Low-Overhead Implementation of Logic Encryption Using Gate Replacement Techniques

Xiaoming Chen†, Qiaoyi Liu†, Yu Wang†, Qiang Xu§, Huazhong Yang†
†Department of Electronic Engineering, Tsinghua National Laboratory for Information Science and Technology, Tsinghua University, Beijing, China
§Department of Computer Science & Engineering, Chinese University of Hong Kong, Shatin, N.T., Hong Kong
Email: chenxm1986@gmail.com, yu-wang@tsinghua.edu.cn

***Abstract*—The globalization of the semiconductor industry has caused many challenges to prevent intellectual property (IP) piracy. Logic encryption is an effective technique for hardware IP protection. Researchers have proposed various logic encryption techniques, which introduce large overheads in delay, power and area. This paper aims to significantly reduce these overheads by proposing a novel gate replacement-based implementation. A simulated annealing algorithm is adopted to find the optimal replacement positions. Experimental results show that our implementation reduces 15% to 30% of the area overhead, and 60% to 80% of the delay and power overheads, while the encryption quality is almost not degraded. The idea of gate replacement can be applied to various XOR/XNOR-based logic encryption approaches.**

***Index Terms*—Gate replacement; Hardware security; Logic encryption**

I. Introduction

The continuous globalization of the integrated circuit (IC) design process and fabrication industry have raised the intellectual property (IP) piracy problem. Attackers can reverse-engineer a manufactured IC and then claim the ownership of the IP. Untrusted foundries can over-produce ICs from reverse-engineered designs and illegally sell the counterfeit copies. Therefore, IP vendors are facing many challenges to protect IPs from piracy, reverse engineering, and overproduction. It is estimated that the semiconductor industry loses 4 billion dollars annually due to IP infringement [1].

One way to protect hardware IPs is to encrypt the circuit functionality using a so-called logic encryption technique [2]–[12] so that only authorized users can use the chips. Logic encryption hides the circuit functionality by inserting a number of key gates which are controlled by key bits. Only when the correct key vector is applied, the circuit can operate correctly. On applying incorrect key vectors, the circuit cannot generate correct outputs. Consequently, even if attackers obtains the encrypted netlist by reverse-engineering, they cannot know the correct netlist unless they know the key.

Most of existing logic encryption approaches insert XOR/XNOR key gates into the original netlist. Key gates can be inserted at random positions [4], but the encryption quality cannot be guaranteed. To improve the encryption quality, key gates should be inserted into carefully selected positions, such that the ambiguity for attackers is maximized [5]. However, a major disadvantage is the introduced overhead. The delay and power overheads can be larger than 200% for some circuits as reported in [5]. A low-overhead logic encryption approach is proposed in [7], which inserts key gates only on non-critical paths so the delay is not affected. However, the encryption quality is completely ignored.

Researchers have also proposed some attack methods against logic encryption [9]–[14]. Most of them rely on observed correct input-output pairs, which leads to several limitations. One can obtain only a small set of input-output pairs of a combinational block from a manufactured IC, because internal logics cannot be arbitrarily controlled in an actual chip, especially for a partial scan design. It is expected that the correct key vector can be solved from these observed input-output pairs. However, these attack methods are likely to get many key vectors that can satisfy the observed input-output pairs. Verifying which is the correct one is still an impossible task. On the other hand, people have also developed defense techniques. For example, two latest researches [2], [3] have successively defeated the recent satisfiability (SAT) attack method [13]. Till now, all attack methods to logic encryption have been defeated.

This paper aims to reduce the logic encryption overheads while maintaining the encryption quality. We propose a novel gate replacement-based implementation for logic encryption. This technique reuses some original gates in the circuit so that the area, delay, and power overheads are reduced. We make the following contributions in this paper.

- We propose a novel idea of using gate replacement for logic encryption, which aims to reduce the area, delay, and power overheads.
- We show that compared with XOR/XNOR key gate insertion [5], the area overhead is reduced by 15% to 30%, and the delay and power overheads are reduced by 60% to 80%. Meanwhile, the encryption quality is almost not degraded, if the replacement positions are carefully selected by a simulated annealing (SA) algorithm [15].

Please note that our primary goal is to propose a new fundamental implementation to reduce the overheads of logic encryption approaches, but not to propose a new logic encryp-

This work was supported by National Natural Science Foundation of China under Grant 61532017 and Grant 61432017, and the National Science Fund for Excellent Young Scholars under Grant 61622403.

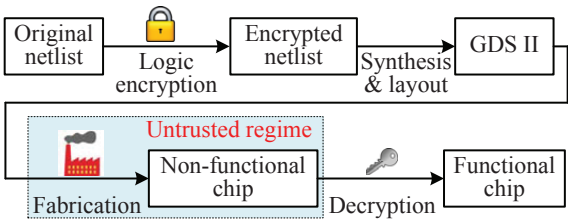

Fig. 1: Logic encryption and decryption in the IC design and fabrication flow.

tion algorithm to defeat attacks. Although we investigate gate replacement by implementing it into a specific logic encryption approach, the idea can also be applied to other logic encryption approaches. For example, it should be able to be applied into the two most recent logic encryption approaches [2], [3] which can defeat the SAT attack [13].

II. Background And Motivation

We will introduce the background of logic encryption and present our motivation in this section.

A. Logic Encryption

Fig. 1 shows logic encryption and decryption in the IC design and fabrication flow. The original netlist is encrypted in the design stage. After synthesis and layout, the design is sent to the foundry for fabrication. The foundry is untrusted. Attackers in the foundry can obtain the encrypted design without the key values. After fabrication, the key values will be loaded into a tamper-proof memory for decryption, and then chips are sold to the market. One can also integrate a physical unclonable function (PUF) [16] into each chip to assign a unique key to each chip [7].

Logic encryption can be described as follows. A combinational block can be descried by a multi-input and multi-output function $\mathbf{z} = \mathbf{f}(\mathbf{x})$, where $\mathbf{x}$ is the input vector and $\mathbf{z}$ is the output vector. The encrypted circuit is described by $\mathbf{z} = \mathbf{f}'(\mathbf{x}, \mathbf{k})$, where $\mathbf{k}$ is the key vector. Let $\mathbf{k}_0$ be the correct key vector. Only when $\mathbf{k}_0$ is applied, the circuit can operate correctly, i.e., for $\forall \mathbf{x}$, $\mathbf{f}(\mathbf{x}) = \mathbf{f}'(\mathbf{x}, \mathbf{k}_0)$. On applying any incorrect key vector, the circuit cannot generate correct outputs for some or all input vectors, i.e., for $\forall \mathbf{k} \neq \mathbf{k}_0$, $\exists \mathbf{x}$ such that $\mathbf{f}(\mathbf{x}) \neq \mathbf{f}'(\mathbf{x}, \mathbf{k})$.

The quality of logic encryption can be evaluated by Hamming distance [5], [6]. According to the analysis in [5], if a circuit has M output bits and the Hamming distance of the output bits after logic encryption is H, an attacker has to consider C_M^H output combinations to crack the encryption. Obviously, a 50% Hamming distance (i.e., $H = \frac{M}{2}$) maximizes the ambiguity for attackers. This means that on applying an incorrect key vector, it is desirable that 50% of the output bits should be corrupted. To evaluate the Hamming distance after logic encryption, we can apply a set of random input vectors and random key vectors to evaluate an average Hamming distance between the correct and corrupted outputs.

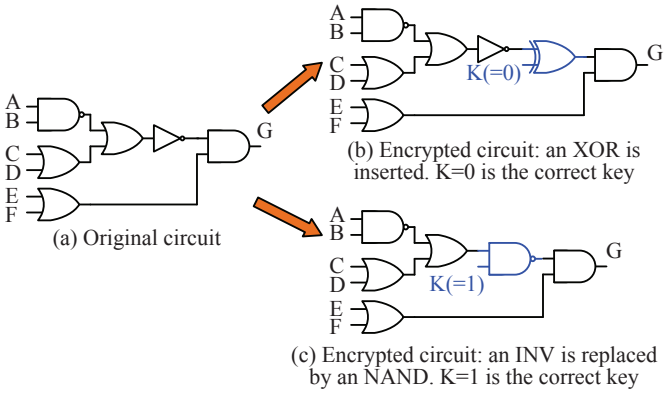

Fig. 2: A simple example to illustrate our motivation.

TABLE I: Comparison between XOR/XNOR-based and gate replacement-based logic encryption approaches.

	Area (μm^2)	Delay (ns)	Power (μW)	Hamming dist.
Unencrypted	8.32	0.217	0.8243	N/A
XOR inserted	13.52	0.303	1.9288	37.5%
Replacement	11.44	0.248	1.4478	35.16%

The fundamental idea of logic encryption is to insert XOR and XNOR key gates, and the additional inputs caused by key gate insertion are key bits. The approach proposed in [5] inserts key gates at carefully selected positions that have the largest impact on the output, and, thus, the Hamming distance tends to reach 50%. Consider a simple example shown in Fig. 2 (a). Assume that an XOR key gate is inserted as shown in Fig. 2 (b). If the key bit is incorrect, the encrypted circuit produces incorrect outputs for 48 input vectors out of the possible 128 input vectors (including the key bit), so the Hamming distance is $\frac{48}{128}$ =37.5%.

B. Motivation

Inserting XOR and XNOR gates leads to large overheads in area, delay, and power. If we can reuse some gates in the original circuit to act as key gates, the overheads can be significantly reduced. For example, we can replace an INV gate by a two-input NAND gate, as shown in Fig. 2 (c). The additional input of the replaced gate is the key bit. When the key is incorrect, the encrypted circuit produces incorrect outputs for 45 input vectors out of the possible 64 input vectors, so the Hamming distance is $\frac{45}{128}$ =35.16%, which is only a little smaller than that of XOR insertion. However, as compared in Table I, the overheads are significantly reduced. Consequently, it is possible to replace some gates in the circuit, such that we can achieve a similar effect to XOR/XNOR key gate insertion with lower overheads.

Please note that Fig. 2 (c) is only a simple example to illustrate our motivation. Actually only applying such a replacement does not provide any ambiguity, because the replaced NAND gate can operate correctly only when the key bit is 1. If the key bit is 0, its output is fixed, which is easy to

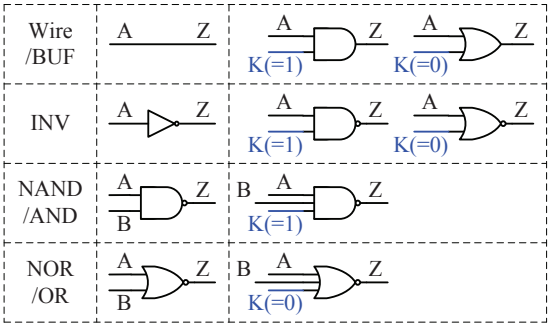

Fig. 3: Direct gate replacement for logic encryption. K(=x) is the key bit and x is the correct key bit.

detect. The ambiguity will come from combining two different gate replacement schemes proposed in the next section.

III. Logic Encryption Using Gate Replacement

The gate replacement technique was originally used for leakage power and/or aging optimization [17], [18]. In this section, we will present the idea of using the gate replacement technique for logic encryption.

A. Direct Gate Replacement

The fundamental idea of gate replacement is to replace a gate by a same-functional gate with one more input pin. For example, a two-input NAND gate can be replaced by a three-input NAND gate. The additional input is treated as a key bit and the replaced gate is called a key gate. When the key bit is correct, the key gate operates as the same as the original gate. On the contrary, when the key bit is incorrect, the key gate cannot work correctly. This approach is named *direct gate replacement (DGR)*. Fig. 3 shows the DGR implementations for basic gate types and wires, in which the key bit is marked by K and the correct key bit is marked by K(=0) or K(=1).

However, as mentioned above, only using DGR to encrypt a circuit is very weak. If an attacker can obtain the encrypted netlist by reverse-engineering, all the correct key bits can be trivially obtained by checking the type of each key gate. For an AND or NAND key gate, the correct key bit is 1, while for an OR or NOR key gate, the correct key bit is 0.

B. Indirect Gate Replacement

To tackle the above problem, we propose a complementary approach, in which the correct key bit is opposite with that in DGR for each type of key gates. In DGR, if the key bit is incorrect, the output of the key gate is fixed. In other words, in DGR, a fixed output of the key gate corresponds to the wrong state. If we can also make the fixed output correspond to the correct state, then we have two opposite cases and the above problem is solved. To achieve this goal, we still apply the gate replacement technique, but the additional input is indirectly controlled by the key bit. Instead, an extra key gate which is directly controlled by the key bit is inserted and connected to the additional input of the replaced gate, such that the fixed output of the key gate corresponds to the correct state of the

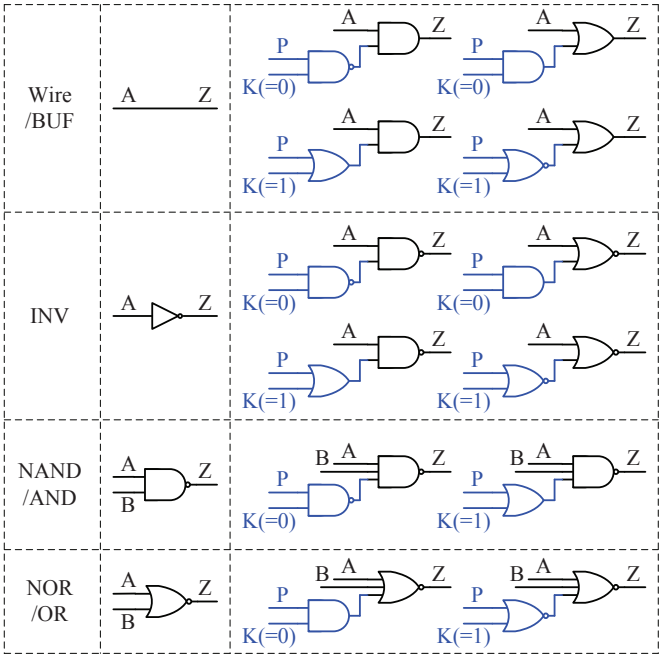

Fig. 4: Indirect gate replacement for logic encryption. The blue gate is the inserted key gate. K(=x) is the key bit and x is the correct key bit. P is a floating input that needs to be connected to a valid wire (without forming loops).

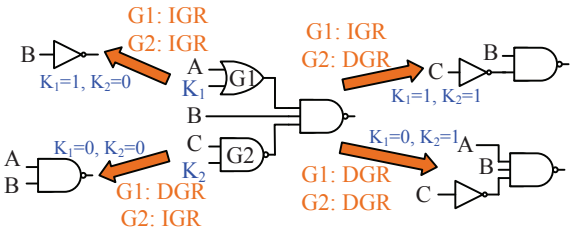

Fig. 5: Example of logic encryption using gate replacement. Key bit values and encryption methods corresponding to the four different possibilities are shown.

replaced gate. There is a floating input after inserting the extra key gate. The floating input should be connected to a valid wire in the circuit (without forming loops). This approach is named *indirect gate replacement (IGR)*. Fig. 4 shows the IGR implementations for basic gate types and wires. In IGR, the correct key bit is opposite with that in DGR when the key gates are of the same type.

DGR and IGR are both required to encrypt a circuit to confuse attackers. Both DGR and IGR can be applied to wires or gates. Specifically, any input or output wire of any gate, or any gate itself, can be considered to be encrypted by DGR or IGR, if the encryption method is valid. We show a simple example in Fig. 5, in which the decrypted circuit has four possibilities according to the key bit values and encryption methods of gate replacement. It is impossible to identify the correct decrypted circuit if no other information is available.

C. Practical Issues

There are two practical issues that can weaken the encryption, so they must be solved when implementing logic encryption using DGR and IGR.

First, from Fig. 3, the number of inputs of a key gate is at least three for an original NAND, AND, NOR, or OR gate. However, the key gates shown in Fig. 4 always have two inputs. As such, key gates with more than two inputs can be trivially detected. To avoid this problem, some key gates implemented by IGR should also have more than two inputs. In this case, more than one floating inputs are generated from one key gate implemented by IGR. In our implementation, the number of inputs of each key gate implemented by IGR is randomly generated. This will also be mentioned in the next section.

Second, in IGR, if we consider the key gate and the corresponding replaced gate as a pair of "key gate + replaced gate", there are eight possible pairs that can appear in IGR: AND + OR, AND + NOR, NAND + AND, NAND + NAND, OR + AND, OR + NAND, NOR + OR, and NOR + NOR. Hence, if any other pair (e.g., AND + AND) appears, attackers trivially know that the former gate in the pair is the key gate and the key gate is implemented by DGR. To avoid this problem, DGR should be only applied to such pairs. In other words, if one gate and one of its immediate fanout gates form a pair that does not appear in the possible pairs of IGR, this gate or its output wire cannot be encrypted by DGR.

D. Advantages Compared with XOR/XNOR Insertion

Compared with other logic encryption implementations which insert XOR/XNOR gates, the gate replacement-based implementation has two promising advantages. First, gate replacement is more flexible. XOR/XNOR gates can be regarded as wire replacement, but our approach can be applied to both wires and gates. The second advantage is the low overhead. XOR/XNOR insertion can lead to large area, power, and delay overheads, but our approach is expected to significantly reduce these overheads, because gate replacement can reuse some original gates in the circuit, as illustrated in Section II-B.

IV. Simulated Annealing for Logic Encryption

In this section, we will use an SA algorithm to select gate replacement positions. We will first explain some challenges of selecting replacement positions, and then we will show the flow of the SA algorithm. We want to emphasize again that we are NOT developing a new logic encryption algorithm, but we will show that by carefully selecting the replacement positions, gate replacement can achieve a similar encryption quality as previous implementations.

A. Challenges of Selecting Replacement Positions

Two important factors should be considered when selecting gates/wires to replace for gate replacement.

First, in order to achieve the expected 50% Hamming distance, the selected gates/wires should have the largest impact on the output among all gates/wires. If a replaced gate/wire only affects few output bits, this encryption is weak. Selecting an individual replacement position with the largest impact on the output can be achieved by a fault analysis method [5]. A natural idea is to develop a heuristic algorithm which replaces a gate/wire with the largest impact each time. However, the problem is that different replaced gates can affect each other. This issue can be explained as follows. One position is selected such that it has the largest impact on the output. However, after the second position is also selected and replaced, the first position may not have the largest impact due to the replacement of the second position. Hence, in order to get a global optimum, we should consider all the candidate positions simultaneously.

Second, signal probabilities should also be considered when selecting positions to replace. We take the first implementation of IGR for a wire as an example to explain this issue. When the key bit is correct, its function is simply $Z = A$. When the key bit is incorrect, its function is $Z = A\bar{P}$. If the signal probabilities of the two cases (A and $A\bar{P}$) are similar, this encryption is also weak, because the output keeps the same under most of input possibilities. To maximize the ambiguity, we should select A with a large signal probability and $\bar{P}$ with a small signal probability. If so, there will be a big difference between the two cases, and, hence, it tends to generate a large Hamming distance. The problem is that, a wire with the largest impact on the output may not have a satisfactory signal probability, and vice versa. In addition, the first problem also exists here, because late replacements can change the signal probabilities such that early selections may become bad.

Considering the above two problems, it is challenging to select replacement positions heuristically for gate replacement. Consequently, in this paper, we adopt the SA algorithm, which is expected to converge to the global optimum in probability [19].

B. SA Algorithm

Algorithm 1 shows the overall flow of the proposed SA algorithm to select replacement positions. It starts from a randomly generated solution followed by a typical SA procedure. The objective is the Hamming distance, which is evaluated by applying 500 random input/key vectors. We make an improvement compared with the standard SA algorithm. When the Hamming distance of the latest solution is between 49% and 51%, the SA procedure exits in advance regardless of the current temperature (lines 7 to 10), since this solution is already good enough.

The core operation of the SA algorithm is to generate a neighboring solution from an existing solution (line 5 of Algorithm 1), which is shown in Algorithm 2. We consider four operations: swapping an encrypted position and an unencrypted position, changing the encryption method on an encrypted gate, changing a floating connection on an IGR-encrypted gate, and re-generating floating connections on an IGR-encrypted gate. The four operations have different probabilities, since their potential to improve the objective are different. The two issues explained in Section III-C have

Algorithm 1 SA algorithm for logic encryption.

Input: $nKey$, $T_0 = 1.5$, $r = 0.9999$;//$nKey$: number of key bits. T_0: initial temperature. r: annealing speed.
Output: $Keys$, hd;//$Keys$: resultant key. hd: Hamming distance of the solution.

Randomly select $nKey$ positions and randomly apply encryptions at these positions. The solution is denoted by S;
$hd =$ EvaluateHD(S);
$T = T_0$;
while $T > 10^{-4}$ **do**
  $S' =$ Neighbor(S);
  $hd' =$ EvaluateHD(S');
  if $hd' \geq 0.49$ **and** $hd' \leq 0.51$ **then** //Exit SA loop in advance if solution is good enough
    $S = S'$; $hd = hd'$;
    break;
  end if
  if Rand$(0,1) < \exp\left(\frac{hd'-hd}{T}\right)$ **then**
    $S = S'$; $hd = hd'$;
  end if
  $T = T \times r$;
end while
Generate $Keys[0], Keys[1], \cdots, Keys[nKey-1]$ according to the encryption method at each position for solution S;

Algorithm 2 $S' =$ Neighbor(S).

Input: S;//S: current solution.
Output: S';//S': a neighboring solution of S.

$r =$ Rand$(0,1)$;
if $r < 0.5$ **then** //Swap two positions. 50%
  Randomly select an encrypted position and remove the encryption. Randomly select an un-encrypted position and randomly apply a valid encryption;
else if $r < 0.7$ **then** //Change on a gate. 20%
  Randomly select a gate with at least one encrypted gate, input wire, or output wire. Randomly remove an encryption and randomly apply a valid encryption;
else if $r < 0.9$ **then** //Change a floating connection. 20%
  Randomly select a floating connection of an IGR encryption. Randomly re-connect it to a valid wire.
else //Re-generate floating connections. 10%
  Randomly select an IGR encryption. Randomly re-generate number of floating inputs and randomly re-connect them to valid wires.
end if

been considered when applying encryption. When connecting a floating input, we select a wire that does not form any loops. Since Algorithm 2 is self-explanatory, we will not explain it in more detail.

V. Experimental Results

A. Experimental Setup

The gate replacement-based logic encryption approach is implemented by C++. We use the ISCAS-85 benchmarks [20] to evaluate the proposed approach. We apply logic encryption on the raw netlists and then the encrypted netlists are synthesized using a 65nm library named "CORE65LPSVT". The raw netlists are also synthesized to act as un-encrypted circuits for overhead comparison. The circuit delay is set as the synthesis constraint so it is minimized during logic synthesis.

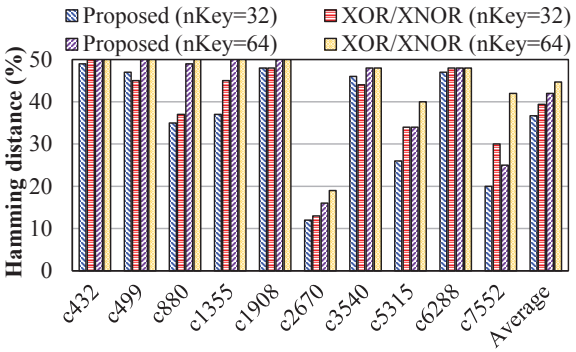

Fig. 6: Comparison of the Hamming distance.

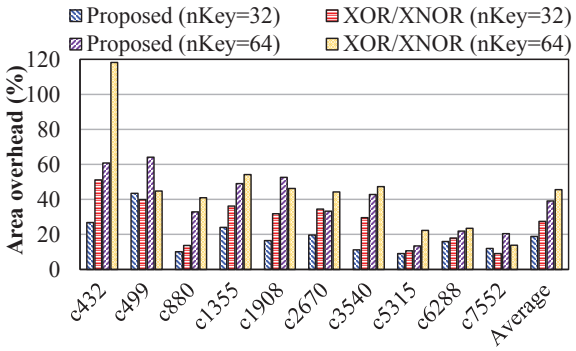

Fig. 7: Comparison of the area overhead.

B. Comparison

In this subsection, we will compare the Hamming distance and logic encryption overheads between the proposed approach and an existing approach which inserts XOR/XNOR key gates [5]. We will compare the two approaches when the key size is 32 and 64, respectively.

Fig. 6 compares the Hamming distance. The average Hamming distance of our approach is about 3 percentage points smaller than that of the XOR/XNOR-based approach. For 3 benchmarks (c1355, c5315 and c7552), the difference is larger than 5 percentage points, and for the other 7 benchmarks, our approach gets almost the same Hamming distance as the existing method. Considering the physical meaning of the Hamming distance explained in Section II-A, a small difference in the Hamming distance cannot significantly affect the encryption quality. For example, if a circuit has 100 output bits and the Hamming distance is 30 and 40, the number of output combinations that an attacker has to consider is 2.94×10^{25} and 1.37×10^{28}, respectively. As can be seen, even when the Hamming distance is decreased by 10 percentage points, the number of output combinations that an attacker has to consider is still an astronomical figure.

Figs. 7, 8, and 9 compare the area, delay, and power overheads, respectively. On average, our approach has smaller overheads than the XOR/XNOR-based approach. The average area overhead is reduced by 15% to 30%, which is smaller than the reduction in the delay and power overheads. The reason can be explained as follows. The area overhead of

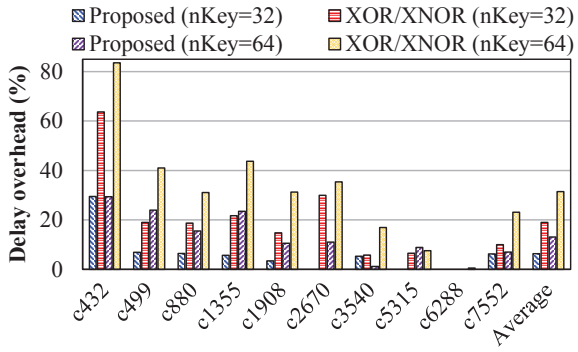

Fig. 8: Comparison of the delay overhead.

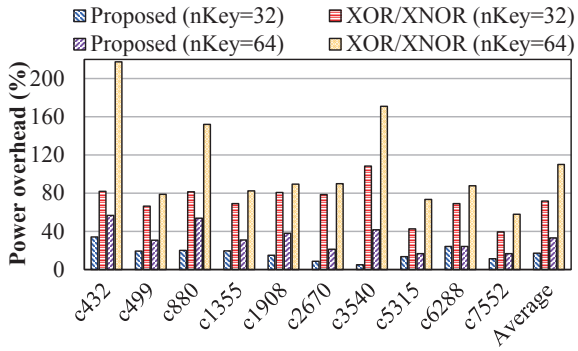

Fig. 9: Comparison of the power overhead.

TABLE II: Statistical results of 100 runs.

		Hamming dist.	Area (μm^2)	Power (ns)	Delay (μW)
c880	Mean	34.6%	1309.7	1.161	292.9
	Std. dev.	0.64%	82.3	0.041	16.0
	$\frac{\text{Std. dev.}}{\text{Mean}}$	1.8%	6.3%	3.5%	5.5%
c1355	Mean	35.2%	1140.3	1.251	643.7
	Std. dev.	1.33%	61.6	0.046	53.1
	$\frac{\text{Std. dev.}}{\text{Mean}}$	3.8%	5.4%	3.7%	8.2%

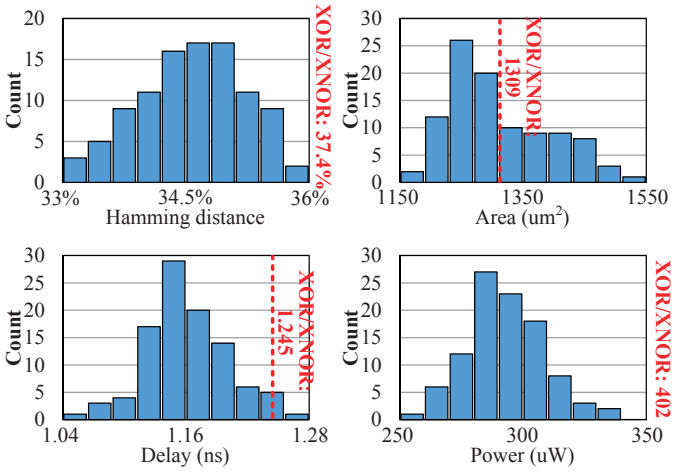

Fig. 10: Histograms of the results of 100 runs for c880.

an IGR replacement can be similar to that of an XOR or XNOR key gate when the inserted key gate has more than two inputs. However, the reductions in the delay and power overheads are very promising. As shown in Figs. 8 and 9, the delay and power overheads are reduced by about 60% and 80%, respectively. The delay and power overheads of our approach with 64 key bits are even smaller than those of the XOR/XNOR-based approach with 32 key bits. The benchmark c6288 is a multiplier which has a very long critical path, so the delay overhead of both our approach and the XOR/XNOR-based approach are ignorable. The power overheads of both approaches look rather high. The reason is that the circuit delay is set as the synthesis constraint so the synthesizer always tries to minimize the delay, which significantly increases the power.

C. Stability of SA Algorithm

Since the results of the SA algorithm are random, it is necessary to analyze the stability of the algorithm. We take c880 and c1355 as two examples. We run the SA algorithm 100 times and the statistical results are shown in Table II. As can be seen, the ratio of the standard deviation to the mean is quite small, indicating that the randomness of the results is low.

To further illustrate the stability of the algorithm, we show the histograms for c880 in Fig. 10. The results of the XOR/XNOR-based approach are also marked on the figure for comparison. The Hamming distance obtained by our approach is a little smaller (1 to 4 percentage points) than that of the XOR/XNOR-based approach. 60, 95, and 100 out of the 100 runs get smaller area, delay and power overheads than the XOR/XNOR-based approach, respectively. These comparisons also reveal that our approach tends to get smaller area, delay and power overheads than the XOR/XNOR-based approach. Consequently, we can conclude that, by applying a random result obtained from the SA algorithm, we still have a high probability that the overheads can be reduced. To obtain a more reliable result, we can select an optimal encryption scheme from a number of SA runs.

VI. Conclusions

In this paper, we have proposed a novel idea of using the gate replacement technique to implement logic encryption, such that the area, delay and power overheads are reduced. We use an SA algorithm to find the optimal replacement positions. We have shown that compared with the XOR/XNOR-based approach, the average area overhead is reduced by 15% to 30%, and the average delay and power overheads are reduced by 60% to 80%, while the Hamming distance is decreased by only 3 percentage points on average. These results can be regarded as the maximum potential of gate replacement in logic encryption. The proposed gate replacement technique is a new fundamental implementation for logic encryption. In our future work, we will investigate how to apply it into other logic encryption approaches that can defeat the SAT attack.

REFERENCES

[1] "Innovation is at Risk: Losses of up to $4 Billion Annually due to IP Infringement. [Online] http://www.semi.org/en/Issues/IntellectualProperty/ssLINK/P043785."

[2] M. Yasin, B. Mazumdhar, O. Sinanoglu, and J. Rajendran, "SARLock: Resisting SAT attacks on Logic encryption," in *HOST*, 2016.

[3] Y. Xie and A. Srivastava, "Mitigating SAT Attack on Logic Locking," in *CHES*, 2016, pp. 127–146.

[4] J. A. Roy, F. Koushanfar, and I. L. Markov, "EPIC: Ending Piracy of Integrated Circuits," in *DATE*, March 2008, pp. 1069–1074.

[5] J. Rajendran, H. Zhang, C. Zhang, G. Rose, Y. Pino, O. Sinanoglu, and R. Karri, "Fault Analysis-Based Logic Encryption," *TC*, vol. 64, no. 2, pp. 410–424, Feb 2015.

[6] A. Baumgarten, A. Tyagi, and J. Zambreno, "Preventing IC Piracy Using Reconfigurable Logic Barriers," *IEEE D&T*, vol. 27, no. 1, pp. 66–75, Jan 2010.

[7] J. Zhang, "A Practical Logic Obfuscation Technique for Hardware Security," *TVLSI*, vol. 24, no. 3, pp. 1193–1197, 2016.

[8] J. B. Wendt and M. Potkonjak, "Hardware Obfuscation Using PUF-based Logic," in *ICCAD*, 2014, pp. 270–277.

[9] S. M. Plaza and I. L. Markov, "Solving the Third-Shift Problem in IC Piracy With Test-Aware Logic Locking," *TCAD*, vol. 34, no. 6, pp. 961–971, June 2015.

[10] Y. W. Lee and N. A. Touba, "Improving logic obfuscation via logic cone analysis," in *LATS*, March 2015, pp. 1–6.

[11] S. M. Plaza and I. L. Markov, "Protecting Integrated Circuits from Piracy with Test-aware Logic Locking," in *ICCAD*, 2014, pp. 262–269.

[12] M. Yasin, J. Rajendran, O. Sinanoglu, and R. Karri, "On Improving the Security of Logic Locking," *TCAD*, vol. 35, no. 9, pp. 1411–1424, 2016.

[13] P. Subramanyan, S. Ray, and S. Malik, "Evaluating the security of logic encryption algorithms," in *HOST*, May 2015, pp. 137–143.

[14] M. Yasin, S. M. Saeed, J. Rajendran, and O. Sinanoglu, "Activation of Logic Encrypted Chips: Pre-Test or Post-Test?" in *DATE*, March 2016, pp. 139–144.

[15] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *SCIENCE*, vol. 220, no. 4598, pp. 671–680, 1983.

[16] B. Gassend, D. Clarke, M. van Dijk, and S. Devadas, "Silicon Physical Random Functions," in *CCS*, 2002, pp. 148–160.

[17] L. Yuan and G. Qu, "Enhanced leakage reduction technique by gate replacement," in *DAC*, June 2005, pp. 47–50.

[18] Y. Wang, X. Chen, W. Wang, Y. Cao, Y. Xie, and H. Yang, "Leakage Power and Circuit Aging Cooptimization by Gate Replacement Techniques," *TVLSI*, vol. 19, no. 4, pp. 615–628, April 2011.

[19] V. Granville, M. Krivanek, and J.-P. Rasson, "Simulated annealing: a proof of convergence," *TPAMI*, vol. 16, no. 6, pp. 652–656, Jun 1994.

[20] F. Brglez, "A neutral netlist of 10 combinational benchmark circuits and a target translation in FORTRAN," in *ISCAS*, 1985.