Poster: Loop Circuit Optimization with Bootstrapping over Fully Homomorphic Encryption

Hiroki Sato, Akira Umayabara, Yu Ishimaki, Hayato Yamana
Waseda University
Tokyo, Japan
Email: {hsato, uma, yuishi, yamana}@yama.info.waseda.ac.jp

***Abstract* — Fully Homomorphic Encryption (FHE) enables us to compute arbitrary circuits over encrypted data without decryption. To evaluate a complex circuit, a time-consuming operation called *bootstrapping* is required. Reducing the number of bootstrapping operations leads directly to reducing the entire computation time of the circuit. An optimization problem which minimizes the number of bootstrapping operations is called *bootstrap problem*, which is NP-complete. In previous work to tackle the problem, objective circuits must be represented as a directed acyclic graph. Thus, the previous methods cannot handle loop-carried dependencies for the circuits containing any loop, which results in no optimization over iterations. In this paper, we propose a method to decide a near-optimal placement of bootstrapping operations in a loop circuit by adopting *loop unrolling* technique. Compared to a naïve method, our method successfully reduced the number of bootstrapping operations per loop iteration up to 50 percent for a toy circuit, and up to 63 percent for a nearest neighbor classification circuit.**

Keywords—FHE, bootstrapping, loop unrolling

I. Introduction

Fully Homomorphic Encryption (FHE) scheme is a public key cryptography where we can perform arbitrary computation over ciphertexts without decryption [1]. FHE makes it possible to securely delegate computation and/or analysis on confidential data to an untrusted third-party like cloud computing.

In 2009, Gentry first proposed the mechanism of FHE. After that, many follow up works have appeared. All existing FHE schemes have a same property; every ciphertext includes some noise which grows with every arithmetic operation. To get a correct decryption result, an operation called bootstrapping to reset the noise is required. However, the bootstrapping consumes huge computation time. Especially in a loop, huge number of bootstrapping operations is indispensable. Given a circuit to compute, minimizing the number of bootstrapping operations directly results in reducing the entire computation time. The minimization problem is called *bootstrap problem*.

In previous work to tackle the problem, researchers have considered a circuit represented as a directed acyclic graph (DAG) that cannot handle loop-carried dependencies for a loop circuit, which results in no optimization over iterations. To remove the restriction, we propose a new method adopting *loop unrolling*. The main contribution of this paper is to show the effectiveness of *loop unrolling* in the bootstrap problem.

This work was supported by JST CREST Grant Number JPMJCR1503, Japan.

II. Background

A. Leveled Fully Homomorphic Encryption (leveled FHE)

In this paper, we focus only on leveled FHE. The following description is a simplified model of leveled FHE. A parameter *level* $L \in \mathbb{Z}(> 0)$ is decided along with key generation, and each ciphertext has a parameter *noise-level* ℓ. Just after the encryption, $\ell = L$. The evaluation of XOR (or addition) gate to two ciphertexts whose *noise-level*s are ℓ_1 and ℓ_2 yields a ciphertext whose *noise-level* is $\min(\ell_1, \ell_2)$. The Evaluation of AND (or multiplication) gate yields a ciphertext whose *noise-level* is $\min(\ell_1, \ell_2) - 1$. While a ciphertext keeps $\ell > 0$, its decryption is correct. When *noise-level* $\ell \leq 0$, the correctness of the decryption result is not guaranteed. By executing bootstrapping, the *noise-level* of ciphertext is reset to $N(> 0)$, so that we can continue its computation over the ciphertext.

B. Bootstrap Problem

Since the bootstrapping is time-consuming, minimizing the number of bootstrapping operations in a circuit directly results in reducing the entire computation time. In previous work, the bootstrap problem is proved to be NP-complete [2], and its optimization result can be obtained by converting the problem to integer liner programming (ILP) [3]. In the previous works, a data-flow graph should be represented as a DAG, which does not contain any loop. Thus, previous methods can optimize only one iteration of a loop circuit, which results in the lack of optimization over iterations.

III. Baseline and Proposed Method

As a baseline, we consider following two methods directly adopting the previous work proposed by Paindavoine et al. [3].

- Optimal Method: A loop is unfolded completely to have its whole DAG over iterations followed by adopting the previous work [3]. Note that, it is impossible to unfold whole iterations because we usually do not know the total iteration number statically. Only to confirm the optimal placement of bootstrapping operations, we consider this method.
- Naïve Method: The method of previous work [3] is applied to one iteration of a loop, and bootstrapping operations are additionally placed at the end of each iteration because it becomes impossible to predict the required *level* without the placement.

We extend the Paindavoine's method [3] to handle loops. Here, we assume that a circuit satisfies the following two constraints: 1) there are no loop-carried dependencies whose distance is more than 1, and 2) there are no branches in the loop. In these constraints, we propose a method to decide a near-optimal placement of bootstrap operations. Our method also keeps its graph size small to shorten the optimization time.

A. Loop Unrolling

Our method manages "loop-carried noise dependencies" by adding the constraints to ILP which expresses data dependencies among iterations. In addition, a given loop is unfolded a few times. We will show from the experimental result that unfolding only a few times is enough to get near-optimal result.

B. Level Parameter Selection

If an initial *level* is set high, the computation cost, such as multiplication and bootstrapping, increases though the required number of bootstrapping operations becomes small. It is not simple to decide the best *level*, so that we estimate entire computation time t_{total} of formula (1) by varying *level* parameters for choosing the best ones to shorten t_{total}. Here, t_{bs} denotes computation time of a bootstrapping operation, t_{mul} is multiplication time, n_{bs} and n_{mul} denote required number of bootstrapping operations per iteration and that of multiplication.

$$t_{total} = t_{bs} \times n_{bs} + t_{mul} \times n_{mul} \qquad (1)$$

IV. Experiments and Discussion

We conducted two experiments to evaluate our method using HElib[4]. First, we applied baseline and proposed methods for a toy circuit shown in Fig.1. The result is shown in TABLE I and TABLE II. Second, as a real circuit, we constructed a nearest neighbor classification circuit and applied our method. The result is shown in TABLE III and TABLE IV. As shown in the tables, we varied the parameters to find the optimal ones.

The result shows that by unrolling just a few times, at most 8, our method outputs near-optimal number of bootstrapping operations, i.e., only 1.0 ~ 1.2 times larger than the optimal. This result shows that small number of loop unrolling is enough to have a near-optimal solution.

V. Conclusion

In this paper, we proposed a new bootstrapping optimization method over loop circuits and showed the effectiveness. Our future work includes the optimization both for arbitrary circuits including various kinds of data dependencies and for other operations beyond bootstrapping like re-linearization.

References

[1] C. Gentry, "A Fully Homomorphic Encryption Scheme", Ph.D. Thesis, Stanford University, 2009.

[2] T. Lepoint, P. Paillier, "On the Minimal Number of Bootstrappings in Homomorphic Circuits", Financial Cryptography and Data Security, LNCS, vol. 7862, pp.189-200, 2013.

[3] M. Paindavoine, B. Vialla, "Minimizing the Number of Bootstrappings in Fully Homomorphic Encryption", Selected Areas in Cryptography, LNCS, vol. 9566, pp. 25-43, 2015.

[4] HElib. http://shaih.github.io/HElib/index.html accessed in 2017-2-15.

TABLE I. Number of Bootstraps per Iteration (Toy Circuit)

Parameters (L, N)		(18,7)	(20,9)	(22,11)	(24,13)	(26,15)	(28,17)
(a) Optimal Method (50 iterations)		0.92	0.82~ 0.92[a]	0.68	0.60	0.52	0.44
(b) Naïve Method		2.00	2.00	2.00	2.00	2.00	2.00
(c) Proposed Method (# of iterations to optimize)	1	2.00	2.00	2.00	2.00	2.00	2.00
	2	1.50	**1.00**	1.00	1.00	1.00	1.00
	3	**1.00**	1.00	1.00	1.00	0.67	0.67
	4	1.25	1.00	**0.75**	0.75	0.75	**0.50**
	5	1.20	1.00	0.80	0.80	0.60	0.60
	6	1.00	1.00	0.83	0.83	0.67	0.67
	7	1.14	1.00	0.86	**0.71**	**0.57**	0.57
	8	1.13	1.00	0.75	0.75	0.63	0.50
	9	1.00	1.00	0.78	0.78	0.67	0.56
	10	1.10	1.00	0.80	0.80	0.60	0.60
(c) / (a)		1.09	1.22~ 1.09	1.10	1.18	1.10	1.14
(c) / (b)		0.50	0.50	0.38	0.36	0.29	0.25

[a.] Unable to get optimal solution because of memory shortage.

TABLE II. Optimization Result (Toy Circuit)

Parameters (L, N)	(18,7)	(20,9)	(22,11)	(24,13)	(26,15)	(28,17)
Security [bit]	180.7	146.6	120.7	107.4	90.9	80.7
Multiplication Time [sec.]	0.143	0.162	0.158	0.165	0.168	0.191
Bootstrap Time [sec.]	77.27	82.89	85.00	86.29	86.50	90.38
Result of TABLE I. [# of bootstrap / iter.]	1.00	1.00	0.75	0.71	0.57	0.50
Estimated Circuit-Computation-Time [sec. / iter.]	78.13	83.86	64.70	62.26	50.31	46.34

TABLE III. Number of Bootstraps per Iteration (NN Circuit)

Parameters (L, N)		(22,11)	(24,13)	(26,15)	(28,17)	(30,19)
(a) Optimal Method (18 iterations)		1.11	1.06~ 0.89	0.89	0.67	0.56
(b) Naïve Method		2.00	2.00	2.00	2.00	2.00
(c) Proposed Method (# of iterations to optimize)	1	2.00	2.00	2.00	2.00	2.00
	2	1.50	1.5	**1.00**	1.00	1.00
	3	1.33	1.33	1.00	1.00	1.00
	4	**1.25**	1.25	1.00	**0.75**	0.75
	5	1.40	1.20	1.00	0.80	0.80
	6	1.33	1.17	1.00	0.83	**0.67**
	7	1.29	1.14	1.00	0.86	0.71
	8	1.25	**1.13**	1.00	0.75	0.75
(c) / (a)		1.13	1.27~ 1.07	1.12	1.12	1.20
(c) / (b)		0.63	0.57	0.50	0.38	0.34

TABLE IV. Optimization Result (NN Circuit)

Parameters (L, N)	(22, 11)	(24,13)	(26, 15)	(28,17)	(30,19)
Security [bit]	120.7	107.4	90.9	80.7	65.4
Multiplication Time [sec.]	0.158	0.165	0.168	0.191	0.190
Bootstrap Time [sec.]	85.00	86.29	86.50	90.38	93.25
Result of TABLE I. [# of bootstrap / iter.]	1.25	1.13	1.00	0.75	0.67
Estimated Circuit-Computation-Time [sec. / iter.]	107.99	99.32	88.35	69.89	64.57

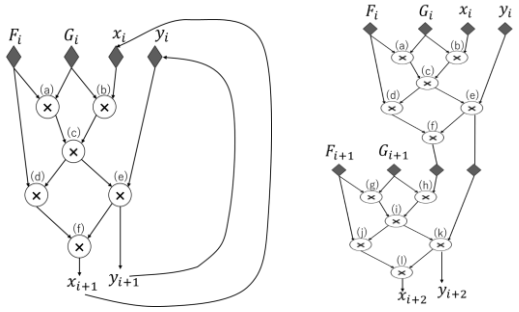

Fig. 1. A Toy Circuit (left) and its Unrolled Circuit Example (right)