2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC) | 979-8-3503-9354-5/24/$31.00 ©2024 IEEE | DOI: 10.1109/ASP-DAC58780.2024.10473831

Logic Locking over TFHE for Securing User Data and Algorithms

Kohei Suemitsu Kotaro Matsuoka Takashi Sato Masanori Hashimoto
Dept. Communications and Computer Engineering, Kyoto University

***Abstract*—This paper proposes the application of logic locking over TFHE to protect both user data and algorithms, such as input user data and models in machine learning inference applications. With the proposed secure computation protocol, algorithm evaluation can be performed distributively on honest-but-curious user computers while keeping the algorithm secure. To achieve this, we combine conventional logic locking for untrusted foundries with TFHE to enable secure computation. By encrypting the logic locking key using TFHE, the key is secured with the degree of TFHE. We implemented the proposed secure protocols for combinational logic neural networks and decision trees using LUT-based obfuscation. Regarding the security analysis, we subjected them to the SAT attack and evaluated their resistance based on the execution time. We successfully configured the proposed secure protocol to be resistant to the SAT attack in all machine learning benchmarks. Also, the experimental result shows that the proposed secure computation involved almost no TFHE runtime overhead in a test case with thousands of gates.**

I. Introduction

Data and algorithms have substantial value in real-world applications such as medical and commercial sectors. Consider a scenario involving a medical enterprise and an algorithm-driven company. The medical enterprise owns a dataset, while the company has a data mining algorithm. The company prefers not to disclose its proprietary algorithm, and the medical enterprise is reluctant to share its dataset due to its potential sensitivity. This impasse can be resolved through a *private function evaluation* (PFE) protocol, allowing the medical enterprise to receive the result of the data mining algorithm running privately on its dataset.

In two-party PFE problem, Party P_1 holds a private function f and optionally, a private input x_1. Conversely, Party P_2 holds a different private input, x_2. The goal is for both parties to independently calculate the value of $f(x_1, x_2)$ without third-party involvement. Subsequently, one or both parties can obtain the value of $f(x_1, x_2)$, with no capacity to infer additional information beyond their specified outputs. PFE, a special case of secure computation, diverges significantly from standard *secure function evaluation* (SFE) as the function f is public in SFE, but confidential in PFE. Existing literature on PFE protocols, on the other hand, often restricts permissible functions to specific types. We focus on functions implemented by arbitrary circuits and discuss general-purpose PFE protocols. A common PFE protocol approach utilizes Universal Circuits (UC). UC refers to a sequence of circuits $U = \{U_n\}_{n\in\mathbb{N}}$, each of which can take as input a circuit C of size n and a valid input x, and output $C(x) \leftarrow U_n(C, x)$. A limitation of UC-based PFE protocols is that a Boolean UC has an optimal size $|U_n| = \Theta(nlogn)$ [1]. Thus, when the circuit size is large, the size increase resulting from UC usage renders UC-based PFE impractical.

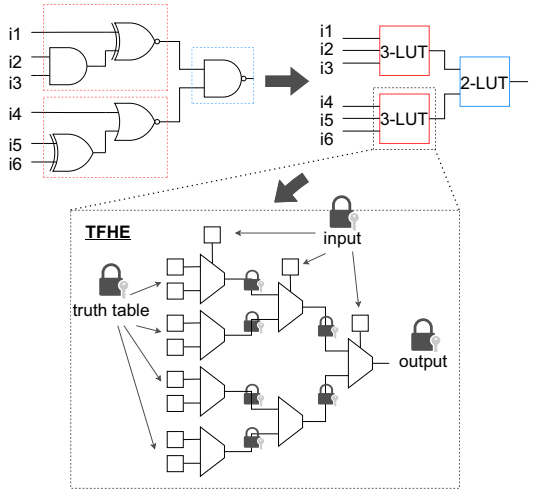


Fig. 1: Conceptual overview of the proposed LUT-based logic locking over TFHE. The inputs i1 to i6 and the logic locking keys, which are truth table values, are encrypted. The encrypted output value is obtained without decryption.

In this paper, we propose a general-purpose secure computation approach combining logic locking with conventional secure computation methods like Fast Fully Homomorphic Encryption over the Torus (TFHE) [2] and garbled circuit [3]. Logic locking, initially developed as a countermeasure for untrusted IC foundries [4], is used to obscure the algorithm. Earlier literature lacked formalism delineating the precise security level a logic locking scheme should provide. However, [5] shows that several programmable logic-based methods [6]–[9] can be modeled within a UC security framework, indicating logic locking could facilitate a more efficient PFE implementation compared to UC.

Fig. 1 illustrates our proposed secure computation method, which protects both the algorithm (e.g., machine learning models) and user data by using logic locking to safeguard the algorithm and evaluating it in an encrypted form using TFHE. Notably, the logic locking key, represented as a binary sequence, is encrypted with the model owner's private key, providing mathematical security assurance. This is a stark departure from conventional logic locking methods used for untrusted foundries. Here, it should be mentioned that our main idea could be implemented using garbled circuits, while we assume TFHE as the underlying technique in this paper. Also, it is important to mention that the proposed method is a general-purpose secure computation approach including sequential circuits, while its significance is discussed with combinational circuit-based machine learning in this paper.

Meanwhile, as logic locking attacks have been thoroughly studied, methods such as SAT attack and its extensions [10],

979-8-3503-9354-5/24/$31.00 ©2024 IEEE

[11] could be applied to the proposed secure computation. Thus, resistance to these attacks is crucial. This work focuses on LUT-based logic obfuscation, as suggested by [12], [13], stating that the number of LUTs and the replacement strategy for LUTs can effectively enhance attack resistance.

Contributions: In brief, our contributions are as follows:

1) We introduce logic locking over TFHE as a novel general-purpose secure computation approach that protects both the algorithm and user data. The proposed protocol facilitates two-party *private function evaluation* (PFE) [14].
2) We suggest encrypting the logic locking key with TFHE. Traditional logic locking relies on tamper-proof chip protection to shield secret key values from attackers. Yet, even with this safeguard, a malicious third party with access to the locked circuit's oracle could acquire the correct key. Encrypting the logic locking key renders it theoretically inaccessible to any third party, even those with physical system access.
3) Our experiments confirmed that obfuscated algorithms withstanding the SAT attack could be achieved with an acceptable overhead concerning TFHE execution time.
4) We conducted a security analysis under the honest-but-curious (HBC) model, wherein participants comply with the protocol and unauthorized information acquisition is prevented, substantiating our protocol's security. Furthermore, we explored countermeasures for scenarios extending beyond the HBC model and carried out experiments to assess their efficacy.

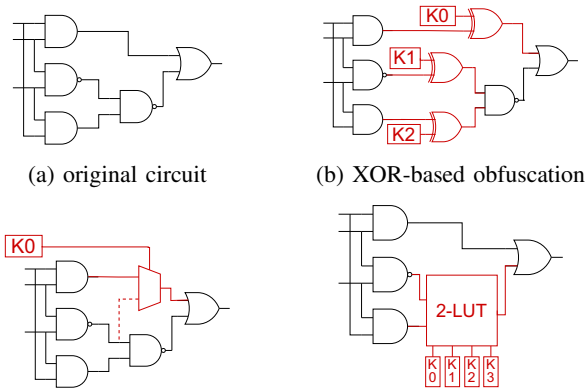
The rest of this paper is structured as follows. Section II provides background information on TFHE, logic locking, and SAT attacks for deobfuscation. Section III details the proposed secure computation protocol, encompassing the protocol definition, logic locking procedure, and security analysis. Section IV presents the experimental results. Finally, Section V concludes the paper.

II. Background

A. Fast Fully Homomorphic Encryption over the Torus: TFHE

Homomorphic Encryption (HE) is a unique form of encryption allowing computations on encrypted data without needing decryption [15]. Depending on the types of functions allowed for computation, HE can be classified into various types. Notably, Fully Homomorphic Encryption (FHE) supports the evaluation of arbitrary functions through a process known as bootstrapping, proposed in Gentry's seminal work [16]. This technique is used to mitigate the noise produced within the ciphertexts during computations.

TFHE [2], [17] is one kind of FHE and can evaluate arbitrary Boolean circuit over encrypted ciphertexts. One key advantage of TFHE is its swift bootstrapping time relative to other Fully Homomorphic Encryption (FHE) schemes. For example, TFHE's bootstrapping time approximates 10 ms, whereas schemes like BGV with HElib can take minutes [18]. Owing to the efficient bootstrapping process per logical operation, the evaluation time of logic functions in TFHE, frequently represented as combinational logic circuits, is directly proportional to the number of logic operations (gates). This feature is highly advantageous for efficient computations.



(a) original circuit (b) XOR-based obfuscation

(c) MUX-based obfuscation (d) LUT-based obfuscation

Fig. 2: Logic locking.

B. Logic Locking and SAT attack

Logic locking [4] is a technique used to protect Intellectual Property (IP) from untrusted foundries. In this approach, additional logic gates, known as key gates, are inserted into the original function, as depicted in Fig. 2. These key gates are controlled by a logic locking key, which is stored in on-chip memory. The key gates employed in logic locking often include XOR/XNOR gates [4], [19]–[21], MUX gates [20], [22]–[24], or look-up tables [12], [25], [26]. Only when the correct logic locking key is applied, does the Integrated Circuit (IC) function correctly. Therefore, attackers attempt to steal the logic locking key to gain access to the function.

The SAT attack [10] is a potent technique for retrieving the logic locking key. It works by iteratively eliminating incorrect keys using *Distinguished Input Patterns* (DIPs) – specific inputs that yield different outputs for different keys. Through a series of iterations, all incorrect keys can be eliminated within seconds to minutes, even for large circuits. After identifying each DIP, a new constraint is added to the SAT solver's satisfiability problem. This continues until the solver cannot find a satisfying assignment. At this stage, any key that meets all previous DIPs is considered the correct key for the obfuscated circuit.

C. LUT-based Obfuscation

LUT-based obfuscation employs Look-Up Tables (LUTs) as key gates in logic locking [8], [12], [13]. LUTs are commonly used in FPGAs to implement arbitrary combinational logic functions, with their truth-table values provided externally. In LUT-based obfuscation, these truth-table values serve as the logic locking key.

In this work, we utilize Look-Up Tables (LUTs) for obfuscation due to several reasons. Firstly, LUT-based logic locking demonstrates the potential to be modeled within a Universal Circuit (UC) security framework as suggested in [5]. Secondly, as mentiond in [12], compared with other techniques (e.g., XOR and MUX), LUTs can expand the truth-table size exponentially with increased inputs. Furthermore, LUTs, in replacing the original logic gates and locking the Integrated Circuit (IC), preserve only a portion of the original design, whereas XOR gate locking maintains the entire original design while introducing key gates. Thirdly, in conventional LUT-based obfuscation, logic locking for supply chain attacks

with large LUTs results in significant design overhead in power, performance, and area during silicon implementation, as reported in [6]. However, the proposed method presents the algorithm using logical expressions and evaluates the logic circuit virtually using TFHE on computers. Therefore, considerations related to silicon-related overheads, such as power, and area, are deemed unnecessary in this context.

III. Proposed Secure Computation Protocol: Logic Locking over TFHE

A. Definitions

The principal protocol considered in this paper is two-party *private function evaluation* (PFE) [14]. Subsequent sections provide thorough explanations of the protocol.

Definition of data owner and model owner: We define the "data owner" as a user possessing their own computing resources and input data, while the "model owner" refers to an individual who possesses the protected algorithm (e.g., a machine learning model). In the context of inference services using machine learning, the model owner typically holds the machine learning model that requires protection. Both the data and model owners aim to perform their respective operations with minimal disclosure of information about each other.

B. Assumptions for The Proposed Protocol.

we consider the honest-but-curious (HBC) model, assuming that the attacker is either on the side of the data holder or the algorithm holder. This model places limitations on their behavior, implying that they are expected to act honestly but may still be curious about the other party's information.

C. Proposed Protocol

The protocol of the proposed logic locking over TFHE consists of eight phases, as depicted in Fig. 3, and each phase is explained in the following.

1) **keyGen**: the model owner generates a set of secret key sk and public key pk.
2) **Transmit**: the model owner transmits pk generated in step 1 to the data owner.
3) **Obfuscate**: the model owner obfuscates the function f by LUT-based obfuscation.
4) **Encryption**: the model owner encrypts the truth table of LUTs with the secret key sk and transmits the obfuscated function, which consists of the encrypted truth tables and a circuit netlist, to the data owner. The data owner also encrypts the input data with the public key pk
5) **Evaluation**: the data owner evaluates the encrypted data by evaluating the obfuscated function using the TFHE ciphertexts and the circuit netlist.
6) **Mask**: The result obtained in step 5 is still encrypted, and hence it must be decrypted by the model owner who possesses the secret key sk. However, if it is sent to the model owner directly, the evaluation result will be leaked. Therefore, the encrypted result is masked with, for example, one-time pad.
7) **Decryption**: the model owner decrypts the masked encrypted result using his secret key sk and transmits the masked result to the data owner.

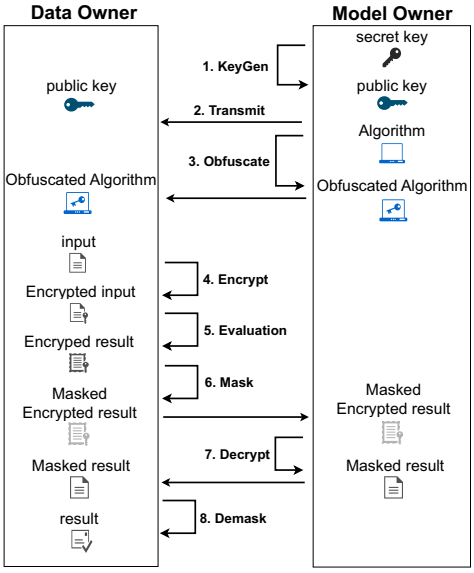


Fig. 3: Proposed protocol.

8) **Demask**: the data owner demasks the masked result in step 7 and obtains the result.

D. Procedure of Logic Locking over TFHE

In this section, we explain the procedure of logic locking over TFHE using Fig. 1.

1) Derive a logical expression of the algorithm to be kept secure.
2) Map the algorithm, formulated as a logical expression, to a LUT-based structure with function determined by LUT truth table values. Leveraging established techniques from FPGA logic mapping and minimization can facilitate this process.
3) Encrypt the truth table values using the secret key.

Experimentally, circuits solely composed of LUTs are often SAT attack-resilient. Despite the multiplexer overhead introduced by LUT-based logic mapping, we mitigate this by converting some LUTs into primitive logic gates in the mapped circuit, reducing primitive logic gate count and improving TFHE evaluation time. This partial LUT mapping approach is evaluated in Section IV.

E. Security Analysis

In this section, we analyze the security of the proposed logic locking scheme over TFHE. We begin by discussing the assumptions made in the security analysis. Following that, we describe various potential attack methods that could be employed against the logic locking scheme over TFHE.

1) Assumptions of security analysis: We assume that the data owner, having physical access to the computational resource and the locked algorithm, cannot decipher the function due to the model owner's exclusive access to the secret key. The data owner masks the encrypted result using techniques such as a one-time pad, requiring decryption by the model owner, who is unaware of the decrypted result. Under the HBC model, the security of encrypted data, including input, logic locking key, and all results, depends on the difficulty

of decrypting ciphertexts in TFHE's Chosen-Plaintext Attack (CPA) setting. The robustness of LWE-based FHE schemes, founded on established hardness assumptions, safeguards the encrypted data.

2) Possible attack methods: We infer that only the model owner knows the decrypted result, and potential attacks are confined to methods excluding decryption of the logic locking key. In the HBC model, the data owner may query to gain the model owner's algorithm. Thus, we consider the SAT attack [10]—noted as the most potent against logic locking methods in Section II-B—as a possible method. If the SAT attack demands considerable execution time, it will likely fail.

In addition to the above, we should also consider the oracle-less attack. A notable example of an oracle-less attack, specifically dedicated to LUT-based obfuscation, is found in [27]. However, [27] assumes that the original circuits are synthesized for ordinary logic gates, and that some of the gates are replaced with LUTs afterwards. On the other hand, the proposed method starts with the circuit fully consisting of LUTs. Therefore, the attack efficiency is unclear. Moreover, reproducing this attack poses challenges due to the lack of details about which learning model should be used to predict the truth table. Addressing this issue is one of our future work.

3) Remark: When the function is obfuscated, it conceals the type of machine learning model used, reducing the likelihood of successful model extraction attacks. Also, numerous queries might alert the model owner to a potential attack.

4) Attack beyond HBC model and potential countermeasure: Our protocol may be vulnerable to malicious data owners beyond the HBC model, who could send the encrypted logic locking key to the model owner for decryption. Several countermeasures can be considered:

i) For every inference, re-synthesize the algorithm to generate a different LUT-based circuit and change the key pair. This makes any decrypted logic locking key fragments from previous circuits invalid. Although key generation and circuit re-synthesis are costly for the model owner, they aren't always necessary as obfuscated circuits are highly resistant to SAT attacks. Additionally, the small output size of many inference models reduces the decryption size of the logic locking key per malicious inference query, further decreasing the required update frequency. This update frequency will be experimentally evaluated in the next section. The feasibility of circuit re-synthesis for this purpose will be addressed in future work.

ii) Enable verifiable computation in TFHE. This would allow the model owner to verify whether the masked encrypted result is malicious. We anticipate TFHE will support this feature. Alternatively, garbled circuits could replace TFHE, as they already support verifiable computation.

IV. Experiments and Results

In this section, we conduct experimental evaluations of the proposed logic locking over TFHE, focusing on both security and performance aspects. For security evaluation, we measure the execution time required to infer the correct key using SAT attack as a metric, as discussed in Section III-E. Regarding performance evaluation, we measure the execution time of TFHE and investigate the performance overhead introduced by the logic locking mechanism.

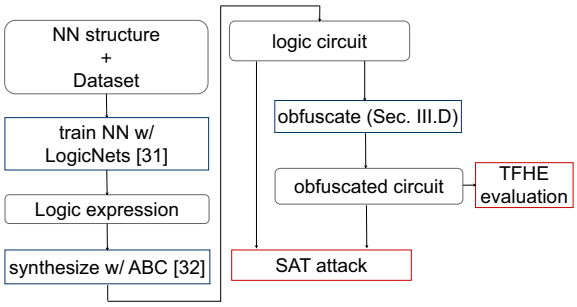


Fig. 4: Experiment flow.

A. Experimental Setup

We conducted evaluations on a system with an Intel Core i9-10850K 10-core CPU @ 3.60GHz, 32GB RAM, and Ubuntu 20.04.2 LTS.

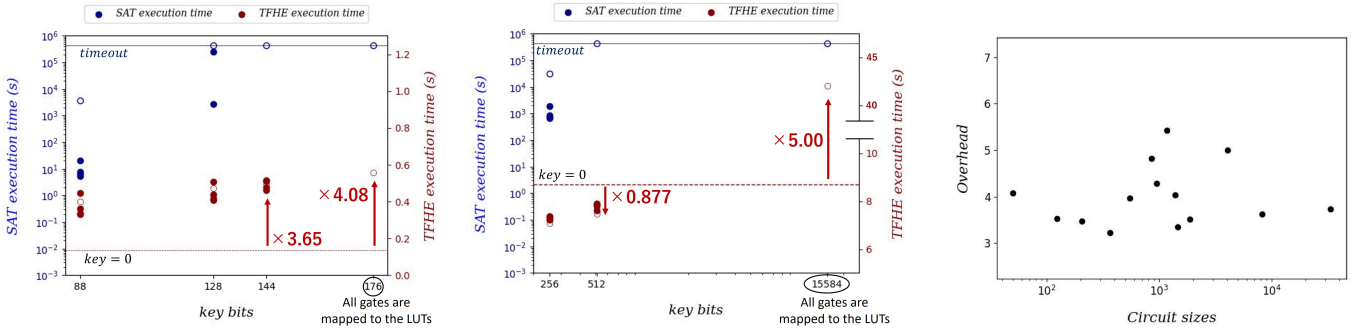
For security assessment, we employed the Satisfiable Modulo Theories (SMT)-attack tool [11] that integrates theoretical solvers, facilitating complex modeling and stronger attacks. Consistent with previous works [6], [8], we set a timeout of 5 days. If the solvers fail to identify the correct key within this duration, we deem the attack as unsuccessful, confirming our method's security.

We utilized the Iyokan [28] framework for evaluating a cryptographic logic circuit using TFHE, adopting its default 80-bit security parameters [17]. To our knowledge, Iyokan provides the fastest execution time among TFHE frameworks.

We tested our method using a decision tree and Combinational Logic Neural Networks (CLNNs). The decision tree model was trained with the UCI heart disease dataset [29] using Python's scikit-learn library [30], converted to a 14-bit integer representation, and manually translated to a Verilog format logic expression. CLNNs represent neural networks as logic circuits, embedding the neural network parameters directly within the logic circuit, a distinct characteristic of this representation. For CLNNs, we employed LogicNets [31] to encapsulate all neural network operations and weights into logical expressions, output in Verilog format. We tested two tasks from [31]: Jet Substructure Classification (JSC) and Network Intrusion Detection (NID). These models were selected to demonstrate the proposed protocol's secure inference capabilities for machine learning models, although it is applicable to other general algorithms.

B. Procedure and Implementation of Experiments

Fig. 4 outlines the experimental procedure validating our secure computation protocol for CLNNs. We first trained sparsely-connected, activated-quantized MLPs for the tasks mentioned earlier using LogicNets PyTorch library. The training process involved 1000 epochs, a mini-batch size of 1024, the ADAM optimizer, and a step-decay learning rate schedule starting at 0.1. Subsequently, we automated the logic expression generation in Verilog format from the trained network using a custom Python script. LogicNets' output was then mapped to 2-LUT and 3-LUT using Yosys-ABC [32].



(a) NID-S (b) JSC-S

Fig. 5: Execution times of SAT attack and TFHE for different logic locking key bits. Open-circle dots correspond to the strategy that selects LUTs near the primary output.

Fig. 6: TFHE runtime overhead vs. circuit size with 100% LUT mapping.

TABLE I: Details of the chosen benchmarks.

Name	Neurons (or Nodes)	In	Out	Accuracy	Circuit sizes
NID-S	593, 100	593	1	89.01%	50
NID-M	593, 256, 128, 128	593	1	89.62%	1,116
NID-L	593, 100, 100, 100	593	1	86.98%	860
JSC-S	64, 32, 32, 32	16	5	67.34%	4,056
DT	11	17	1	86.6%	206

Table I details the benchmarks used. Circuit sizes are computed based on logic gates, considering gate type-dependent TFHE execution time. In TFHE, NOT gates have negligible impact on execution time, while 2-input MUX gates require double the evaluation time compared to 2-input logic gates like AND and OR. Thus, we calculate the circuit size as 2 × (# of 2-input MUX gates) + (# of 2-input logic gates).

For partial LUT mapping, we select LUTs from the pre-mapped circuit. We re-synthesize the circuit with Yosys-ABC [32], keeping selected LUTs and integrating AND, NAND, OR, and NOT gates. Two selection strategies are employed: random selection and near-output selection. For each logic locking key bit, we conducted five trials, four using random selection and one near-output selection.

The SAT attack tool [11] requires bench format netlists for obfuscated and original circuits. We used a custom Python script to convert Verilog to bench format. TFHE evaluation was done using Iyokan [28] with the obfuscated netlist input.

C. Security Analysis against SAT attack

Fig. 5 displays the SAT attack execution times on obfuscated NN circuits for NID-S and JSC-S tasks, illustrating variations with the number of logic locking key bits. Evidently, the SAT attack time extends with more key bits. For the NID-S task, three-fifths of the 128-bit obfuscated circuits resisted the SAT attack. In 144 and 176-bit instances, all obfuscated circuits timed out during the SAT attack. In the JSC-S task, the SAT attack timed out for obfuscations of 512 bits or more. When the JSC-S circuit was entirely mapped with LUTs, the logic locking key reached 15,584 bits, suggesting that such large-scale obfuscation is unnecessary for SAT attack resistance.

The SAT attack resistance varies based on the LUT replacement selection. For example, the SAT attack time for the 88-bit NID-S task exhibited a three-order-of-magnitude difference among five trials. The strategy selecting LUTs near primary outputs (open-circle dots) outperformed the random selection strategy (filled-circle dots), aligning with [33]'s report.

TABLE II: Execution times of SAT attack.

Name	88-bit key	144-bit key
NID-M	1.248×10^3 s	Timeout
NID-L	6.847×10^3 s	Timeout
DECISION-TREE	0.3109 s	Timeout

Table II shows the SAT attack times for three other benchmark circuits. For these, we applied the LUT selection strategy choosing LUTs near primary outputs. These benchmarks achieved resistance to the SAT attack with 144-bit obfuscation.

D. Runtime Performance of TFHE

Fig. 5 shows the variation in TFHE execution time with respect to logic locking key bits on NID-S and JSC-S tasks. The execution time increases linearly with the key bits as the circuit size expands linearly. Moreover, on the JSC-S task, the LUT selection strategy choosing LUTs near the primary output yields a shorter TFHE execution time than the random selection strategy. This suggests that concentrating LUTs near primary outputs allows more logic synthesis flexibility, leading to smaller circuits.

We evaluated the overhead from LUT-based obfuscation, including the TFHE execution time for the original unobfuscated circuit (zero key bits). In the NID-S circuit with only LUTs (176 key bits), the runtime overhead is 4.08×. For 144-bit obfuscation, the overhead decreases to 3.65×, and all five trials resist the SAT attack.

For the JSC-S task with only LUTs (15584-bit obfuscation), the overhead is 5.00×, mostly independent of circuit size as it arises from LUT overhead over primitive gates. Interestingly, for 512-bit obfuscation, the overhead is 0.877×, indicating that the obfuscated circuit's TFHE execution time is shorter than the original circuit's due to fewer LUTs. The logic synthesis tool uses heuristics, as logic mapping is NP-hard, so the observed reduction in circuit size could be due to such side effects. Thus, our experiments empirically show that an obfuscated algorithm representation, resistant to SAT attacks, can be achieved with minimal overhead, potentially obscured by logic synthesis result volatility.

Fig. 6 shows the overhead of LUT-based obfuscation in TFHE runtime for various circuit sizes with ISCAS benchmarks and others mentioned in the paper. It indicates that the overhead remains between 3 to 5 times, suggesting the obfuscation overhead is independent of circuit size.

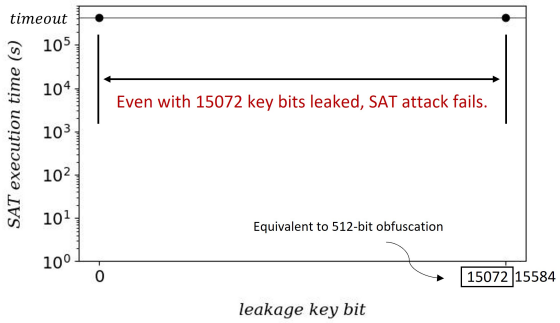


Fig. 7: Resilience against the SAT attack to partial leakage of logic locking key from full LUT-based circuit (JSC-S).

E. Resilience to Partial Leakage of Logic Locking Key

As mentioned in Section III-E4, the proposed protocol may be vulnerable to attacks beyond the HBC model. One countermeasure is to generate a new LUT-based circuit with a fresh pair of public and secret keys by re-synthesizing the algorithm for each inference. However, this carries substantial computational costs. Hence, we evaluated the tolerance to logic locking key leakage against SAT attacks.

Fig.7 shows that even with 15,072 leaked key bits, the SAT attack remains unsuccessful for the JSC-S task, with five trials conducted using random key-bit selection. Fig.7 suggests that updating cryptographic keys and obfuscated circuit is needed only after every 3,000 inferences, as five bits, equivalent to inference output bits, leak per inference. The high resistance of the full-LUT circuit to SAT attacks allows for less frequent updates. A similar pattern was observed with the smaller NID-S task circuit, where 32-bit key leakage was deemed tolerable. For both JSC-S and NID-S tasks, the required logic locking key bits under partial key leakage (512 for JSC-S and 144 for NID-S) are consistent with the results in Fig. 5.

V. Conclusion

In this study, we present a novel approach for protecting both user data and algorithms in two-party PFE scenarios. We introduce logic locking to obscure the algorithm, and TFHE for processing encrypted user data using the encrypted logic locking key. Our method's security has been analyzed under the honest-but-curious model. We applied this approach to machine learning inference applications, specifically combinational logic neural networks (CLNNs) and decision trees. Our experiments demonstrate that the prepared benchmarks resist SAT attacks. Further, we found that an obfuscated algorithm representation resilient to the SAT attack can be achieved with minimal TFHE runtime overhead.

Acknowledgement

This work is supported by JST CREST Grant Number JPMJCR19K5, Japan.

References

[1] L. G. Valiant, "Universal circuits (preliminary report)," in *Proc. ACM symposium on Theory of computing*, 1976.
[2] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, "Tfhe: fast fully homomorphic encryption over the torus," *Journal of Cryptology*, vol. 33, no. 1, pp. 34–91, 2020.
[3] A. C.-C. Yao, "How to generate and exchange secrets," in *Symposium on Foundations of Computer Science*, 1986.
[4] J. A. Roy, F. Koushanfar, and I. L. Markov, "EPIC: Ending piracy of integrated circuits," in *Proc. DATE*, 2008.
[5] P. Beerel, M. Georgiou, B. Hamlin, A. J. Malozemoff, and P. Nuzzo, "Towards a formal treatment of logic locking," *Cryptology ePrint Archive*, 2022.
[6] G. Kolhe *et al.*, "Security and complexity analysis of LUT-based obfuscation: From blueprint to reality," in *Proc. ICCAD*, 2019.
[7] H. M. Kamali, K. Z. Azar, H. Homayoun, and A. Sasan, "Interlock: An intercorrelated logic and routing locking," in *Proc. ICCAD*, 2020.
[8] S. D. Chowdhury, G. Zhang, Y. Hu, and P. Nuzzo, "Enhancing SAT-attack resiliency and cost-effectiveness of reconfigurable-logic-based circuit obfuscation," in *Proc. ISCAS*, 2021.
[9] K. Shamsi, D. Z. Pan, and Y. Jin, "On the impossibility of approximation-resilient circuit locking," in *Proc. HOST*, 2019.
[10] P. Subramanyan, S. Ray, and S. Malik, "Evaluating the security of logic encryption algorithms," in *Proc. HOST*, 2015.
[11] K. Z. Azar, H. M. Kamali, H. Homayoun, and A. Sasan, "SMT attack: Next generation attack on obfuscated circuits with capabilities and performance beyond the SAT attacks," *IACR TCHES*, pp. 97–122, 2019.
[12] A. Baumgarten, A. Tyagi, and J. Zambreno, "Preventing IC piracy using reconfigurable logic barriers," *IEEE Design & Test of Computers*, 2010.
[13] K. Juretus and I. Savidis, "Reduced overhead gate level logic encryption," in *Proc. GLSVLSI*, 2016.
[14] V. Kolesnikov and T. Schneider, "A practical universal circuit construction and secure evaluation of private functions," in *International Conference on Financial Cryptography and Data Security*, 2008.
[15] R. L. Rivest, L. Adleman, M. L. Dertouzos *et al.*, "On data banks and privacy homomorphisms," *Foundations of secure computation*, vol. 4, no. 11, pp. 169–180, 1978.
[16] C. Gentry, *A fully homomorphic encryption scheme*. Stanford Univ., 2009.
[17] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachene, "Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds," in *Proc. ASIACRYPT*, 2016.
[18] V. Shoup and S. Halevi, "Bootstrapping for HElib," Cryptology ePrint Archive, Report 2014/873, Tech. Rep., 2014.
[19] J. Rajendran, Y. Pino, O. Sinanoglu, and R. Karri, "Security analysis of logic obfuscation," in *Proc. DAC*, 2012.
[20] J. Rajendran, H. Zhang, C. Zhang, G. S. Rose, Y. Pino, O. Sinanoglu, and R. Karri, "Fault analysis-based logic encryption," *IEEE Trans. Computers*, vol. 64, no. 2, pp. 410–424, 2013.
[21] Y. Xie, C. Bao, Y. Liu, and A. Srivastava, "2.5 D/3D integration technologies for circuit obfuscation," in *Proc. MTV*, 2016.
[22] J. B. Wendt and M. Potkonjak, "Hardware obfuscation using PUF-based logic," in *Proc. ICCAD*, 2014.
[23] S. M. Plaza and I. L. Markov, "Solving the third-shift problem in IC piracy with test-aware logic locking," *IEEE TCAD*, vol. 34, no. 6, pp. 961–971, 2015.
[24] Y.-W. Lee and N. A. Touba, "Improving logic obfuscation via logic cone analysis," in *Proc. LATS*, 2015.
[25] S. Khaleghi, K. Da Zhao, and W. Rao, "IC piracy prevention via design withholding and entanglement," in *Proc. ASP-DAC*, 2015.
[26] B. Liu and B. Wang, "Embedded reconfigurable logic for ASIC design obfuscation against supply chain attacks," in *Proc. DATE*, 2014.
[27] K. Shamsi and G. Zhao, "An oracle-less machine-learning attack against lookup-table-based logic locking," in *Proceedings of the Great Lakes Symposium on VLSI 2022*, 2022, pp. 133–137.
[28] K. Matsuoka, R. Banno, N. Matsumoto, T. Sato, and S. Bian, "Virtual secure platform: A five-stage pipeline processor over TFHE," in *Proc. USENIX Security*, 2021.
[29] A. Asuncion and D. Newman, "UCI machine learning repository," 2007.
[30] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.
[31] Y. Umuroglu, Y. Akhauri, N. J. Fraser, and M. Blott, "LogicNets: co-designed neural networks and circuits for extreme-throughput applications," in *Proc. FPL*, 2020.
[32] C. Wolf, "Yosys open synthesis suite," 2016.
[33] H. M. Kamali, K. Z. Azar, K. Gaj, H. Homayoun, and A. Sasan, "LUT-lock: A novel LUT-based logic obfuscation for FPGA-bitstream and ASIC-hardware protection," in *Proc. ISVLSI*, 2018.