Week 1

IP Protection

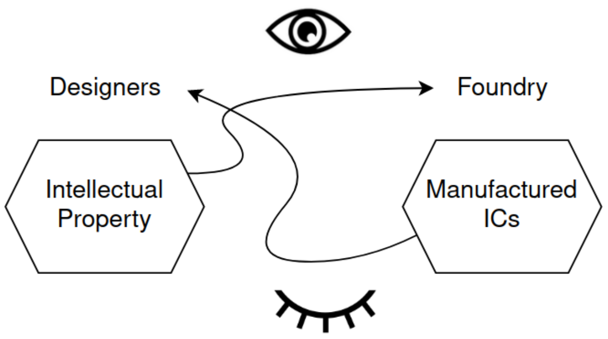
References:

Active Hardware Metering for Intellectual Property Protection and Security[1], Evolution of Logic Locking[2], Ozgur Sinanoglu[3]

Purpose of the week:

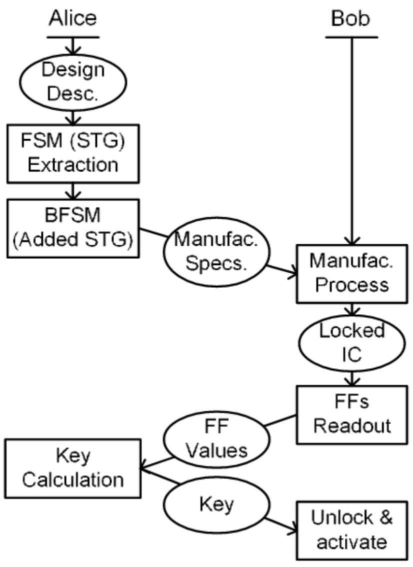
To get acquainted with the types of problems in protection of semiconductor ICs.

Brief:



Hardware pervades us, IoT to embedded systems to companion desktop/handheld computers, all are controlled by ICs manufactured in concentrated foundries holding most of the designing companies as clients. This leads to asymmetry in resource sharing, foundries have access to the IP of designing clients, but designers don't have control over how the foundry (or later testing phases) takes care of their design. Copyright infrigement against softwares have received huge attention, compared to that hardware protection doesn't level right.

Active Hardware Metering:



based on inherent unclonable invariability of ICs.

Capability to lock each IC, lock designed during designing, is realised inherently upon manufacturing. Has the option to remotely disable the IC.

This is performed via Boosted Finite State Machine implementation, which are powered by unique IDs. Consists of states that leads to "power-up" (back to normal IC behavior) upon provision of some fixed inputs, and because only the designer knows the transition table, it's not possible with finite resources to crack that input with high dimensional input space.

Authors of [1] utilises:

Week 1

i. uniqueness of ICs due to manufacturing variations

ii. structural manipulation of ICs can still lead to same behaviour

for implementing active metering, by interwinding both the points as unclonable IDs of each chip into their FSM, renamed as BFSM.

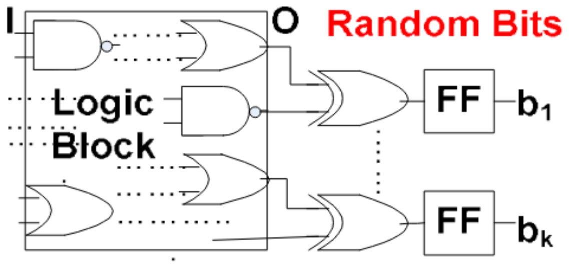
This is done as the above diagram shows, upon manufacturing the ICs, each being unfunctional initially, the foundry has to read the FlipFlop values of each IC (can be read non-destructively?1?) and send back to the designer, which then would observe from the transition table which key that acts as input to the IC can get it "powered up". This key is sent back to the foundry to activate IC.

Now to generate the variability-based ID,

- It had been proposed to add additional circuitry, which on manufacturing variations would generate unique random IDs, but this can be easily enough removed/tampered.
- Loftstrom et al. proposed a method for mismatching the devices based on changing the threshold of the circuits by placing the impurity of random dopant atoms.
- Maeda et al. proposed implementing the random IDs on poly-crystalline silicon thin film transistors
- Su et al. have proposed a technique to generate random IDs by using the threshold (?what threshold?) mismatches of two NOR gates that are positively feeding back each other
- Using PUFs which are widely used in authentication, which map a set of physical properties to a response, example maybe using bias voltages of transistors, or physical locations of wire lines or delays in critical paths. PUFs are unique, since process variations cause significant delay differences among ICs coming from the same mask. But this rises the need of a database of IC matching of challenges to responses.
- The Authors of [1] propose of including security features not in fabrication stage, but during behavioral synthesis stage.

Active metering is integrated into the *standard synthesis flow*, and is low overhead, generalizable (can be implemented on structures common to all designs), and resilient against attacks.

Boosted FSM:



Assumptions: Existence of RUB (Random Unique Block), responsible for generation of unique IDs for each chip. Desirable that they be time-invariant.

Integration to the IC's sequential design's FSM, assume that it had *m* distinct states. Assume that state of the FSM is stored in *k* one-bit flip-flops (FFs). *k* FFs represent 2^k states, out of which *m* are of the original IC, rest are don't care. The metering system adds extra part to the FSM, with transitions possible from added states to the reset state q_0 of the original design.

Week 1

Choose the "power-up" state to be a function of manufacturing variability and thus be unique in each instance of IC. Choose 2^k - *m* >> *m*. This ensures that upon powering on of the IC, its initial state is highly likely to be in one of the states of the Boosted FSM.
If the IC gets powered up to q_{a0} state (by the RUB), then to move to q_0 state, the foundry needs to enquire the designer for referring the transition table, else it would be a problem of exponential complexity (on the number of FFs).

An uninformed user who does not have the information about the transition table (e.g., foundry), can readout the data about the initial added state q_{a0}, but this information is not sufficient for finding the sequence of primary input combinations to arrive at the reset state q_0. However, the person who has the information about the structure of the STG, upon receiving the correct state, would exactly know how to traverse from this locked state to q_0.

Since the power-up state is unique for each IC, sequence of inputs (key) that traverse the power-up state to the reset state is also specific to each IC. One needs to store the key which performs the traversal at the power-up state on each chip.

Remote Disabling:

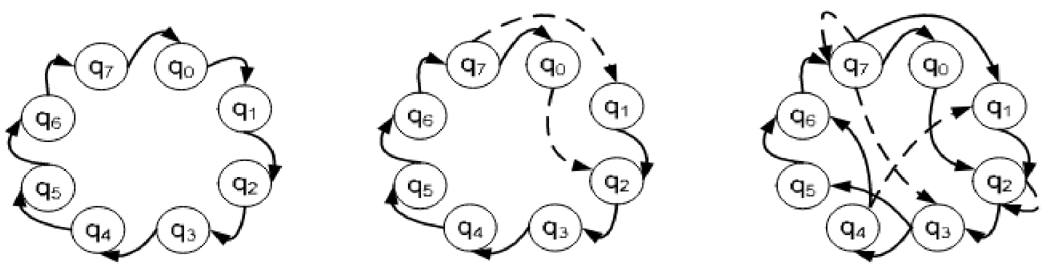
The designer can save the RUBs and keys of all ICs activated. Suppose some provision is created that she can monitor the chips remotely, then she could have added some transitions in the original FSM back to the Boosted states, rendering them useless again if need be.

(?Abstracting away from circuits and treating them as FSM capable of transition from any state to any other specified state is hard for me right now, any verification that this is always possible?)

The Authors used the Birthday paradox to prove that no 2 ICs can feasibly have the same power-up state (or the RUB). (?How is it ensured that power-up states are the same everytime?)

TODO: Analyse SRAM in MOS perspective how powering on makes it attain the same state, identifying it uniquely.

There is also a need to ensure that the keys are distinct in all parts of their sequences, or there is a very small shared subsequence between different keys. This is granted by making multiple paths on the graph from each of the states to the reset state.



(a) Ring counter (b) Reconnecting a state (c) Adding a few edges

Week 1

Possible Attacks

Assumptions: The adversary knows all the concepts of the proposed hardware metering scheme, has the complete knowledge of the design at all levels of abstraction provided to the foundry (e.g., logic synthesis level netlist, and physical design GDS-II file, but no behavioral specification), can simultaneously observe all signals (data) on all interconnects and flip-flops (FFs), and can measure, with no error, all timing characteristics of all gates in the ICs.

Q. Do we assume that adversary know about q_0? If yes how? If no, then it's impossible to know if it's on the right track, no?
A. Yes it is, look at another activated IC's reset state, all are the same.

(?Why not try sequential implementation in NAND flash somehow?)

(?How are power-up states related to RUBs?)

i. Brute Force: Applying different input sequences in hope to reach the reset state Scanning works by reading out the FF values for a few ICs and storing them. The FFs in the current IC are then monitored for the existence of a common state with the stored ones. In case a state that was read in the previous ICs is reached, the adversary uses the same key for traversal to the reset state.

ii. Reverse Engg of FSM: Seperate original states from added states

iii. Combinational redundancy removal: remove the combinational logic blocks not necessary for correct behavior of circuit

iv. RUB emulation (Invasive): Implement hardware having identical functional and timing characterstics as an unlocked IC

v. Initial power-up Capture And Replay (Invasive): Load FFs of other ICs with same power-up state of a known unlocked IC

vi. Initial Reset CAR (Invasive): Load FFs with Reset state

vii. Control signals CAR: learn the control signals and emulate them, to bypass the FSM entirely

viii. Differential FF activity measurement: investigate activities of FF in unlocked IC, for the same input, and eliminate FFs having different values

ix. Creating identical IC using selective IC release: Similar RUB ICs are reported, getting back their keys. So by birthday paradox, one of the ICs is bound to work for unreported ICs. Closeness of characterstics by closeness of power-up state.

Possible Defenses:

Black hole states and Trapdoor black holes: a state with no way back to original FSM, and a state with a long specific route back to reset.

Creation of Specialised Functional FSM: Where the reset itself is a function of RUB!

Obfuscation of state activities and state encodings: Can't differentiate between original and boosted.

Evolution of Logic Locking:

Against: Piracy, Reverse Engineering, Hardware Trojans;
Addressed by: IC Metering, Watermarking, IC Camouflaging, Split Manufacturing, Logic Locking

Watermarking

Watermarking of IP must remain functionally correct, and prove the ownership of the IP. For non-intrusive Watermarking:

- Optimization problem that's difficult to solve, with it's solution space capable to handle a digital watermark
- Well-defined interpretation of solutions of the problem as an IP
- Existing algorithms that solve the problem without watermark involved
- Protection requirements include: forgining as hard as recreating, tampering is provable

Generic Constraints based Watermarking Procedure:

During the creation of the IP in several stages, watermark each stage with a set of "constraints", then use pre- and post-processing of i/o to disproportionately satisfy large number of such constraints. These constraints aren't revealed.

Generic Signature Verification:

Must show the constraints to others, and calculate P_C that is probability the constraints met by coincidence, which would need to be low to confirm ownership.

Attacks:

Ghosting - adversary proves his own signature through showing his constraints meet P_C threshold.
Tampering - removing the owner's watermark
Forging - put owner's watermark in other not-owned IPs

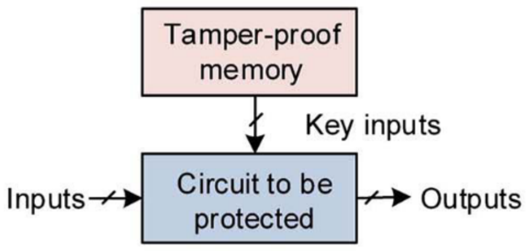
Week 1

Going to physical attacks possible: Physical attacks consists of multiple ways to attack an IC, including probing, fault injection, timing, power analysis, and electromagnetic analysis. They are categorised into invasive and non-invasive (depending on whether IC is physically tampered with)

Logic Locking

Cryptographic notion of a **lock**: an attacker that does not have infinite computational power should not be able to unlock the IC without the knowledge of a key.

A locked circuit has *key* inputs that are driven by on-chip tamper-proof memory. The additional logic may consist of XOR gates or LUTs (?why not other states just like metering did, for sequential circuits?). The IC needs to be activated by loading the secret key onto the tamper-free chip memory.



Earlier, Logic Locking focused on **best locations for inserting key gates**

- random
- fault analysis-based
- strong-interference-based

SAT attacks emerged and changed the paradigm to anti-SAT logic locking (including SARLock, Anti-SAT, TTLock, SFLL), but which became vulnerable back to the previous attacks such as removal attacks.

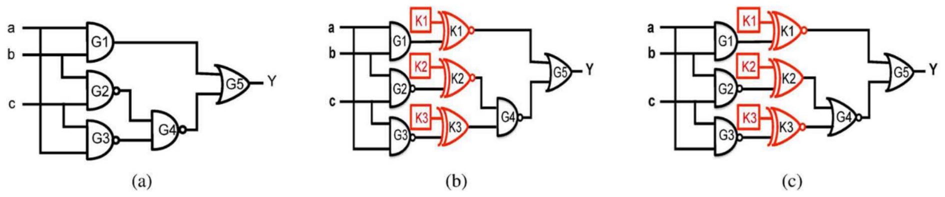


Fig. 3: Logic locking using XOR/XNOR gates [15]. a) An example circuit: majority of three inputs. b) Circuit locked using XOR/XNOR key gates. The correct key is value is 110. c) Locked circuit with inverters absorbed by the key gates. The correct key value is still 110.

Attacks

Assumptions: No-one except the designer is trusted. Adversary has access to netlist AND a functional IC. The only unknown is the key, which is a bit-vector.

Algorithmic: Exploit weakness of logic locking algorithm, brute forced, exact attacks. Includes sensitization attack, SAT attack, circuit partitioning attack

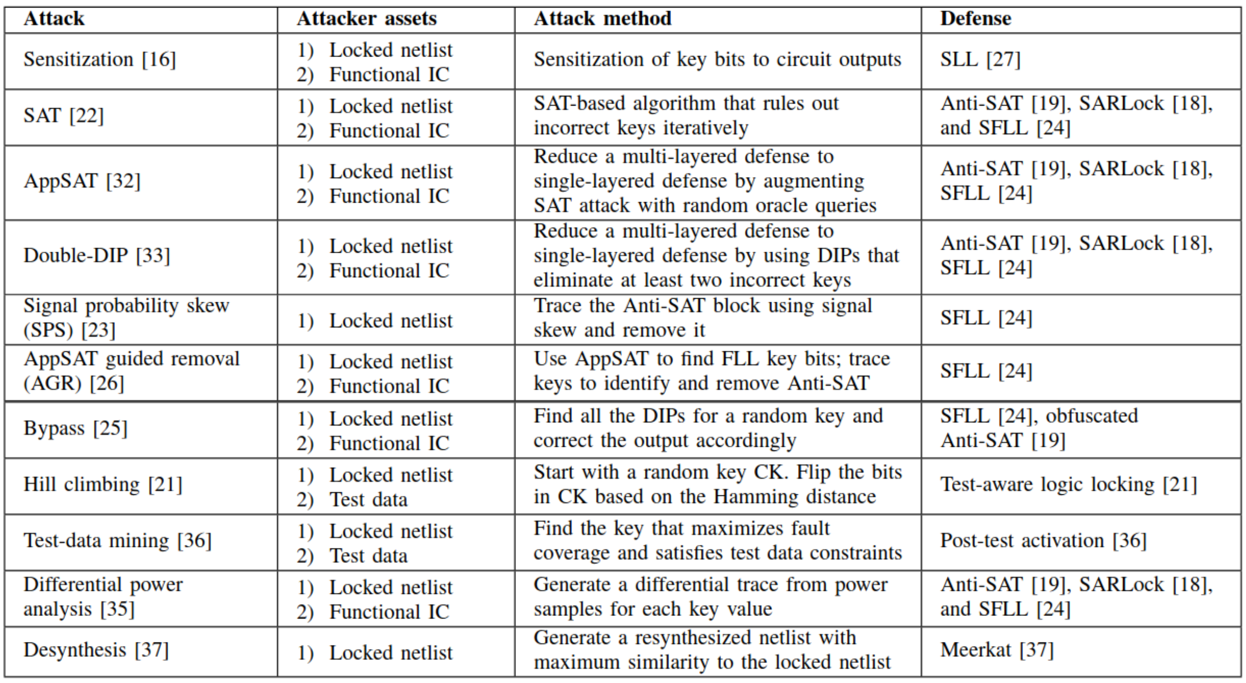
Approximate: Extract an approximately same netlist as original. Includes AppSAT, Double-DIP

Structural: Bypass protection logic. Includes signal probability skew attack, AppSAT guided removal attack, Bypass attack

Week 1

Side-Channel: Exploit covert physical channels like power, timing, test-data. Includes differential power analysis attack, desynthesis attack.

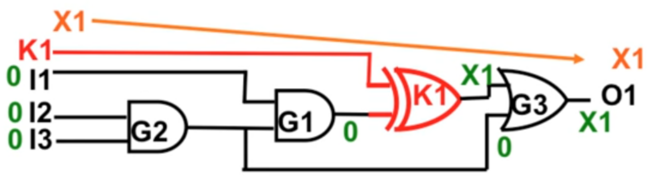
TABLE II: A summary of the attacks against logic locking.



Attack	Attacker assets	Attack method	Defense
Sensitization [16]	1) Locked netlist 2) Functional IC	Sensitization of key bits to circuit outputs	SLL [27]
SAT [22]	1) Locked netlist 2) Functional IC	SAT-based algorithm that rules out incorrect keys iteratively	Anti-SAT [19], SARLock [18], and SFLL [24]
AppSAT [32]	1) Locked netlist 2) Functional IC	Reduce a multi-layered defense to single-layered defense by augmenting SAT attack with random oracle queries	Anti-SAT [19], SARLock [18], SFLL [24]
Double-DIP [33]	1) Locked netlist 2) Functional IC	Reduce a multi-layered defense to single-layered defense by using DIPs that eliminate at least two incorrect keys	Anti-SAT [19], SARLock [18], SFLL [24]
Signal probability skew (SPS) [23]	1) Locked netlist	Trace the Anti-SAT block using signal skew and remove it	SFLL [24]
AppSAT guided removal (AGR) [26]	1) Locked netlist 2) Functional IC	Use AppSAT to find FLL key bits; trace keys to identify and remove Anti-SAT	SFLL [24]
Bypass [25]	1) Locked netlist 2) Functional IC	Find all the DIPs for a random key and correct the output accordingly	SFLL [24], obfuscated Anti-SAT [19]
Hill climbing [21]	1) Locked netlist 2) Test data	Start with a random key CK. Flip the bits in CK based on the Hamming distance	Test-aware logic locking [21]
Test-data mining [36]	1) Locked netlist 2) Test data	Find the key that maximizes fault coverage and satisfies test data constraints	Post-test activation [36]
Differential power analysis [35]	1) Locked netlist 2) Functional IC	Generate a differential trace from power samples for each key value	Anti-SAT [19], SARLock [18], and SFLL [24]
Desynthesis [37]	1) Locked netlist	Generate a resynthesized netlist with maximum similarity to the locked netlist	Meerkat [37]

Sensitization Attack

Sensitisation of net *w* to an output *o* implies *w* propagates to *o*, in true or complementary form. Adversary analyses input patterns that sensitize individual key bits *to primary outputs*. These patterns are applied to functional IC, and via it's responses, he obtains the key bits.

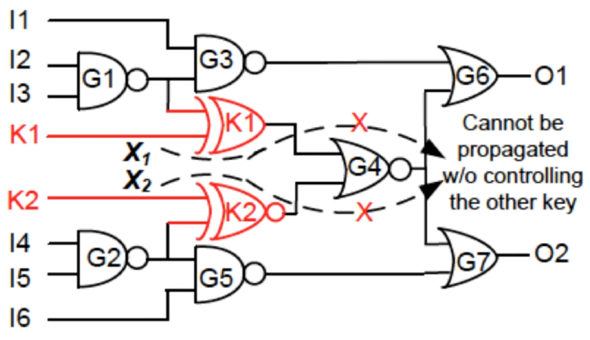


Example in the following toy-netlist, the input pattern that sensitize the key bit is [0 0 0], having K1 feedforward from the XOR and the OR gates at the end. Finding such input patterns with a known netlist is the attack.

Strong Logic Locking Defense

Make it difficult to sensitise the key bits directly to the output. This is done by making key-lines interfere with each other. This leads to the need of attacking interfered bits alltogether instead of divide and conquer.

This is measured using *clique-size*, number of key-gates connected to each other via non-mutable edges.



Week 1

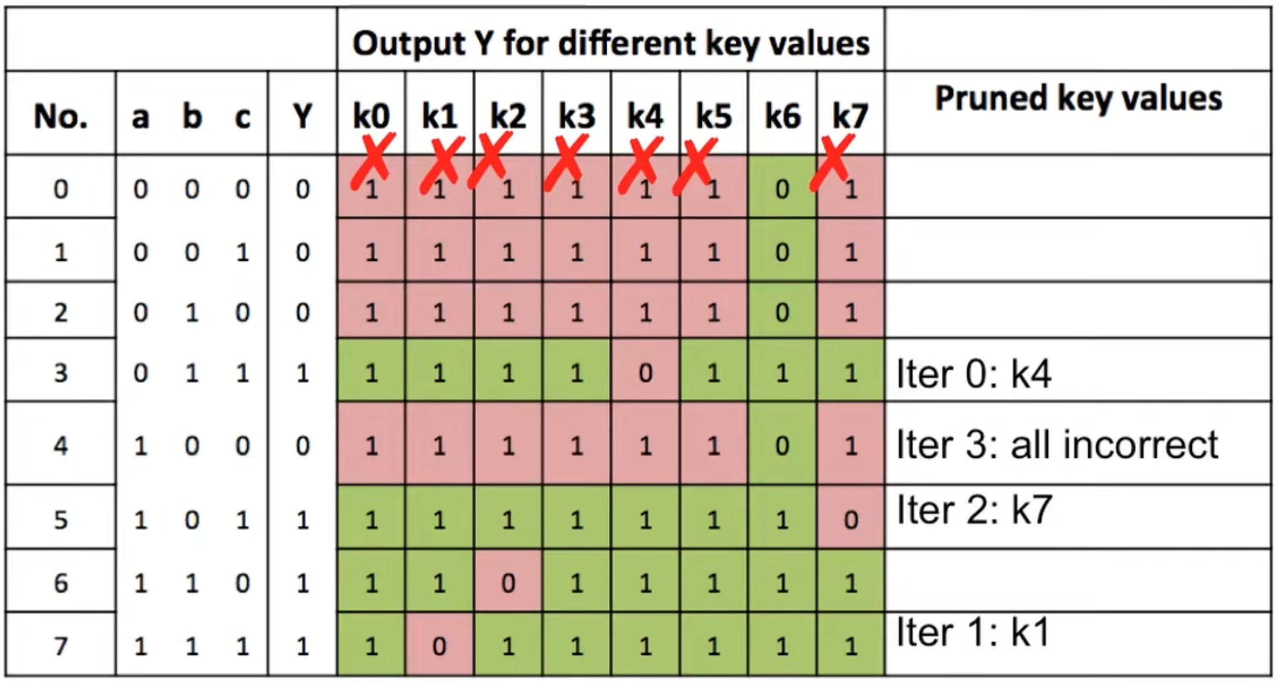
Satisfiability based SAT Attack

Prune key-space iteratively using DIPs (Discriminating Input Pattern). In this, the locked netlist is converted to *Conjunctive Normal Form* (a PoS form).

Elaboration:

The SAT solver takes in the CNF and goes through the input space in search of a DIP, an input pattern for which there are different outputs for 2 or more different key patterns. This leads to pruning away of those keys whose output the oracle IC disagrees with.
If there remains no more DIP, which means all inputs lead to a single output over the left over keys, then all those keys are the actual keys (ideally only one key remains).

Power of SAT attack lies on how powerful DIPs are in partitioning the key space.



					Output Y for different key values								
No.	a	b	c	Y	k0 ✗	k1 ✗	k2 ✗	k3 ✗	k4 ✗	k5 ✗	k6	k7 ✗	Pruned key values
0	0	0	0	0	1	1	1	1	1	1	0	1	
1	0	0	1	0	1	1	1	1	1	1	0	1	
2	0	1	0	0	1	1	1	1	1	1	0	1	
3	0	1	1	1	1	1	1	1	0	1	1	1	Iter 0: k4
4	1	0	0	0	1	1	1	1	1	1	0	1	Iter 3: all incorrect
5	1	0	1	1	1	1	1	1	1	1	1	0	Iter 2: k7
6	1	1	0	1	1	1	0	1	1	1	1	1	
7	1	1	1	1	1	0	1	1	1	1	1	1	Iter 1: k1

Anti-SAT paradigm defense

Come up with logic that are least distinguishing to the key space. The Author says that ideally each DIP eliminates *one* key value, with other keys' and that DIP output together matching with the oracle.

(?If this is done, doesn't such locked IC becomes nearly the original IC for any key?)

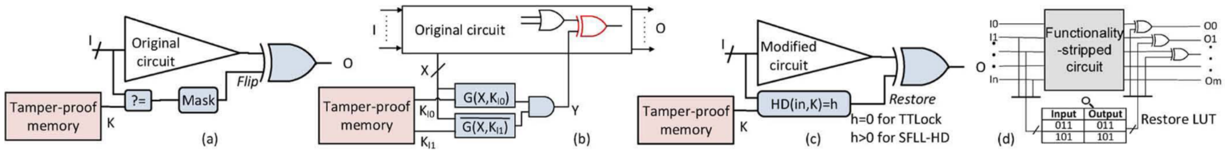


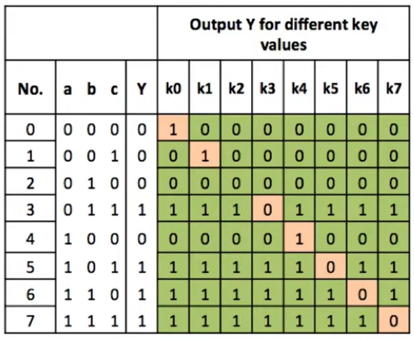
Fig. 5: SAT attack resilient logic locking techniques: a) SARLock [18], b) Anti-SAT [19], c) SFLL-HD [24]/TTLock [30], and d) SFLL-flex [24].

Week 1

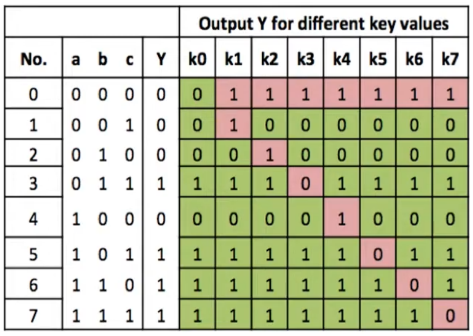
Right now, Logic Locked circuits have common keys apparently, all ICs are designed the same, so all should have the same key, no uniqueness. We can pick up ideas from active metering here.

SARLock:

Build a structure around the main logic cone, that compares the input and the key inserted, if they are the same (somehow make both of them the same dimensional), then output 1, which is XORed with the output of the logic cone. There's a mask in between this line to the output that flips 1 back to 0 if the key is correct. Obviously the mask has the secret key hardcoded.



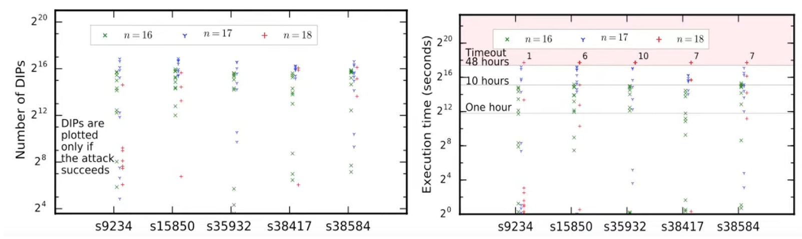
					Output Y for different key values							
No.	a	b	c	Y	k0	k1	k2	k3	k4	k5	k6	k7
0	0	0	0	0	1	0	0	0	0	0	0	0
1	0	0	1	0	0	1	0	0	0	0	0	0
2	0	1	0	0	0	0	0	0	0	0	0	0
3	0	1	1	1	1	1	1	0	1	1	1	1
4	1	0	0	0	0	0	0	0	1	0	0	0
5	1	0	1	1	1	1	1	1	1	0	1	1
6	1	1	0	1	1	1	1	1	1	1	0	1
7	1	1	1	1	1	1	1	1	1	1	1	0



					Output Y for different key values							
No.	a	b	c	Y	k0	k1	k2	k3	k4	k5	k6	k7
0	0	0	0	0	0	1	1	1	1	1	1	1
1	0	0	1	0	0	1	0	0	0	0	0	0
2	0	1	0	0	0	0	1	0	0	0	0	0
3	0	1	1	1	1	1	1	0	1	1	1	1
4	1	0	0	0	0	0	0	0	1	0	0	0
5	1	0	1	1	1	1	1	1	1	0	1	1
6	1	1	0	1	1	1	1	1	1	1	0	1
7	1	1	1	1	1	1	1	1	1	1	1	0

TTLock:

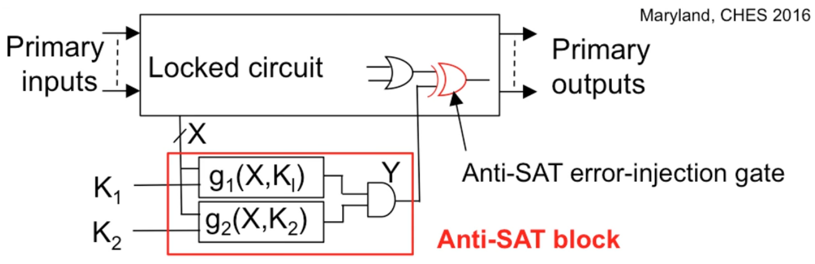
Similar as above, just the mask is inbuilt into the logic cone. For only one input, flip the output of the logic cone, this input corresponds to the correct key.



Anti-SAT:

Based on small error injection in the logic cone, for less effective DIPs. Here, two blocks g_1 and g_2 are complementary only when the correct key K is applied, leading to output of AND being 0, and error injection gate becomes a buffer.

If a wrong key is applied, for a designable g, $|\text{onset}| = p$, where p is the number of inputs for which output of g_1 and g_2 are the same.



When p nears 1 (for AND gate suppose) or 2^n-1 (for NAND gate suppose), the onset value approaches extrema, thus maximum number of inputs behavior would remain the same, of either being corrupted to the output, or not corrupted. In these extrema onset values, SAT attack is not successful.

Note that such "anti" SAT blocks are inherently such that the output is usually either correct or inverted, thus the IC is technically nearly always behaving correctly (or invertedly correctly). This doesn't adhere with the security principle of no information leakage to unauthorised users. There doesn't seem to be any answer to this right now.

Now because onset needs to be extrema in the above blocks, other attacks like *signal probability skew attack* can come in. It looks for signals that are skewed and converging, for different input patterns. Here for suppose g being AND, g_1 would be skewed to 0.5, and g_2 would be to -0.5, both converging to an AND gate. The effectiveness of *signal probability skew attack* increases as key size increases, because of higher proportion of inputs leading to an extrema output of *g*.

Stripped Functionality Logic Locking

It thwarts all known attacks in logic locking, by enabling tradeoff between SAT attack and removal attacks. It's based on "strip and restore", where some of the chip's functionality is restored only in form of the secret keys in on-chip tamper-proof memory.

SFLL-HD

This mechanism helps to save ${}^{k}C_{h}$, where *h* is some Hamming distance away from the actual secret key. Only *k*-bit secret key is saved in a tamper-proof memory, a single comparator along with Hamming distance compute logic, with increasing distance, the number of protected patterns increase binomially, having the attack's resilience drecrease logarithmically with increasing number of protected patterns.

(?How are tamper-proof memory implemented, and do they already contain the secret key or you have to insert in there as a user?)

SFLL-flex

A specified set of input patterns needs to be protected, which is allowed by flex, by compactly representing these patterns using a small set of input cubes. These are stored on-chip LUTs.

Piecing all together:

I read about Active Metering, Watermarking, and Logic Locking this and the previous week. There are a few questions laying around in the text that may need some answers. What I concluded is there are opportunities to mix-up different defenses, that hasn't been mentioned in the literature, such as using the sequential logic in logic locking, using watermarking identification for unique keys, and trying out stripping functional logic in active metering.