What to Lock? Functional and Parametric Locking

Muhammad Yasin, Abhrajit Sengupta
Electrical and Computer Engineering
New York University
{yasin,as9397}@nyu.edu

Benjamin Carrion Schafer,
Yiorgos Makris
Electrical and Computer Engineering
The University of Texas at Dallas
{schaferb,yiorgos.makris}@utdallas.edu

Ozgur Sinanoglu
Electrical and Computer Engineering
New York University Abu Dhabi
ozgursin@nyu.edu

Jeyavijayan (JV) Rajendran
Electrical and Computer Engineering
The University of Texas at Dallas
jv.ee@utdallas.edu

ABSTRACT

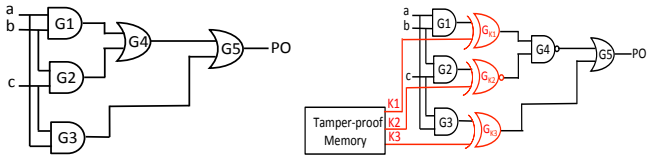
Logic locking is an intellectual property (IP) protection technique that prevents IP piracy, reverse engineering and overbuilding attacks by the untrusted foundry or end-users. Existing logic locking techniques are all based on locking the functionality; the design/chip is nonfunctional unless the secret key has been loaded. Existing techniques are vulnerable to various attacks, such as sensitization, key-pruning, and signal skew analysis enabled removal attacks. In this paper, we propose a tenacious and traceless logic locking technique, TTlock, that locks functionality and provably withstands all known attacks, such as SAT-based, sensitization, removal, etc. TTLock protects a secret input pattern; the output of a logic cone is flipped for that pattern, where this flip is restored only when the correct key is applied. Experimental results confirm our theoretical expectations that the computational complexity of attacks launched on TTLock grows exponentially with increasing key-size, while the area, power, and delay overhead increases only linearly. In this paper, we also coin "parametric locking," where the design/chip behaves as per its specifications (performance, power, reliability, etc.) only with the secret key in place, and an incorrect key downgrades its parametric characteristics. We discuss objectives and challenges in parametric locking.

1. INTRODUCTION

Today's integrated circuits (ICs) are designed and fabricated in a globalized, multi-vendor environment due to the high cost of building and maintaining a foundry [1]. Further, to meet strict time-to-market constraints, a design company may purchase intellectual property (IP) cores from third-party IP vendors. A globalized IC supply chain allows for the crucial assets to be handled by untrustworthy agents and creates opportunities for IP/IC piracy, reverse engineering, counterfeiting, and malicious modifications to the IC in the form of hardware Trojans [2, 3].

GLSVLSI '17, May 10-12, 2017, Banff, AB, Canada

DOI: http://dx.doi.org/10.1145/3060403.3060492.



(a) Example circuit [12]. (b) Circuit locked using XOR/XNOR key gates. The correct key $\{K1, K2, K3\}$ value is 100.

Figure 1: Functional logic locking using XOR/XNOR gates [6].

To thwart such attacks, countermeasures such as IC camouflaging [4] and split manufacturing [5] have been developed. IC camouflaging and split manufacturing protect only against the untrusted user and untrusted foundry, respectively. *Logic locking* is a technique that thwarts IP piracy, overbuilding, and reverse engineering attacks by locking a chip with a secret key [6–11]. To enable chip-locking features, additional logic, e.g., XOR/XNORs gates referred to as *key-gates*, is added to the original netlist to obtain a *locked netlist*. Logic locking protects against both untrusted foundry and user. The secret key needs to be loaded for the design/chip to become functional; otherwise, the chip produces incorrect outputs. Logic locking techniques lock the functionality of the design/chip.

Figure 1(a) shows an example design netlist of a circuit, and Figure 1(b) shows its functionally locked version through two XOR and one XNOR key-gates. One of the inputs of each key-gate is driven by a wire in the original design, while the other input, referred to as the *key-input*, is driven by a key-bit stored in a tamper-proof memory. Logic locking aims to deliver the following protection: (i) Without the knowledge of the secret key, exact design details cannot be retrieved, and (ii) A locked IC (or a locked netlist) will not generate correct output unless it is activated, i.e., the secret key is loaded onto the chip's memory.

1.1 Prior work

Previous logic locking techniques have inserted key-gates based on strategies, such as random logic locking (RLL) [6] and fault analysis-based logic locking (FLL) [7]. **Sensitization attack** sensitizes the key-bits to the outputs by applying judiciously crafted patterns, which, when applied on a working chip, reveals the key-bit values. This attack breaks RLL and FLL [9]. Strong logic locking (SLL) inserts key-gates such that they interfere with each other, making it difficult to sensitize the individual key-bits to outputs [9].

A key-pruning attack that breaks all existing combinational logic-locking techniques has been proposed in [12]. The attack uses Boolean satisfiability (SAT) based algorithms, and we refer to it as the **SAT attack**. The SAT attack uses modern SAT solvers to compute *discriminating input patterns (DIPs)* [12] on the locked netlist (e.g., obtained by reverse-engineering). A working chip with the secret key loaded in its memory is then used as an *oracle* to obtain the responses to these DIPs. A DIP X_d is an input value for which at least two different key values, $k1$ and $k2$, produce differing outputs, $o1$ and $o2$, respectively. Since $o1$ and $o2$ are different, one of the key values or both of them are incorrect. It is possible for a single DIP to rule out multiple incorrect key values. In each iteration, a DIP is generated and applied to the chip. Based on the output observed, a set of keys is eliminated. The process repeats until no DIP can be found, and thus, eliminating all the incorrect keys. Thus, any key in the remaining search space is a valid key. The worst-case scenario for the SAT attack arises when the attack can discriminate at most one incorrect key value with each DIP. In this case, $2^k - 1$ DIPs are required for k key bits. The attack complexity can be represented in terms of the number of DIPs generated by the SAT attack to break a logic-locked circuit.

To thwart the SAT attack, two logic-locking techniques—SARLock [13] and Anti-SAT [14] (see Figure 2)—have been recently proposed. SARLock leaves the logic cone implementation, the IP-to-be-protected, as is, and corrupts the output of the logic cone for all keys but the correct one. The correct key values are hardcoded in logic gates to *mask* the output corruption for the correct key [13]. Anti-SAT inserts key-gates at the inputs of two complementary logic blocks that converge at an AND gate. The output of the AND gate is always 0 only for the correct key; otherwise, it may be 1 for some input values, corrupting an internal node in the original design to produce occasionally incorrect outputs.

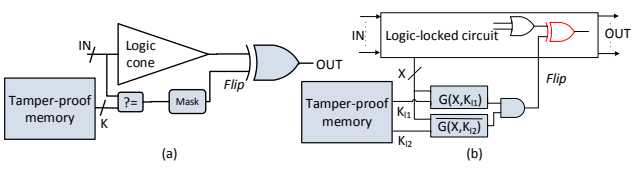


Figure 2: (a) SARLock: The *flip* signal goes low for all input values only on applying the correct key [13]. This technique is vulnerable to removal attack: (b) Anti-SAT: The *flip* signal goes low for all input values only on applying the correct key [14]. This technique is vulnerable to signal skew analysis (SSA) attack [15].

Limitations of existing work. All combinational logic-locking techniques except for SARLock [13] and Anti-SAT [14] are vulnerable to SAT attacks. Yet SARLock [13] and Anti-SAT [14] exhibit security vulnerabilities themselves. SARLock is vulnerable to **removal attack**: given a protected netlist, one can identify the comparator/mask blocks and the XOR gate that directly feeds the output by tracing the transitive-fanout of key-inputs, and remove these blocks, leaving only the unprotected logic cone. Thus, the proprietary IP is recovered. Anti-SAT is vulnerable to **signal skew analysis (SSA) attack** [15]: given a protected netlist, an attacker can identify the output gate of the Anti-SAT block; it is the node whose inputs have the maximal difference in signal probabilities. The attacker can then resynthesize the design with a constant 0 (1) on the output of Anti-SAT, retrieving the original design.

1.2 Contributions

All existing logic-locking techniques are vulnerable. Thus, there is no provably-secure logic locking. Hence, in this paper, we first propose a provably-secure logic-locking technique:

1. We propose a novel logic-locking technique that we refer to as *TTLock*, which provably resists all known attacks, i.e., SAT [12], sensitization [16], and signal skew analysis attacks [15].

2. TTLock modifies the original logic cone by inverting the response to one protected input pattern, while an additional inversion introduced by TTLock restores the correct functionality only for the correct key.

3. Even though the TTLock logic can be identified via a signal-tracing attack, its removal will still leave the remaining logic different than the original one, thwarting removal attacks.

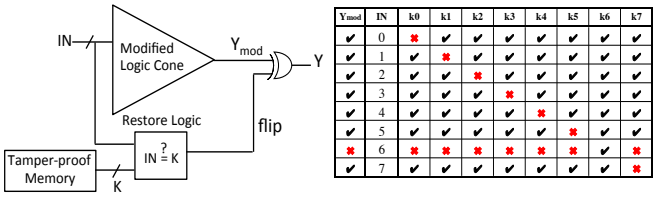
Furthermore, we introduce a novel locking approach, which we coin as *parametric locking*. As opposed to functional (logic) locking where keys are associated with design functionality, parametric locking ensures that the design/chip behaves as per its specifications (performance, power, etc.) only with the secret key in place; an incorrect key downgrades its parametric characteristics, leading to a reduced speed, increased power consumption, or degraded reliability. We elaborate on the objectives and challenges in parametric locking.

2. TTLOCK: CONSTRUCTION AND PROOF OF SECURITY

TTLock *modifies the design logic cone* to invert its output for a selected (protected) input pattern. The modification can be effected via logic gate insertions/replacements. The desired impact is an inverted output for only one input pattern corresponding to the correct key. The *restore unit* of TTLock then inverts the inverted output only for the correct key, thereby restoring the correct output. For any incorrect key, TTLock produces an inverted output for the protected input pattern. Both the key and the protected input pattern are the designer's secrets.

TTLock has the following properties:

- The modification applied to the logic cone is minimal to deliver maximal SAT attack resilience; the discriminating



Ymod	IN	k0	k1	k2	k3	k4	k5	k6	k7
✔	0	✖	✔	✔	✔	✔	✔	✔	✔
✔	1	✔	✖	✔	✔	✔	✔	✔	✔
✔	2	✔	✔	✖	✔	✔	✔	✔	✔
✔	3	✔	✔	✔	✖	✔	✔	✔	✔
✔	4	✔	✔	✔	✔	✖	✔	✔	✔
✔	5	✔	✔	✔	✔	✔	✖	✔	✔
✖	6	✖	✖	✖	✖	✖	✖	✔	✖
✔	7	✔	✔	✔	✔	✔	✔	✔	✖

Figure 3: Proposed TTLock architecture. The logic cone is minimally modified by inverting the response of one input combination w.r.t. original logic cone. The restore logic fixes this inversion for the correct key for the intended input combination and introduces a second inversion for all incorrect keys. The secret key is k_6.

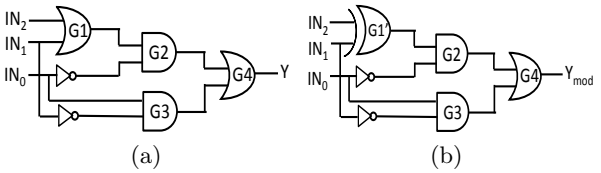


Figure 4: a) Logic cone in the original circuit. b) Modified logic cone in TTLock. $G1$ in logic cone is replaced with $G1'$ in the modified logic cone that inverts the output Y_{mod} for IN=6.

ability of each input pattern is limited to a single incorrect key (See Section 2.2).

- Any reverse-engineering attack on the original logic cone recovers the modified, and thus, the incorrect functionality. The secret key should be known to understand the modification in the logic cone, protecting against removal attacks (See Section 2.2).

2.1 Construction of TTLock

Figure 3 depicts the generic architecture of the proposed TTLock using a simple example that assumes a three-input logic cone protected by a three-bit TTLock key. We consider the logic cone shown in Figure 4(a). Without loss of generality, we make a simplifying assumption that the logic cone input size n equals the key-size; in this example, n=3. TTLock includes a restore logic and an XOR gate. The restore logic consists of a comparator, whose output is 1 iff the input matches the key. The table in the same figure illustrates the inverted outputs. Y_{mod} column shows the inversion for the protected input pattern; this inversion is fixed only for the correct key k_6. Every incorrect key introduces the second inversion for a distinct input pattern.

The modified logic cone (MLC) implementing the inversion in Figure 3 is shown in Figure 4(b). The MLC produces an inverted output for input pattern 6. When the correct key K_{corr}=k_6 is applied, the restore logic asserts its output for input pattern 6, thereby canceling the inversion injected into the logic cone for that pattern and producing the correct/desired output. When an incorrect key, say k_2, is applied and for the input pattern 6, the restore logic output is 0. Consequently, the inverted output of the MLC is retained, thereby producing the wrong output. With k_2 applied, and for the input pattern 2, the restore logic output is 1. Consequently, the correct output of the MLC is inverted, thereby producing the wrong output again.

2.2 Security analysis of TTLock

Threat model: Consistent with all the logic-locking techniques proposed so far, the attacker is assumed to have access to (1) reverse-engineered (locked) netlist and (2) a working chip, i.e., oracle, that has the key loaded in its memory.
SAT attack resilience: TTLock achieves resilience against SAT attack by always ensuring that the SAT attack encounters its worst-case scenario: In each iteration, a DIP can eliminate at most only one incorrect key. For example, as shown in Figure 3, each input pattern eliminates only one incorrect key, thus, requiring 7=2^3-1 iterations. SAT attack can also perform random restarts. For instance, consider the circuit in Figure 3. If it randomly chooses IN=6 (the protected input pattern), it can eliminate all incorrect keys at once. However, an attacker/SAT attack does not know the protected input pattern. Thus, it can only choose the DIP uniformly at random. Thus, the probability of SAT attack choosing the protected input pattern is $\frac{1}{2^n}$. For a design with a large number of inputs, say 80, this probability becomes negligible (i.e., 2^{-80}), ensuring the security of TTLock.
Removal attack resilience. In removal attack, an attacker identifies and removes the protection logic. Since the *flip* signal (the output of the protection circuit) is highly skewed towards 0, it can be identified by the SSA attack. However, the input pattern(s) for which the design produces corrupted outputs are unknown to the attacker as is the secret key. Any removal attack would recover the MLC and not the original design.
Sensitization attack resilience. In TTLock, all the bits of the key converge on the *flip* signal. Therefore, sensitizing any given key-bit through the *flip* signal to Y requires setting all the other key-bits to known values. As all the key-bits are unknown, all the key-gates in TTLock are pairwise secure. Thus, the sensitization attack cannot be launched on TTLock.

2.3 Frequently Asked Questions

Question 1. Would minor modification(s) to the original SAT-based attack algorithm make the proposed scheme vulnerable to other SAT-based attacks? For example, can an attacker can introduce an additional constraint of IN=KEY in the SAT attack algorithm in order to eliminate all incorrect keys in just one iteration?
Answer. It would be feasible to add the suggested modifications/constraints only if the locked netlist had structural/functional traces that an attacker could exploit to identify the protected pattern. From the analysis of the locked netlist alone, the attacker cannot extract the protected input pattern; one has to exercise the oracle to determine whether a particular pattern is the protected one. The probability of an attacker determining the protected input pattern is $\frac{1}{2^n}$.
Question 2. The proposed TTLock scheme requires modification of the original logic cone. Assuming the logic cone has $n = 64$ inputs, such modification requires the construction of a truth table with 2^{64} inputs. Is it feasible to work with such large truth tables?
Answer. The original logic can be modified for the protected input pattern (minterm) by first introducing additional logic that modifies the output of the function for this minterm, and then by resynthesizing the entire design. The resynthesis process blends the added logic with the original design, eliminating its traces, due to various optimiza-

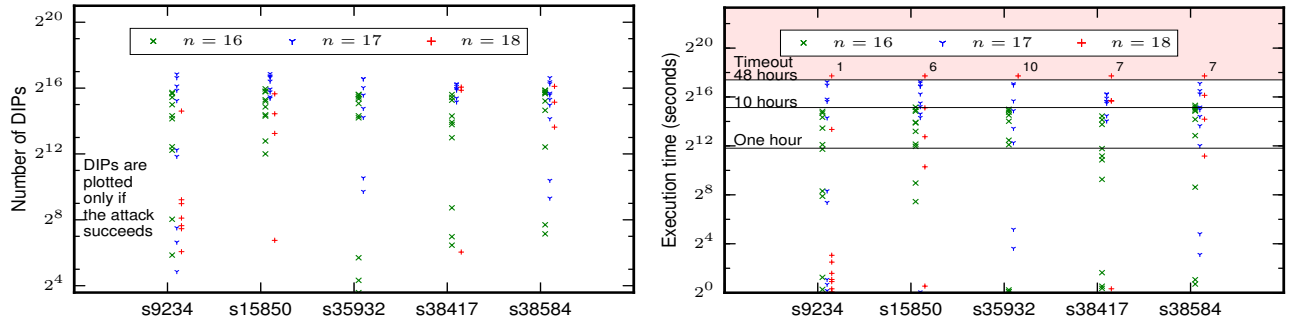


Figure 5: a) Number of DIPs required for the SAT attack [12], and b) execution time (seconds) for TTLock; n={16, 17, 18}. The timeout is 48 hours. Timeout occurs only for n=18. The number of timeout instances (out of 10) for each circuit is shown above the timeout line.

tion and transformation steps. Conventional logic synthesis tools have traditionally been handling designs with input size greater than 64, and they are re-purposed for our objectives.

Question 3. Is the amount of entropy introduced minimal as TTLock only targets one output port of the original design?

Answer. There exists a trade-off between the SAT attack resilience and the output entropy, which is measured in terms of Hamming distance between the output for the correct key and the output for incorrect keys. For maximum SAT attack resilience, the entropy must be minimized. To achieve higher entropy while retaining maximal SAT attack resilience, TTLock may be used in conjunction with high entropy logic-locking techniques such as RLL or FLL.

3. RESULTS

3.1 Experimental setup

In this section, we present experimental results to confirm the security expectations and assess the implementation overhead for the proposed TTLock technique. We run our experiments on the largest ISCAS'89 benchmark circuits. Each experiment is repeated ten times in order to improve the statistical significance. The area, power, and delay overhead is obtained using Cadence RTL Compiler. FreePDK 45nm library is used [17].

3.2 Security analysis

The resilience of TTLock is dictated by the key-size n. The number of DIPs required for the SAT attack to succeed, and the corresponding execution time values are presented in Figure 5 for n={16, 17, 18}.

Impact of key-size n. The expected number of DIPs required to break TTLock is 2^{n-1}. As shown in Figure 5(a), most of the blue entries (n=17) appear closer to 2^{16}, and most of the green entries (n=16) appear closer to 2^{15}. With the exception of a few cases where the attack may be fortuitously successful, TTLock thwarts the SAT attack by forcing the number of DIPs to be exponential in n. The SAT attack does not terminate for larger values of the key-size ($n \geq 18$); the probability of fortuitous attack success becomes exponentially smaller for larger values of n.

Execution time. As shown in Figure 5(b), the execution time of the SAT attack is proportional to the number of DIPs, although there is a slight variation of 3× to 4× across the benchmark circuits; the execution time grows exponentially in n. As noted earlier, most of the experiments with n=18 were aborted after 48 hours of execution.

Table 1: Comparative security analysis of logic-locking techniques against existing attacks. Vulnerability of a logic-locking technique to an attack is denoted by X. TTLock is secure against all attacks.

Attack	RLL [6]	FLL [7]	SLL [9]	Anti-SAT [14]	SARLock [13]	TTLock
Sensitization	X	X	✓	✓	✓	✓
SAT	X	X	X	✓	✓	✓
Removal	✓	✓	✓	X	X	✓

Table 2: APD overhead for TTLock.

Benchmark	s35932			s38417			s38584		
n	16	32	64	16	32	64	16	32	64
Area (%)	1.0	1.8	4.0	2.0	2.8	2.9	1.2	1.2	2.1
Power (%)	1.7	3.4	6.0	1.8	2.9	3.1	1.7	1.8	3.2
Delay (%)	1.2	1.2	1.4	0.2	0.2	0.1	0.8	0.6	0.5

3.3 Area, power, and delay (APD) overhead

Impact of key size n. While the security level increases exponentially in n for TTLock, the APD overhead must be accounted for in choosing n. As shown in Table 2, the APD overhead of TTLock is quite small. The average area, power, and delay overhead for n=64 is 3.0%, 4.1%, and 0.6%, respectively. TTLock comprises a single n-bit comparator, and the overhead increases linearly with n.

These APD results bode well for large-sized industrial circuits where the percentage overhead results are expected to be much smaller than those reported herein.

4. PARAMETRIC LOCKING APPLICATION: PERFORMANCE LOCKING

Till now, we have used logic locking to produce incorrect outputs on applying an incorrect key. We now leverage the logic locking technique to modify/protect the parametric behavior—power, delay, reliability, etc.—of the circuit. This is called parametric locking. Only on applying the correct key, the design exhibits a superior parametric behavior—for instance, high performance, low power, low-energy, highly reliable operations. On applying the incorrect key, the design exhibits a relatively poor parametric behavior. In this paper, we introduce an instance of parametric locking, namely, performance locking at the register-transfer (RT) level. Unlike functional locking, on applying an incor-

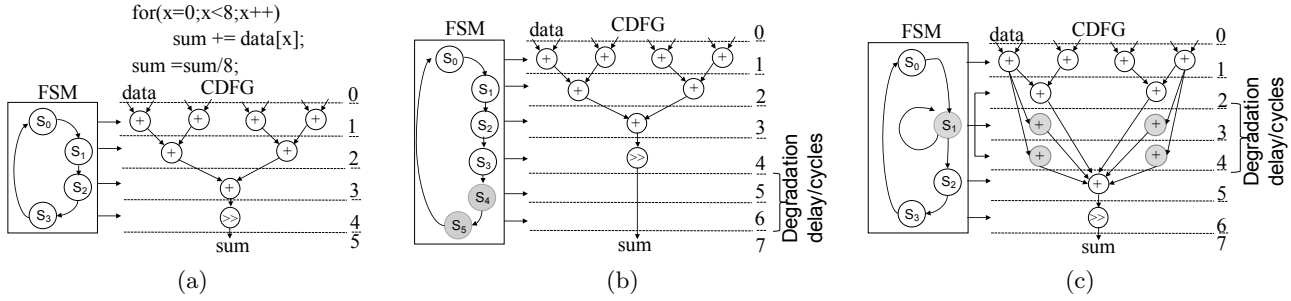


Figure 6: Performance locking. a) High-performance design with correct key. b) Performance locking using two wait states. c) Performance locking using two dummy computations.

rect key, parametric locking does not necessarily produce incorrect outputs but rather degrades the performance.

The section is organized as follows. We introduce the notion of performance locking with a motivating example. We then explain the business model, where performance locking will be useful, and the associated threat model. Security properties of performance locking are detailed next. Finally, outstanding challenges that need to be addressed to make performance locking feasible and effective are elaborated.

4.1 Performance locking at the RT-level

With an incorrect key applied, the design computes the final result by consuming an additional number of clock cycles. The application of the correct key provides a superior performance, i.e., a shorter execution time. In this work, we explain how this behavior can be implemented during high-level synthesis.

Motivational example: Consider the example of a design computing the average of the last eight numbers. The control data flow graph (CDFG) for this design with the correct key is shown in Fig. 6(a). The design requires four clock cycles. In the first three clock cycles, four, two and one addition operations are performed, respectively. At the final clock cycle, a three-bit right shift operation is applied on the data to get the average of eight numbers. Thus, we get the output at the start of fifth clock cycle, resulting in a latency of four clock cycles. We consider this CDFG as the representation of the high-performance design.

Consider the same design where the performance is degraded on applying an incorrect key. This can be performed in two ways, which are depicted in Fig. 6(b) and Fig. 6(c). In the first method shown in Fig. 6(b), two additional wait cycles are introduced to slow down the design. It can be observed that this has the effect of adding two redundant or null states (shown as shaded) in the finite state machine (FSM). Now, the latency of the design is increased from four to six clock cycles. In the second method depicted in Fig. 6(c), the same latency of six clock cycles is achieved. However, the final addition operation is performed during the fifth clock cycle, as opposed to the third clock cycle in the first method. During the third and fourth clock cycles, dummy computations (shown as shaded) are performed with the objective of adding further obscurity for the attacker; he/she faces the additional challenge of distinguishing the real computations from the dummy ones. Only one extra state is added; the FSM loops through this dummy state in the third and fourth clock cycles.

4.2 Business and threat models

Consider a company that intends to design and manufacture the same IC but has two sets of users. The first set of users can pay more than the second set and would naturally expect better IC performance. Currently, this type of business model is supported by speed binning, which is necessitated by unintended process variations. Parametric (performance) locking enables the companies to provide a similar feature intentionally, with security guarantees.

Threat model. The attacker is at the untrusted foundry or can be the end-user. He/she has black-box physical access to the IC with the superior performance. The objective of the attacker is to obtain the key to boost the performance of the ICs. To this end, he/she can buy an IC with high performance from the market, reverse engineer it, and obtain its netlist. Further, he/she can apply chosen inputs to another IC obtained from the market and collect the correct outputs.

4.3 Properties of performance locking

1. **Performance degradation ratio:** The effectiveness of performance locking is dictated by the performance ratio of the high-performance to low-performance design. Consider the same example, where the moving average of eight numbers is computed. The latency for the high-performance design, whose CDFG is shown in Figure 6(a), is four clock cycles. The latency for the low-performance designs, depicted in Figure 6(b) and Figure 6(c), is six clock cycles. Thus, the performance ratio is 1.5. To achieve higher degradation, one can introduce a larger number of wait cycles or insert a higher number of dummy computations, degrading the performance even further.

2. **Overhead:** The hardware added to perform performance locking must incur minimal overhead over the baseline high-performance design. The example presented above demonstrates that performance locking can be achieved at a minimal overhead. In the first method, only extra wait states are inserted, incurring a 31.67%, 35.21% and 1.1% overhead for the area, power, and timing respectively. In the second method, dummy computations are performed by reusing parts of the original design, thereby incurring an overhead of 147%, 150% and 27% in area, power, and timing, respectively. Here, the additional dummy operations lead to a higher power consumption. Here, for comparison the baseline design is the original design without any extra key logic. Thus, performance locking with added key logic leads to an overhead

presented above. However, since the considered example is small, the overhead is high. But, it will amortize for large-scale designs, as the number of flip-flops added to store the keys and the performance locking logic almost remain the same.

3. **Resilience against resynthesis attacks:** An attacker can obtain the high-performance CDFG by applying inputs and observing outputs of a functional IC. He/she can also obtain information about the resources used (number of adders, shift operations, etc.) and their connectivity using reverse engineering methods. He/she can then perform a performance-constrained synthesis for the obtained CDFG and the resource information; the performance constraint can be chosen as the number of clock cycles by which the CDFG has to complete its computation. In the previous example, it is four clock cycles. To this end, an attacker can use well-known resource-constrained synthesis algorithms [18]. Even though the worst-case complexity of those algorithms is NP-Hard, the algorithms obtain solutions much faster in reality. On obtaining the solution to the performance-constrained synthesis problem, an adversary resynthesizes the netlist to obtain the high-performance design. Currently, no defense techniques exist against such a resynthesis attack. This is the major impediment to developing provably-secure performance locking schemes.

5. CONCLUSION

In this paper, we attempt to widen the notion of logic locking. First, we follow the traditional path and propose a logic locking technique, TTLock, that is provably secure against all attacks such as sensitization, SAT, and removal attacks; all the previous logic locking techniques are susceptible to one or more of these attacks. TTLock modifies a logic cone by inverting the output for a protected input pattern and restores this inversion to recover the correct functionality for only the correct key. TTLock ensures provably-secure protection against SAT attacks, while the modification of the logic cone constitutes a strong defense against removal attacks.

Then, we introduce the notion of parametric locking, where the design specifications such as power, performance, and/or reliability may be locked, as opposed to the functionality. In parametric locking, an incorrect key still produces correct results but downgrades the power, performance, and/or reliability of the chip. We present the threat model and properties for parametric locking, and in particular, performance locking. The challenge for performance locking is to achieve increased performance degradation for incorrect keys and offer resilience against resynthesis attacks, without incurring a high overhead.

Acknowledgement. This work was supported by the New York University/New York University Abu Dhabi (NYU/ NYUAD) Center for Cyber Security (CCS).

6. REFERENCES

[1] "Defense Science Board (DSB) study on High Performance Microchip Supply," 2005, [March 16, 2015]. [Online]. Available: www.acq.osd.mil/dsb/reports/ADA435563.pdf

[2] R. S. Chakraborty and S. Bhunia, "Security against Hardware Trojan through a Novel Application of Design Obfuscation," *IEEE/ACM International Conference on Computer-Aided Design*, pp. 113–116, 2009.

[3] M. Rostami, F. Koushanfar, and R. Karri, "A Primer on Hardware Security: Models, Methods, and Metrics," *Proceedings of the IEEE*, vol. 102, no. 8, pp. 1283–1295, 2014.

[4] SypherMedia, "Syphermedia library," http://www.smi.tv/syphermedia_library_circuit_camouflage_technology.html, [April 22, 2016].

[5] R. W. Jarvis and M. G. McIntyre, "Split Manufacturing Method for Advanced Semiconductor Circuits," *US Patent 7,195,931*, 2007.

[6] J. A. Roy, F. Koushanfar, and I. L. Markov, "Ending Piracy of Integrated Circuits," *IEEE Computer*, vol. 43, no. 10, pp. 30–38, 2010.

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

[8] R. S. Chakraborty and S. Bhunia, "HARPOON: An Obfuscation-Based SoC Design Methodology for Hardware Protection," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 28, no. 10, pp. 1493–1502, 2009.

[9] M. Yasin, J. Rajendran, O. Sinanoglu, and R. Karri, "On Improving the Security of Logic Locking," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 9, pp. 1411–1424, 2016.

[10] A. Baumgarten, A. Tyagi, and J. Zambreno, "Preventing IC Piracy Using Reconfigurable Logic Barriers," *IEEE Design & Test of Computers*, vol. 27, no. 1, pp. 66–75, 2010.

[11] S. M. Plaza and I. L. Markov, "Solving the Third-Shift Problem in IC Piracy With Test-Aware Logic Locking," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 6, pp. 961–971, 2015.

[12] P. Subramanyan, S. Ray, and S. Malik, "Evaluating the Security of Logic Encryption Algorithms," *IEEE International Symposium on Hardware Oriented Security and Trust*, pp. 137–143, 2015.

[13] M. Yasin, B. Mazumdar, J. J. Rajendran, and O. Sinanoglu, "SARlock: SAT Attack Resistant Logic Locking," *IEEE International Symposium on Hardware Oriented Security and Trust*, pp. 236–241, 2016.

[14] Y. Xie and A. Srivastava, "Mitigating SAT Attack on Logic Locking," *International Conference on Cryptographic Hardware and Embedded Systems*, pp. 127–146, 2016.

[15] M. Yasin, B. Mazumdar, O. Sinanoglu, and J. Rajendran, "Security Analysis of Anti-SAT," *IEEE Asia and South Pacific Design Automation Conference*, pp. 342–347, 2016.

[16] J. Rajendran, Y. Pino, O. Sinanoglu, and R. Karri, "Security Analysis of Logic Obfuscation," *IEEE/ACM Design Automation Conference*, pp. 83–89, 2012.

[17] J. E. Stine, I. Castellanos, M. Wood, J. Henson, F. Love, W. R. Davis, P. D. Franzon, M. Bucher, S. Basavarajaiah, J. Oh *et al.*, "FreePDK: An Open-Source Variation-Aware Design Kit," *IEEE International Conference on Microelectronic Systems Education*, pp. 173–174, 2007.

[18] D. D. Gajski, N. D. Dutt, A. C.-H. Wu, and S. Y.-L. Lin, *High-level Synthesis: Introduction to Chip and System Design*. Norwell, MA, USA: Kluwer Academic Publishers, 1992.