optimal	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
	Approx.	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
md5	Optimal	0.9892	0.9926	0.9952	0.9969	0.9984	0.9991	0.9994	0.9996	0.9998
	Approx.	0.9950	0.9960	0.9968	0.9974	0.9984	0.9991	0.9994	0.9996	0.9998
mult_32x32	Optimal	0.9995	0.9996	0.9997	0.9998	0.9999	1.0000	1.0000	1.0000	1.0000
	Approx.	0.9997	0.9998	0.9998	0.9999	0.9999	1.0000	1.0000	1.0000	1.0000
sha-1	Optimal	0.9824	0.9874	0.9916	0.9943	0.9971	0.9983	0.9988	0.9993	0.9996
	Approx.	0.9910	0.9927	0.9941	0.9953	0.9971	0.9983	0.9988	0.9993	0.9996
sha-256	Optimal	0.9699	0.9773	0.9833	0.9875	0.9927	0.9958	0.9971	0.9982	0.9991
	Approx.	0.9776	0.9818	0.9855	0.9884	0.9927	0.9958	0.9971	0.9982	0.9991