SAT-attack Resilient Sequential Locking

Seetal Potluri

Chair for Processor Design, Technical University of Dresden

October 5, 2018

Outline

Motivation

- SAT-attack is able to succesfully decrypt combinational-logic-encryption keys[1] [2]
- Real-life digital circuits are sequential in nature, so some of the combinational inputs are driven by flip-flops
- Therefore, means of launching SAT-attack : DFT

[1] J. A. Roy et al, "EPIC: Ending Piracy of Integrated Circuits", DATE 2008

[2] P. Subramanyan et al, "Evaluating the Security of Logic Encryption Algorithms", HOST 2015.

Attack model

- The encryption key is stored in tamper-proof memory, so not available;
- The attacker has access to layout and mask information from a malicious foundry. The gate-level netlist can be reverse engineered from this; and
- The attacker has access to an activated IC on which to apply input patterns and observe outputs. This could be obtained by purchasing an activated IC from the open market[3].

[3] P. Subramanyan et al, "Evaluating the Security of Logic Encryption Algorithms", HOST 2015

Access model

- The attacker applies inputs to the circuit through the DFT architecture;
- In general, the IC vendor keeps test (scan) mode alive even after testing, for the purpose of debugging customer returns [4] [5] (customer returned chips, are debugged for yield learning);
- Another reason why scan is not deactivated, so that customer can apply his test patterns and test for quality themselves.
- Mode of DFT operation for launching SAT-attack: EDT-BYPASS.

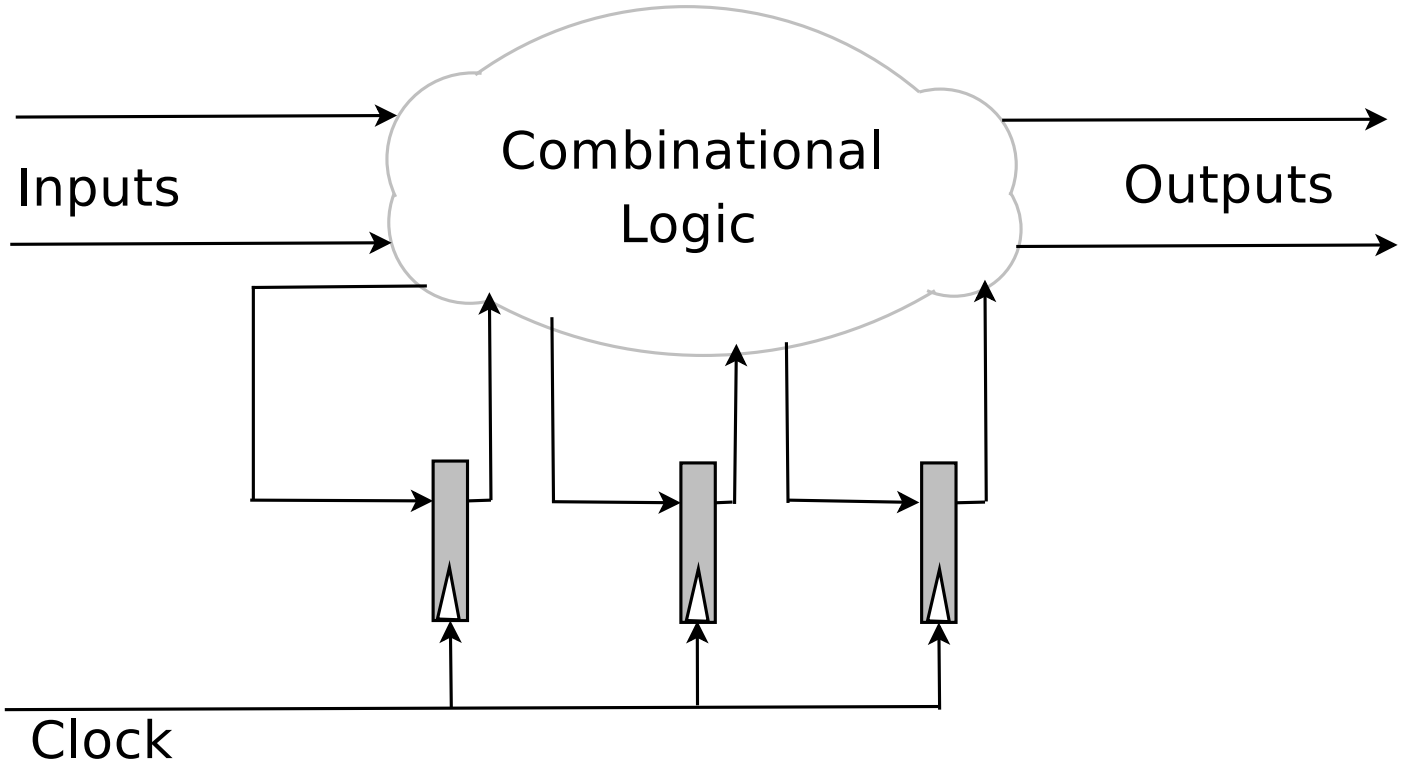
[4] S. Mitra et al, "Robust system design with built-in soft-error resilience", IEEE Computer, 2005.

[5] A. Carbine et al, "Pentium Pro Processor Design for Test and Debug", ITC 1997

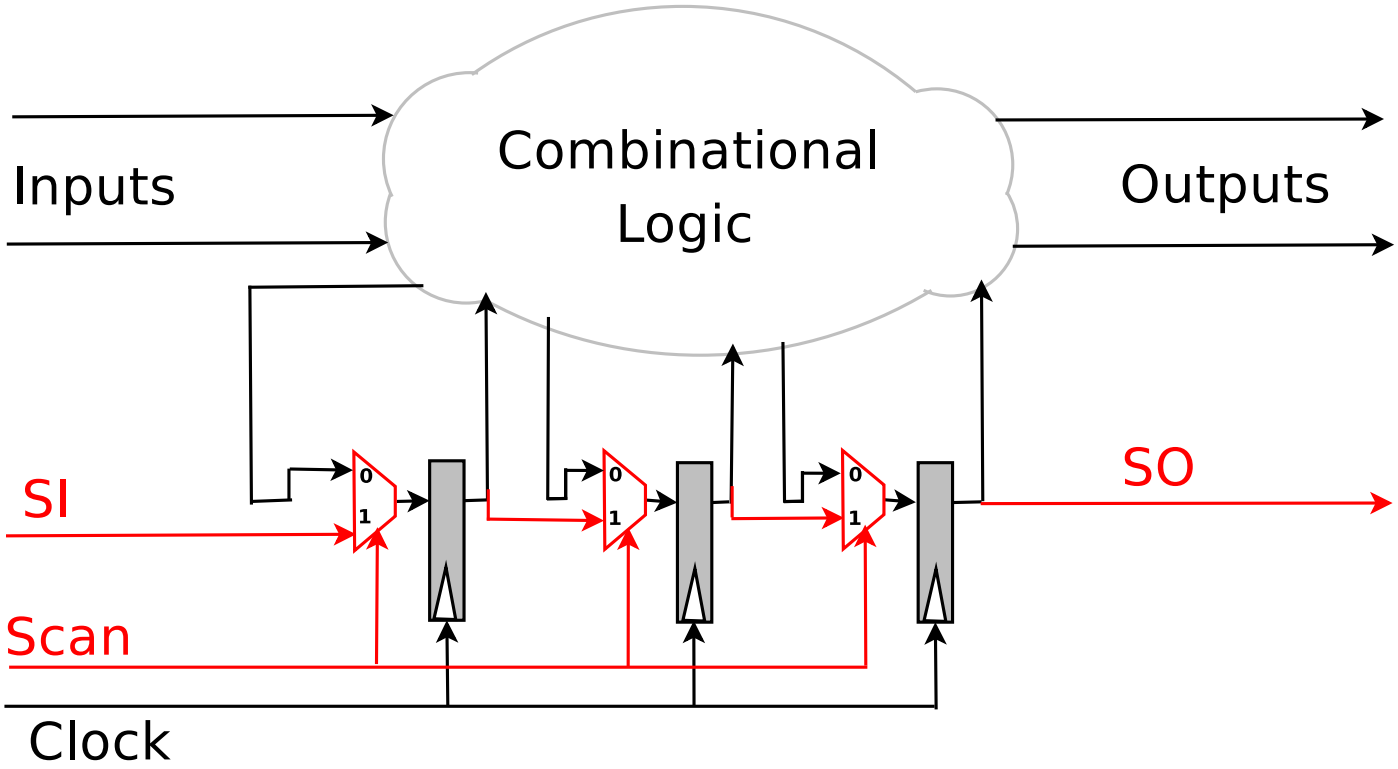
Design for Testability (DFT)

- Prevalently used to improve controllability/observability
- Industry DFT standard: Scan
- Industry DFT Compression standard: Embedded Deterministic Test (EDT)

Normal Circuit

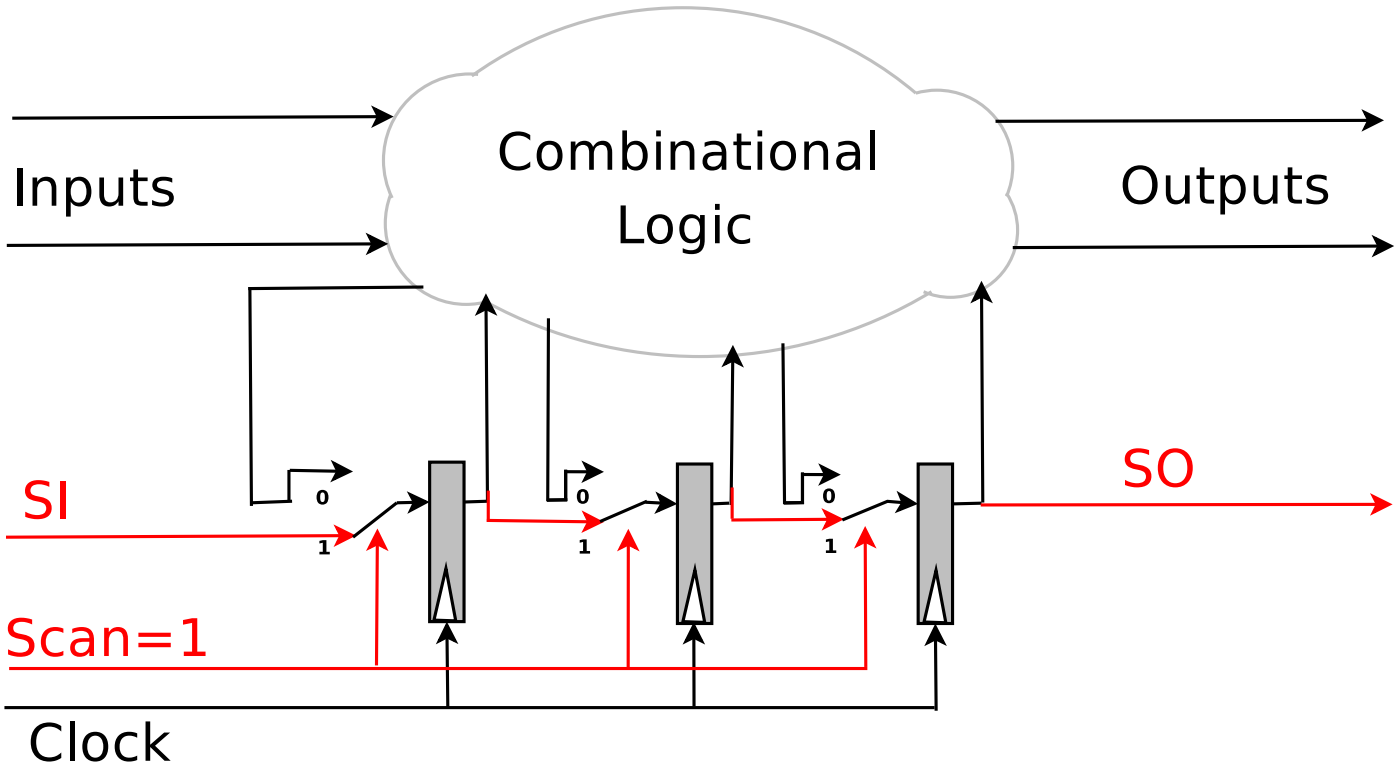


Circuit with Scan Insertion



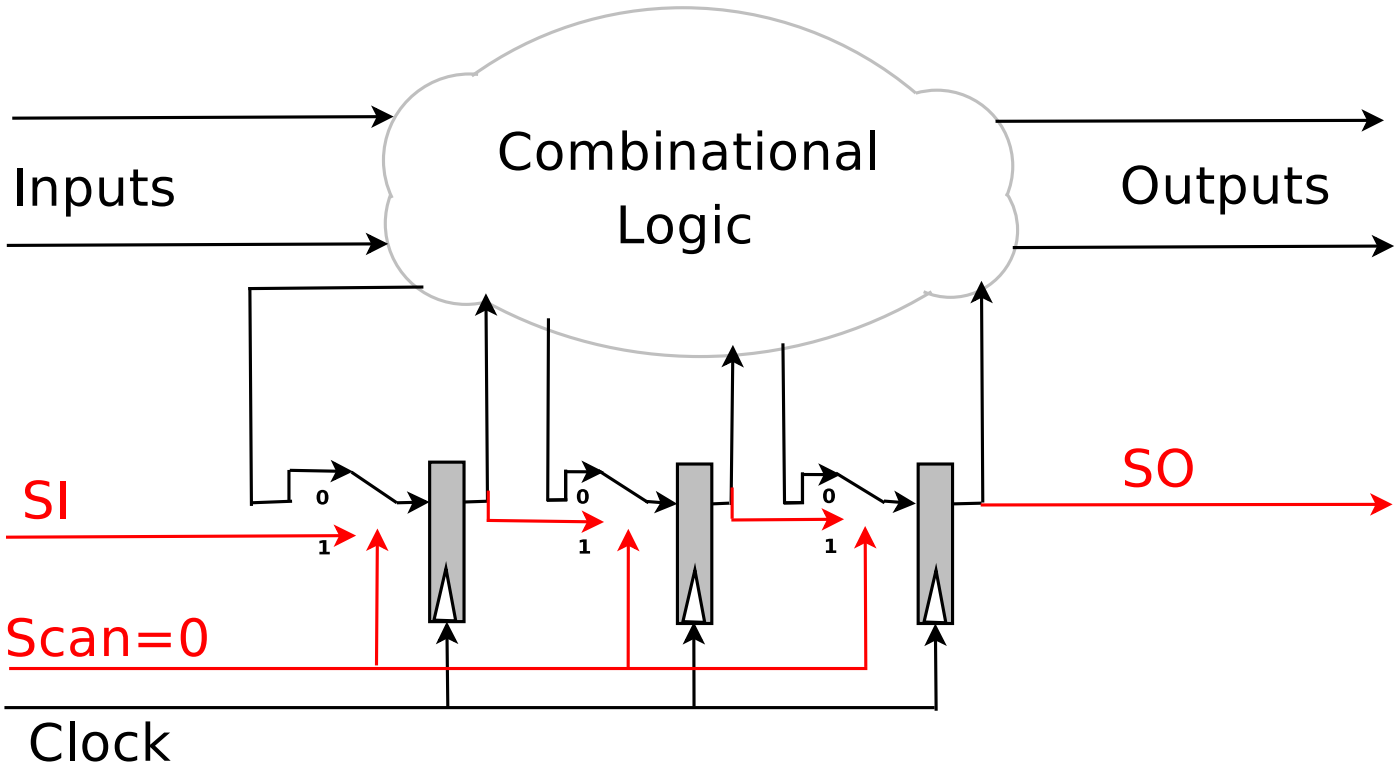
Circuit with Scan Insertion

Figure: Shift mode

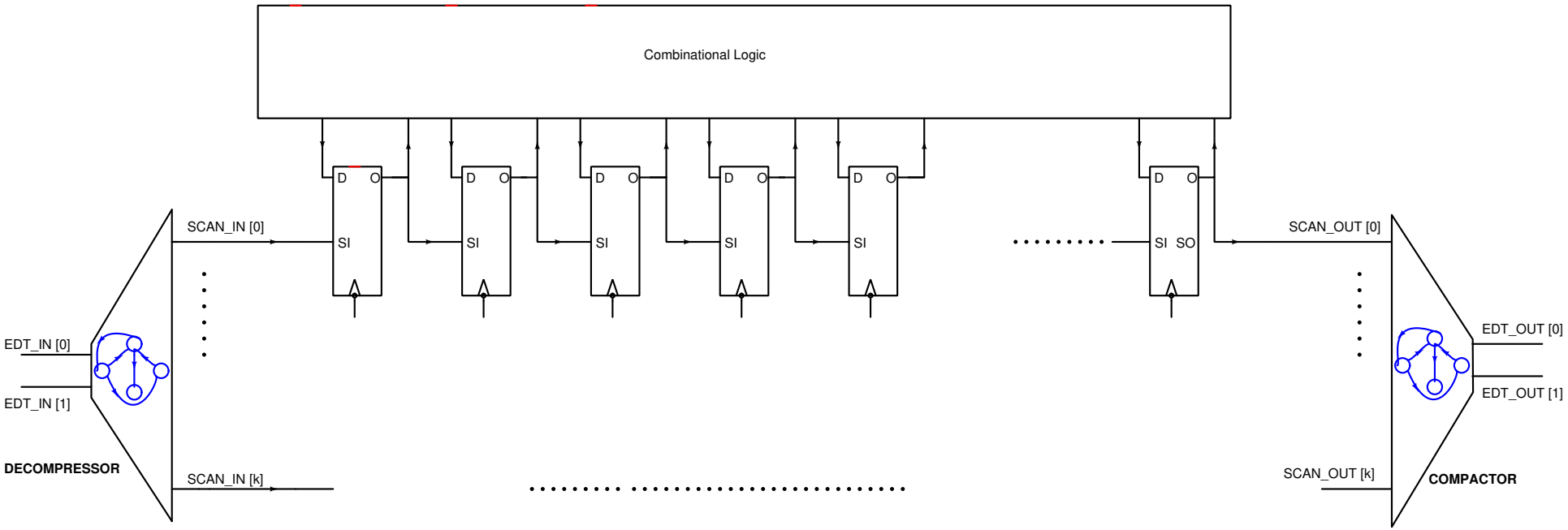


Circuit with Scan Insertion

Figure: Capture mode



Circuit with EDT Insertion



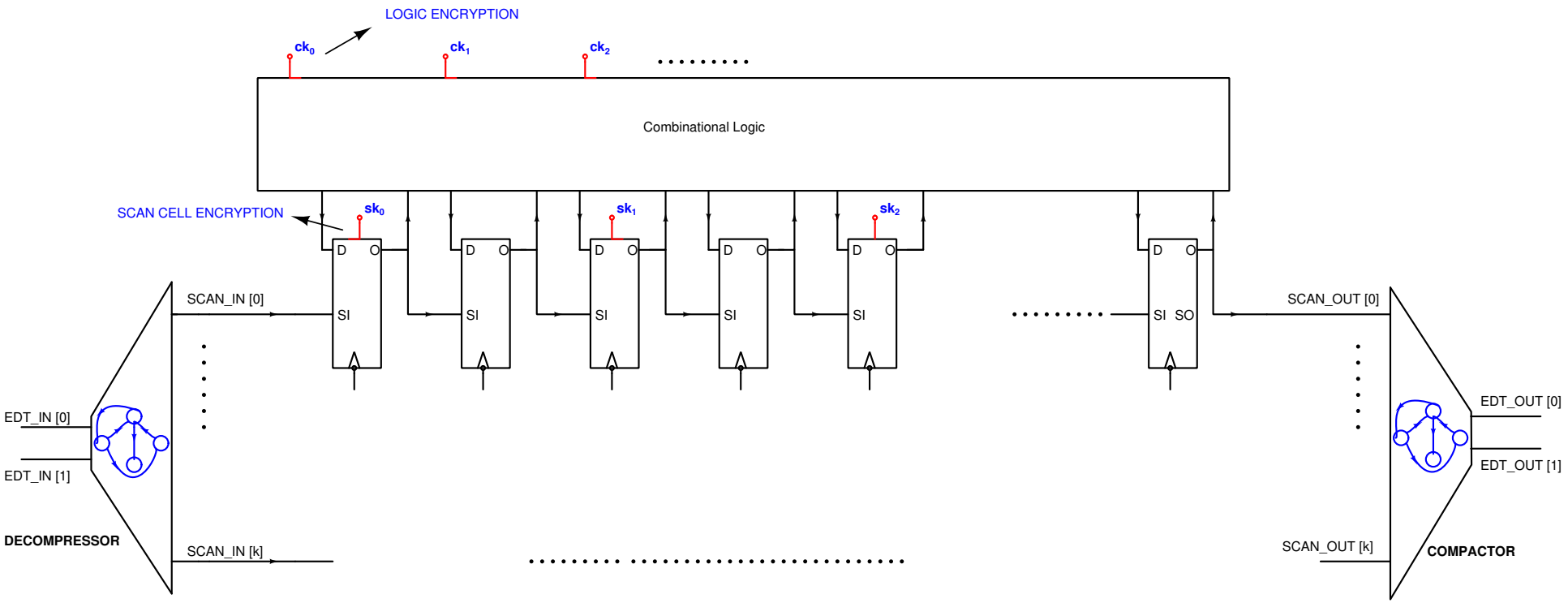
Modes of EDT Operation, SAT-attack

- EDT-mode : Not possible to launch the SAT-attack in this mode, since not possible to assign every scan cell to desired value
- EDT-BYPASS-mode : Possible to launch the SAT-attack in this mode, since all scan cells are daisy-chained and directly accessible

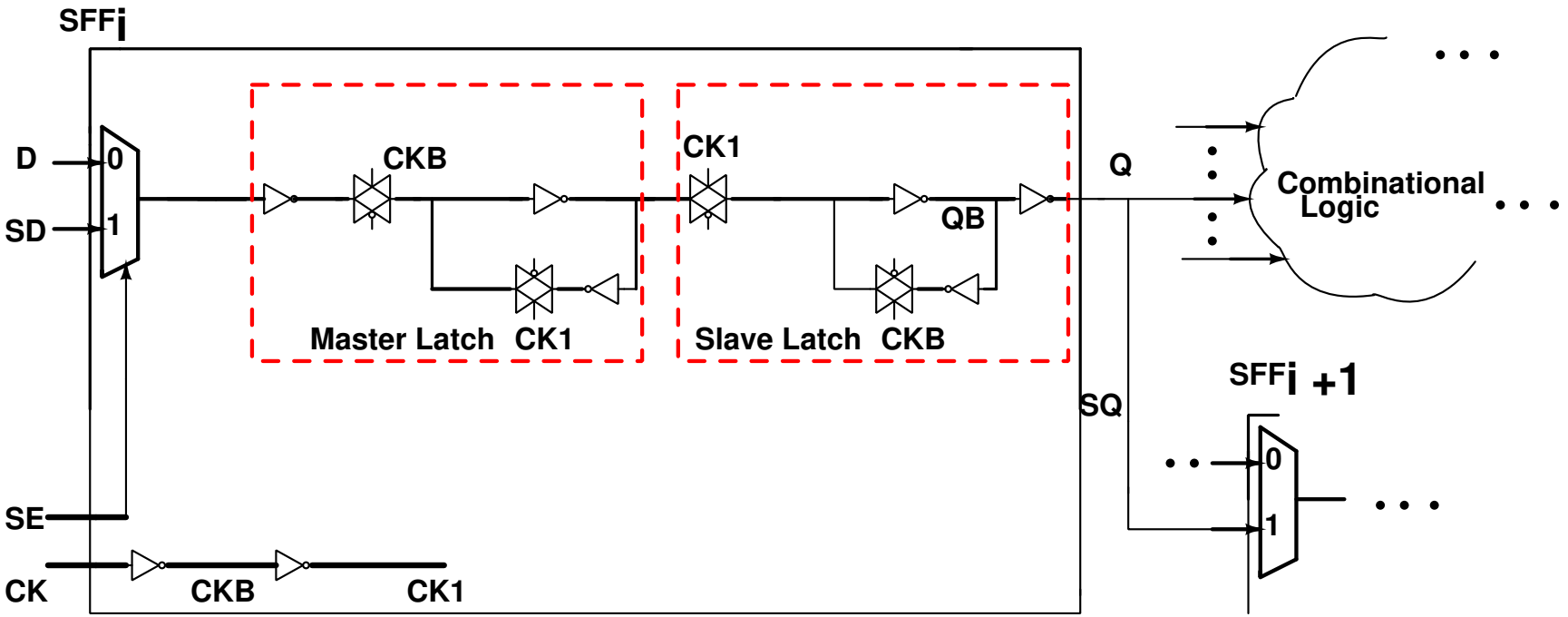
DFT Encryption

Idea: Encrypt DFT path corresponding to the EDT-BYPASS mode, to make sequential circuit resilient to SAT-attack

DFT Encryption



Libary Scan Cell



Existing Encrypted Scan Cell

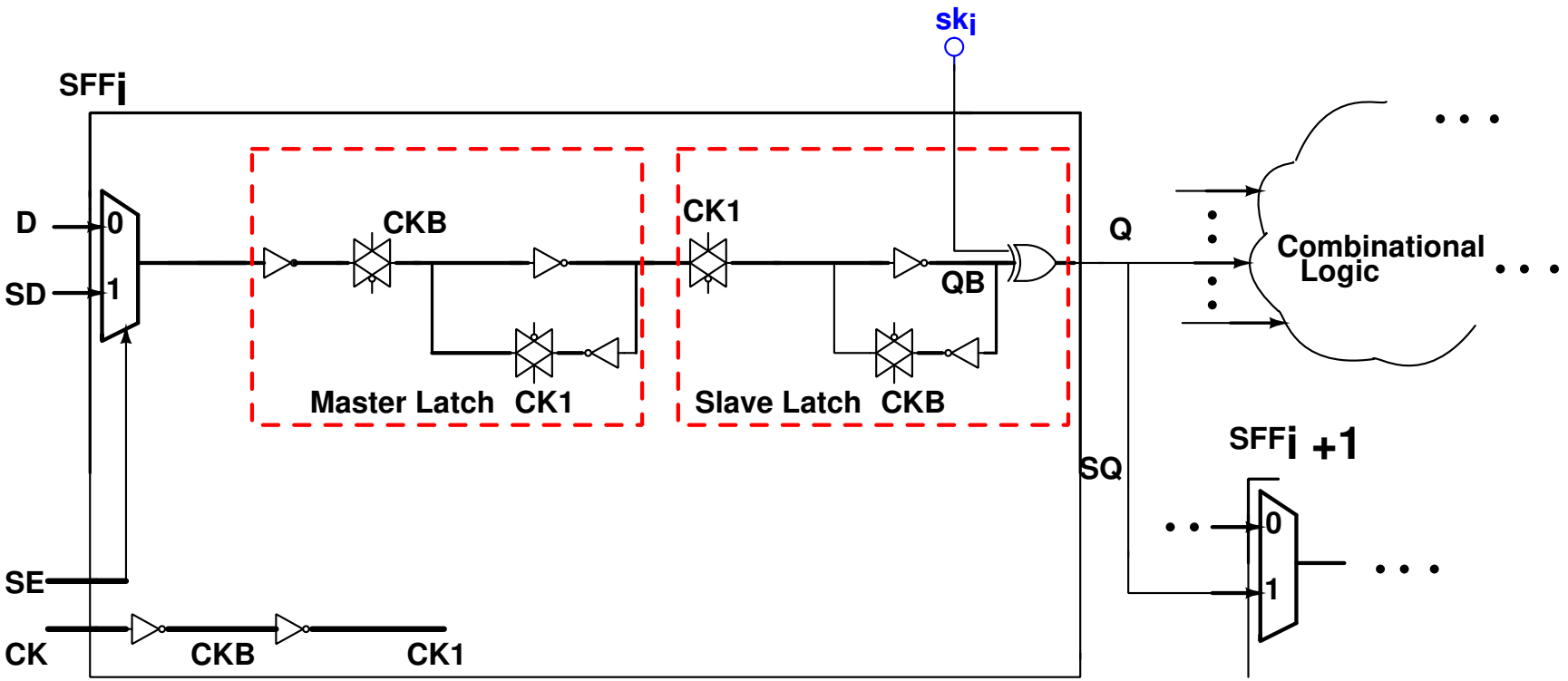


Figure: It is possible to decrypt the correct key[6]

[6] L. Alrahis, "ScanSAT: Unlocking Obfuscated Scan Chains", ASP-DAC 2019

Proposed Encrypted Scan Cell

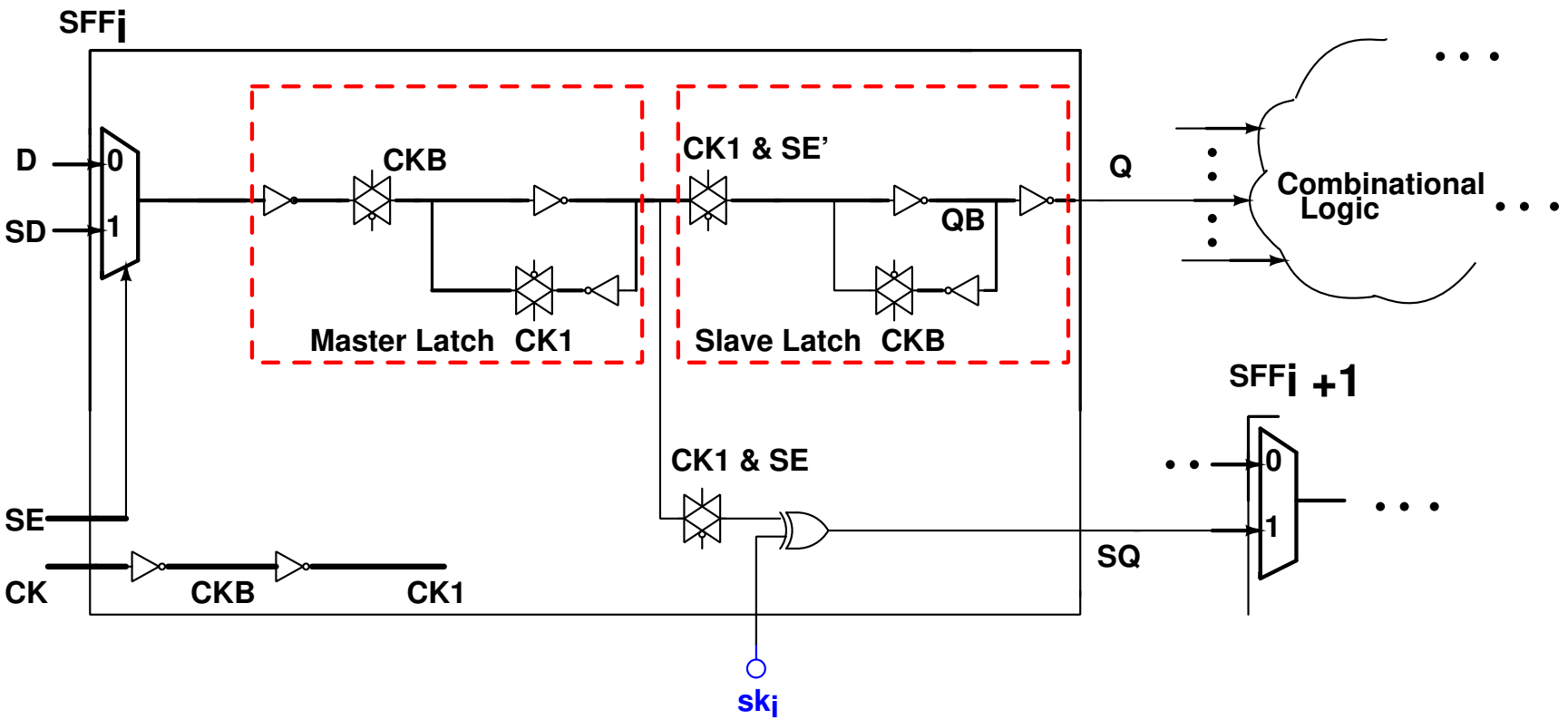


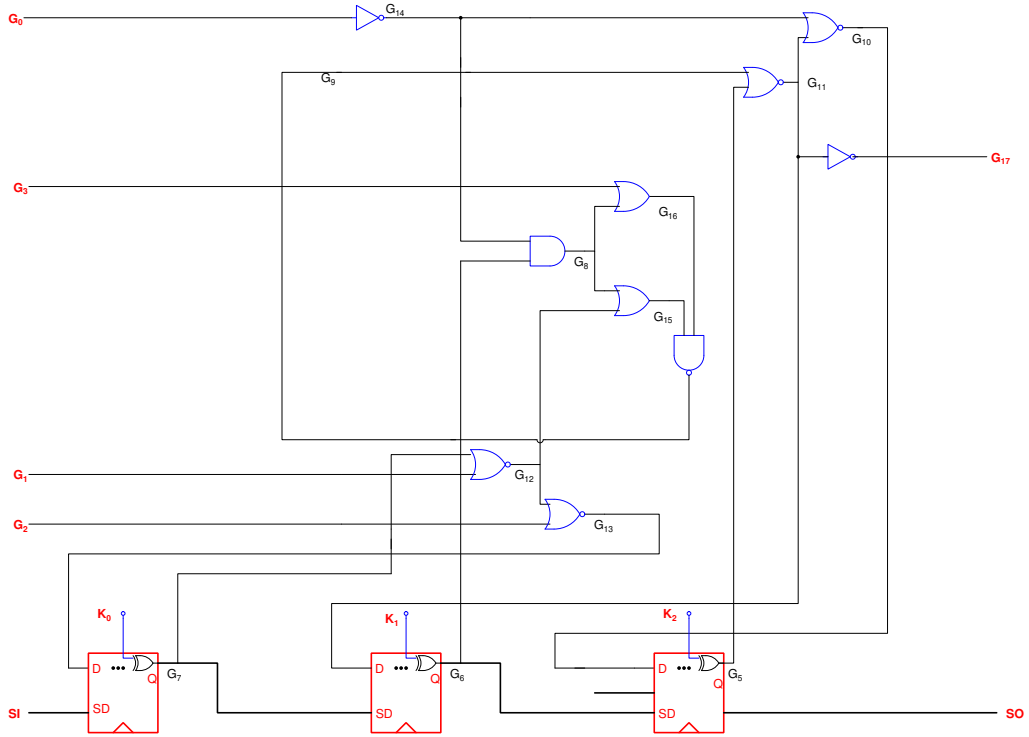
Figure: Only combinational portion of the sequential key locks the functional mode of operation

SAT attack on Encrypted DFT (BYPASS mode)

- All the scan chains connect together to form 1 single chain
- The scan cells can be removed and their inputs/outputs can be unrolled as a function of the XOR gates along the scan-shift path

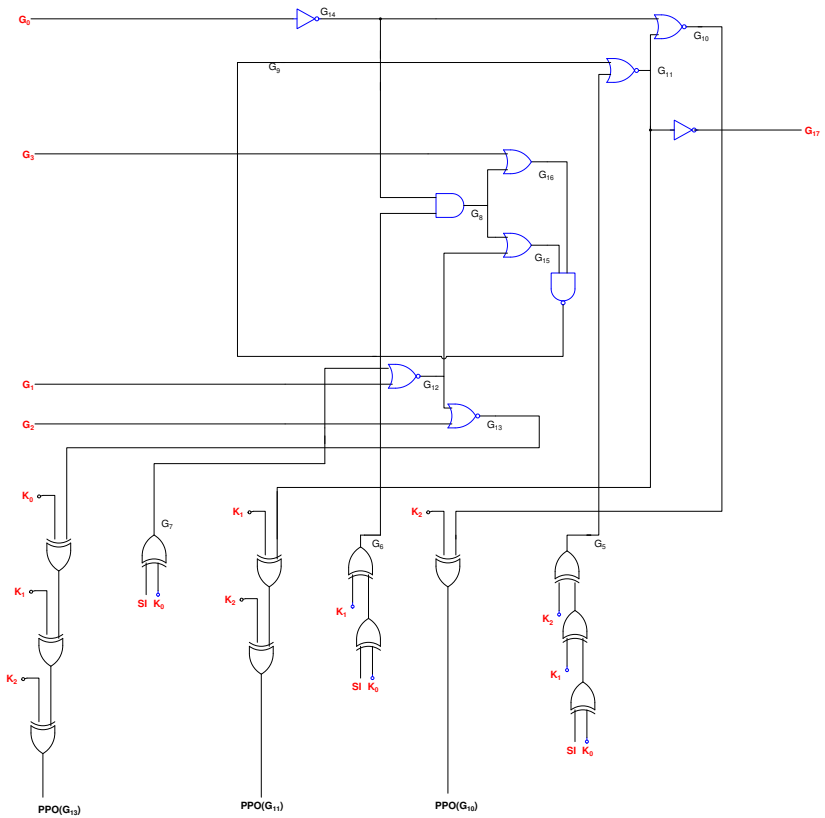
Example

Figure: s27 circuit with all scan cells encrypted



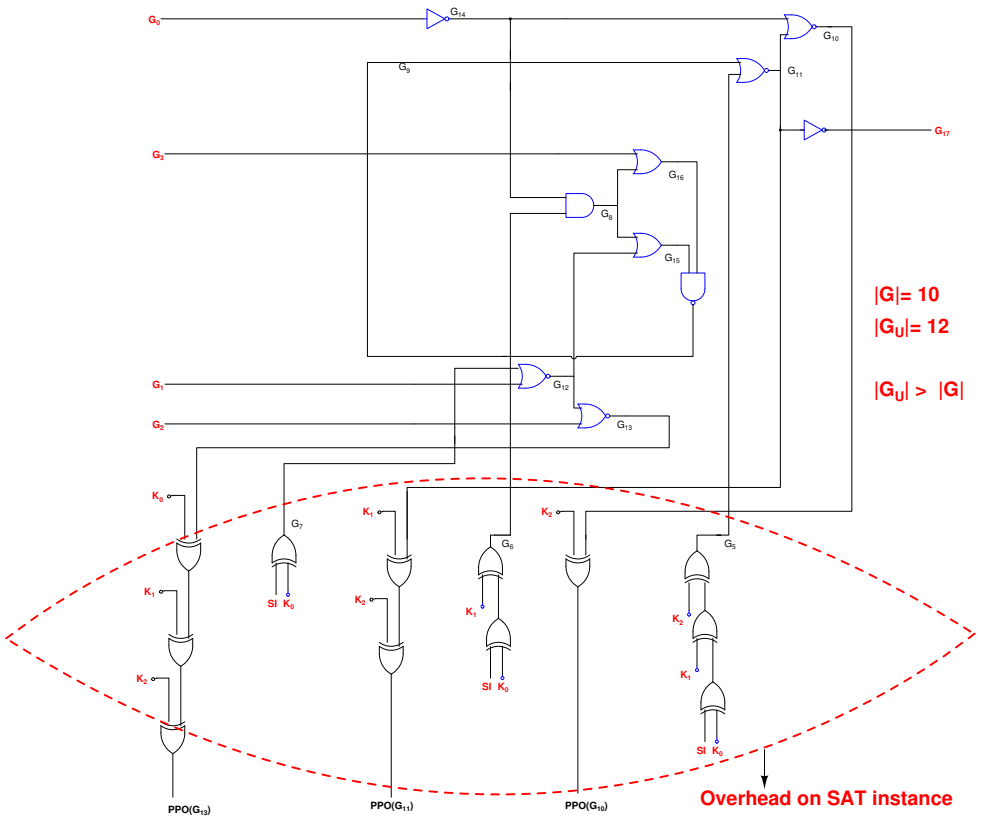
Example

Figure: s27 circuit after scan unroll



Example

Figure: s27 circuit after scan unroll - SAT overhead



Tradeoff

- The higher the number of scan-cells that are encrypted, the better
- The larger the size of the SAT instance, the harder to decrypt
- Encrypting the scan cell incurs slight area, timing and power overheads
- Routing scan-encryption-key bits incur slight area overhead

Notation

Notation

Define scan-unencrypted circuit $C(G, SC, CK)$ and scan-encrypted circuit $C'(G, SC, CK, SK)$, where
$G = \{g_0, g_1, \ldots g_N\}$ is set of gates
$SC = \{sc_0, sc_1, \ldots sc_N\}$ is set of scan cells
$CK = \{ck_0, ck_1, \ldots ck_X\}$ is the combinational-logic-encryption key and
$SK = \{sk_0, sk_1, \ldots sk_Y\}$ is the scan-encryption key

Notation

Define $C'_U(G, CK, G_U)$ be the scan-unrolled circuit, where G_U is the list of gates added to C during unroll.

SAT attack on Encrypted DFT (BYPASS mode)

Table: SAT instance complexity if all SCs encrypted

Benchmark	$\|G\|$	$\|SC\|$	$\|G_U\|$	Increase in SAT instance complexity
b17	27.99K	1407	2033K	$\approx$ 72x
b18	75.8K	3308	11451K	$\approx$ 152x
b19	146.5K	6618	44084K	$\approx$ 300x

Observations

- Let $n = |SC|$
- Encrypting i^{th} scan-cell in the scan-chain, contributes i XOR gates along scan-out path and $n - i + 1$ XOR gates along scan-in path, to G_U
- $(i) + (n - i + 1) = n + 1$. So, encrypting any scan-cell in the scan-chain (independent of the position), contributes $n + 1$ XOR gates to G_U.
- Encrypting all scan-cells contributes to $n(n + 1)$ XOR gates to G_U.

Observations

- Resilience against SAT-attack improves with size of scan-chain
- For e.g:. if an SoC has *1 million* scan-cells, in EDT-BYPASS mode, all of them form a single-chain. So, just encrypting any one scan cell in the chain will add $\approx$*1 million* XOR gates to the SAT instance, which makes the proposed technique very effective, with minimum overhead.

Problem Formulation

Objective

Encrypt a subset of scan-cells(in C), $SC_E \subset SC \ni \frac{|G+G_U|}{|G|}$ is maximized, while encryption overhead is minimized.

Sequential Locking and SAT-solving

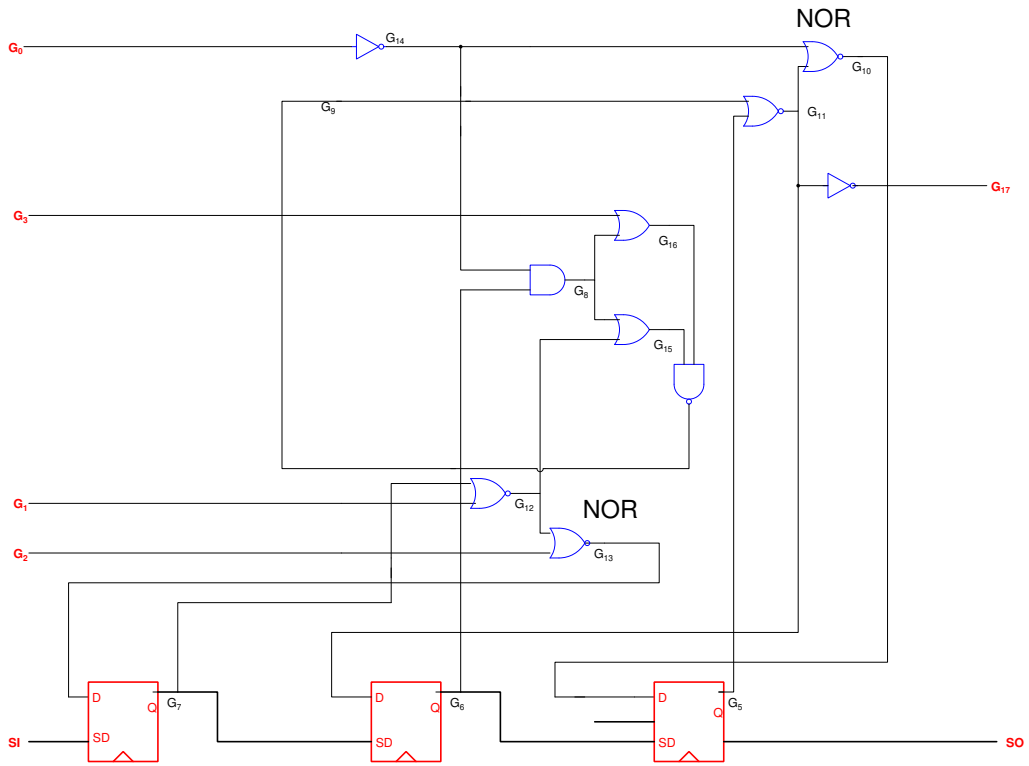
Table: SAT-attack on Scan-unrolled Sequentially Locked c880 circuit

# Scan cells (added at primary outputs) encrypted	Decryption time
0	3.49s
11	9.46s
16	45.86s
20	100.88s
24	250.48s

- Also, the correct key decryption was UNSUCCESSFUL
  1. Actual combinational key=00000011001111101111011011010101100100010000011110
  2. Decrypted combinational key=00000011001111101101011011010100000100100000011000

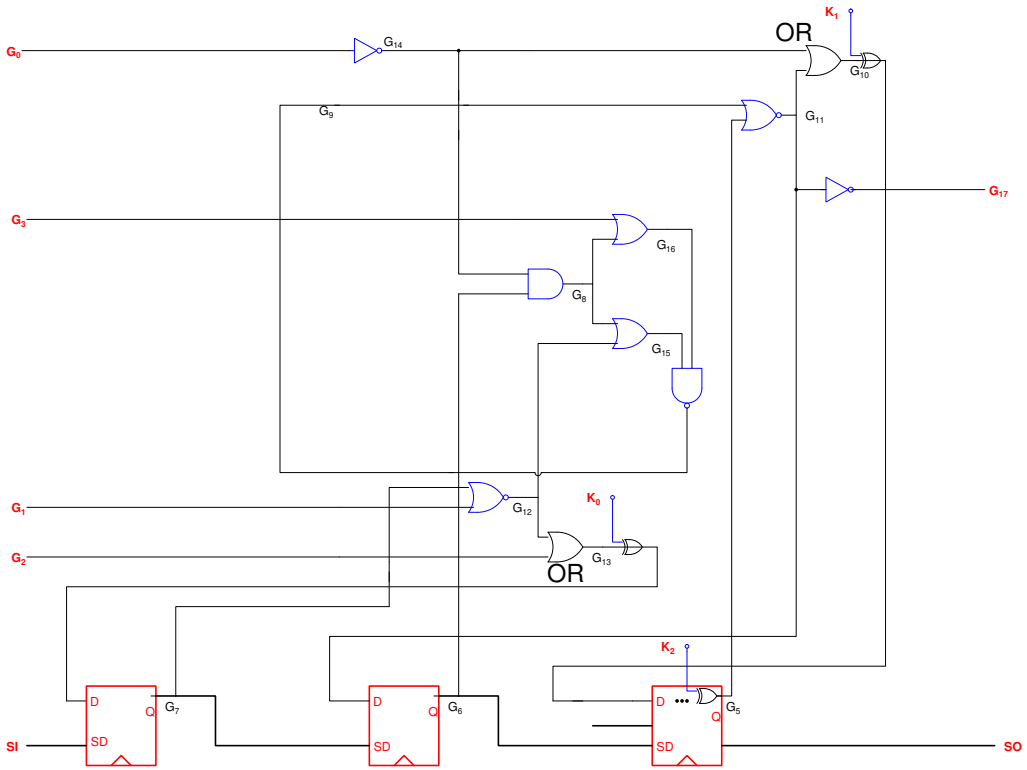
Explanation using small example

Figure: Original s27 circuit



Explanation using small example

Figure: Sequentially Locked s27 circuit



Explanation using small example: Equivalence classes of Keys

- $\{K_0, K_1, K_2\} = 110$ is a valid key
- $\{K_0, K_1, K_2\} = 001$ is also a valid key
- Thus, scan-chain locking helped produce equivalence classes of keys, that cannot be distinguished
- So, the attacker is unable to decrypt the correct key using SAT-attack
- Hence, sequential locking is able to achieve SAT-attack Resilience in two ways:
  1. Decryption of wrong keys during SAT-attack
  2. Increase in SAT-attack runtime by 1-2 orders of magnitude

Combinational key gate removal

Table: Combinationak key gate removal : SAT-attack on Scan-unrolled Sequentially Locked c880 circuit

# combinational key gates removed	Decryption time
0	s
1	s
2	s
3	s
4	s

Akash idea for future work

- Since flip-flops contain lot of useful information than just the state (for e.g:., information about future instructions, branching etc.), is it possible to lock sequential circuits in a different way ? For e.g:., encrypting Processor FSM in some new meaningful way ?
- The encryption should not just be similar to adding few gates on flip-flops, rather some complex encryption scheme that considers the instructions, data flow, FSM etc.

Akash feedback

- How does the decryption runtime change if systematically one-by-one the key gates are removed from the combinational logic ? If runtime does not reduce drastically, then there is huge area savings (because 50% of the circuit is encrypted)
- Agreed that the SAT solver returned different key for sequential locked case for c880
  - Will the different key also activate the IC, for correct functional operation ?
  - Is there a case (any of the benchmarks), where the SAT solver is returning the correct key ?
  - Can we come up with a heuristic, which ensures the solver comes up with a key that will produce wrong functional operation ?
- Use GIT to write the paper (both .docx and latex).

Proposed Experiments

- Since SAT-attack tool deals with bench format, try ITC'99 circuits because they are already available in bench format[7]
- Remove flip-flops and create pseudo-primary-inputs/outputs, and create original-type bench files
- Add combinational-logic-locking gates and create combinational-encrypted-type bench files
- On top of this, add sequential-locking-unrolling gates and create sequential-combinational-encrypted-type bench files
- Before implementing proposed technique on ITC'99 circuits, first check if SAT-attack is successful on large ITC'99 circuits - checking if SAT-attack is scalable

[7] http://www.pld.ttu.ee/ maksim/benchmarks/

Some results

- `b18, b19`, encryption with only 1 key-bit
- The run took 91m 3s, and 256m 57s, respectively to decrypt the 1-key bit.
- I think this is the reason, researchers working on logic locking, partition the circuit and launch the SAT-attack on the individual partitions.
- But I don't see the point in partitioning and then do SAT-solving on individual partitions, because the SAT solver should do the same thing anyway.

Some more important observations

- Combinational logic locking is vulnerable to removal attack
- Sequential logic locking is NOT vulnerable to removal attack
- Limitations on *Encrypt Flip-Flop* paper from IIT-KGP.
  - Adding XOR gates into netlist increases physical design effort
  - Partitioning the circuit, and attacking individual partitions is not possible, because the outputs of some partitions are *internal nodes*, whose logic states are not known (only the states of primary inputs and primary outputs are known in SAT-based attack)

Some more important observations

- `b03`: if encrypting only scan-chain, and no logic-locking, key is getting decrypted in 1.45s
- San-unrolled circuit for `b03`, is as big as scan-unrolled circuit for `c880` (with scan-cells at primary outputs), however in latter case, it took more than 100s
- Conclusion: if only locking flip-flops, it gets decrypted very fast, because it is only XOR chain, which is easy to decode
- However, next and more important question is about correctness of the decrypted key ?

Verification with c880

Figure: Combinational locking; Solver Key $\{k_{18}, k_{97}, k_{111}, k_{133}\} = \{1, 1, 0, 1\}$

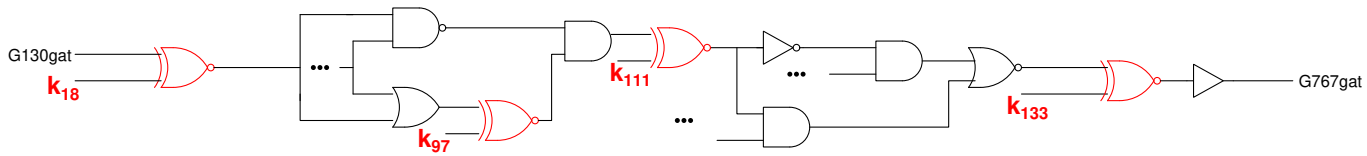
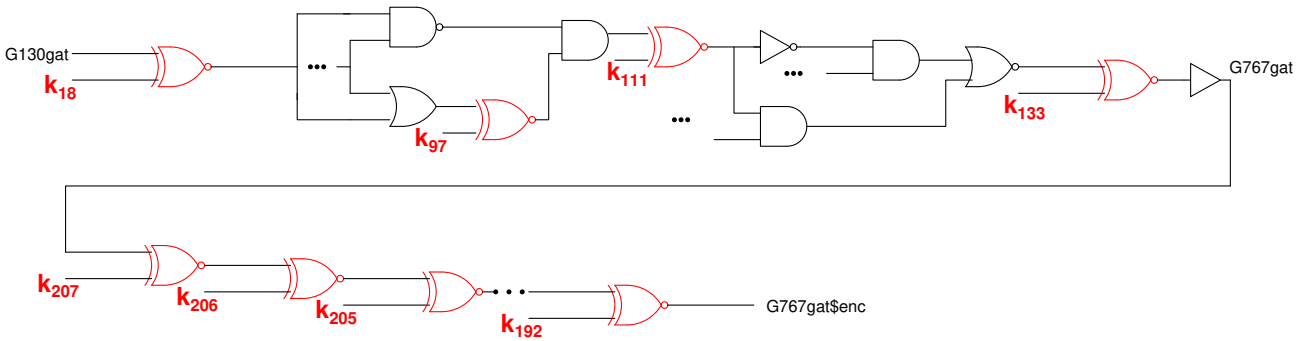


Figure: Sequential locking; Solver Key
$\{k_{18}, k_{97}, k_{111}, k_{133}, k_{207}, k_{206}, k_{205}, \ldots k_{192}\} =$
$\{0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0\}$



Verification with c880

- c880 with combinational locking: key is 1101, which is odd-inversion-parity
- c880 with sequential locking: combinational portion of the key is 0110, which is even-inversion-parity
- Thus, for chip that uses sequential locking, during functional mode of operation, the signal undergoes wrong number of inversions before it reaches the destination flip-flop, thus corrupting the circuit state.
- When the SAT-attack decrypted key is applied: during scan, the circuit functions correctly, but during functional mode of operation, it is corrupt.

Akash feedback on 10.09.2018

- Agreed that the solver currently returns the wrong key, but is it possible to find the correct key ?
- Are there multiple keys that the solver is able to return (equivalence class) ?
- Among them, is it possible to have a correct key ?
- Complexity analysis for idenfifying all keys within the equivalence class; even a theoretical estimate is fine.

Next steps

- Read Subramanyan's paper on the concept of equivalence classes for combinational locking
- Try to extend the same concept to sequential locking

Definitions in SAT-attack paper

- Two keys $\vec{K_1}$ and $\vec{K_2}$, are equivalent, denoted as $\vec{K_1} \equiv \vec{K_2}$, iff for each input value $\vec{X_i}$, the encrypted circuit produces the same output $\vec{Y_i}$, for both keys $\vec{K_1}$ and $\vec{K_2}$. Precisely stated: $K_1 \equiv K_2$ iff $\forall X_i$: $C(\vec{X_i}, \vec{K_1}, \vec{Y_i}) \wedge C(\vec{X_i}, \vec{K_2}, \vec{Y_i})$
- Given two key values, K_1 and K_2, define the input pattern $\vec{X^d}$ as a distinguishing input pattern if the encrypted circuit outputs different values $\vec{Y_1^d}$ and $\vec{Y_2^d}$ when the key inputs are set to $\vec{K_1}$ and $\vec{K_2}$ respectively. More precisely, $\vec{X^d}$ is a distinguishing input pattern for K_1 and K_2 iff $C(\vec{X^d}, \vec{K_1}, \vec{Y_1^d}) \wedge C(\vec{X^d}, \vec{K_2}, \vec{Y_2^d})$.

Limitations of SAT-attack paper

- In section IV.A, they mention, "`c6288` is a multiplier, and multiplers are challenging for SAT solvers in all contexts, not just logic encryption".
- The paper suggests that Algorithm 1 ends, when it finds two keys, $\vec{K_1}$ and $\vec{K_2}$:
  - which are equivalent; and
  - which satisfy input/output observations for distinguishing input patterns so far, on an activated IC.

Limitations of SAT-attack paper

- The paper says loop in Algorithm 1 ends, when no distinguishing inputs can be found. What does this mean ? Does the algorithm look for distinguishing inputs among all possible input combinations (2^M in a circuit with M inputs) ?
- If that is the case, it is exponential anyway.
- Even if this is true, I am still unsure if this is sufficient justification to conclude that $\vec{K_1}$ and $\vec{K_2}$ belong to equivalence class of correct keys. It could very well belong to the equivalence class of wrong keys, but satisfying the input/output observations so far on an activated IC.
- I also did not understand how each iteration of the while loop rules out at least one incorrect equivalence class. All we can say is that we will never again choose two keys that belong to the equivalence class pairs visited before.

Possible next step for combinational locking

- What kind of positioning of key gates within the logic circuit, produces equivalence classes ?
- If that is known, place the key gates differently, so that sizes of equivalence classes is minimized, and thereby the number of equivalence classes is maximized.
- This will increase the time it takes to decrypt using SAT-attack, thus making it hard.
- In the trivial case, check if it is possible to place the key gates in such a way that the size of all equivalence classes is exactly 1. This is similar to the AND-tree inside `c2670` mentioned in the SAT-attack paper, where each key assignment is a *singleton equivalence class.*. Here, the SAT-attack tries to eliminate a wrong key in each iteration of while loop, thus ends doing brute-force search of keys. Clearly this is intractable.

Akash feedback on 12.09.2018

- Check if there is a distinguishing input between the key returned by SAT-attack and the combinational part of the key returned by our method ?
- Orthogonal method : After applying corresponding keys, check if both circuits are formally equivalent, to original circuit.
  - ./lcmp ../benchmarks/original/c880.bench ../benchmarks/rnd/c880_enc50.bench key=00000011001111101110011011010000100100010000000100000
    The output will be the string 'equivalent' if the correct key is provided. If not, the utility will output the string 'different'. The above command should produce the output 'equivalent'. If any of the key bits are changed, it is very likely that the utility will output 'different'.

Next steps

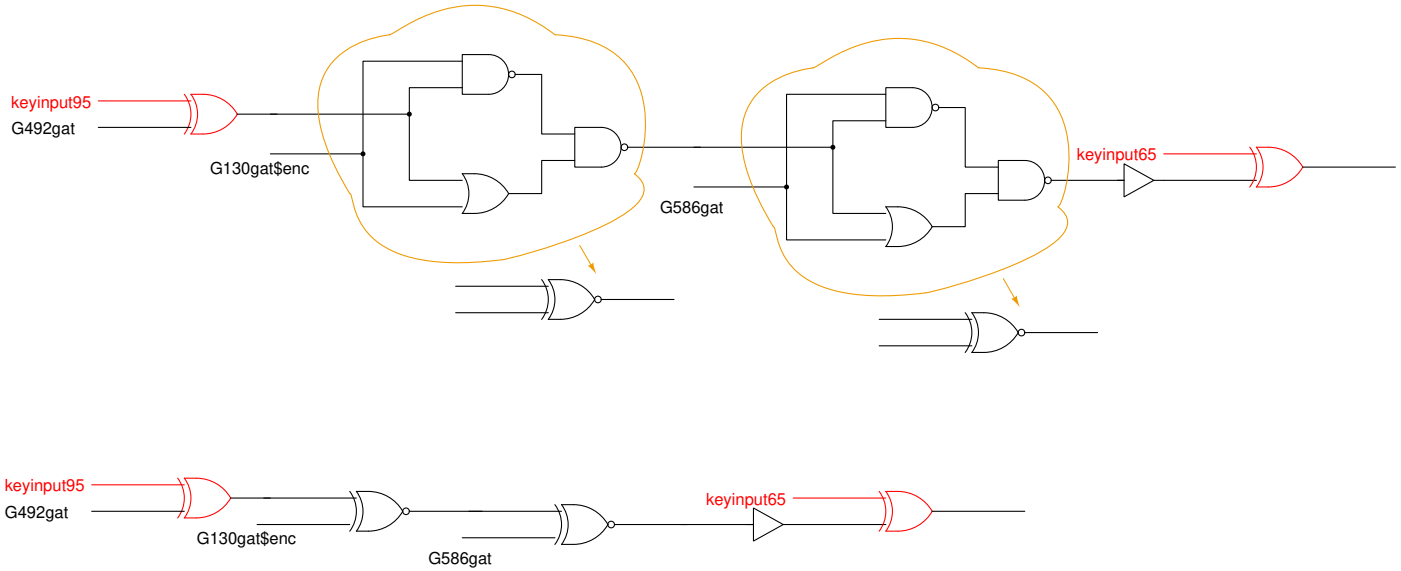
- Need separate combinational netlists for scan-mode (effect of XOR gates along the scan-chain) and functional-mode (effect of XOR gates of capture flops).
- In fact, some gates in scan-mode netlist can be removed to arrive at the functional-mode netlist.

ISCAS85 circuits summary

- `c432` : 27-channel interrupt controller (no XOR chains at primary outputs)
- `c880` : 8-bit ALU
- `c6288`: 16-bit multiplier
- `c499/c1355` : 32-bit SEC circuit
- `c1908`: 16-bit SEC/DED circuit
- `c7552`: 32-bit adder/comparator
- `c5315`: 9-bit ALU

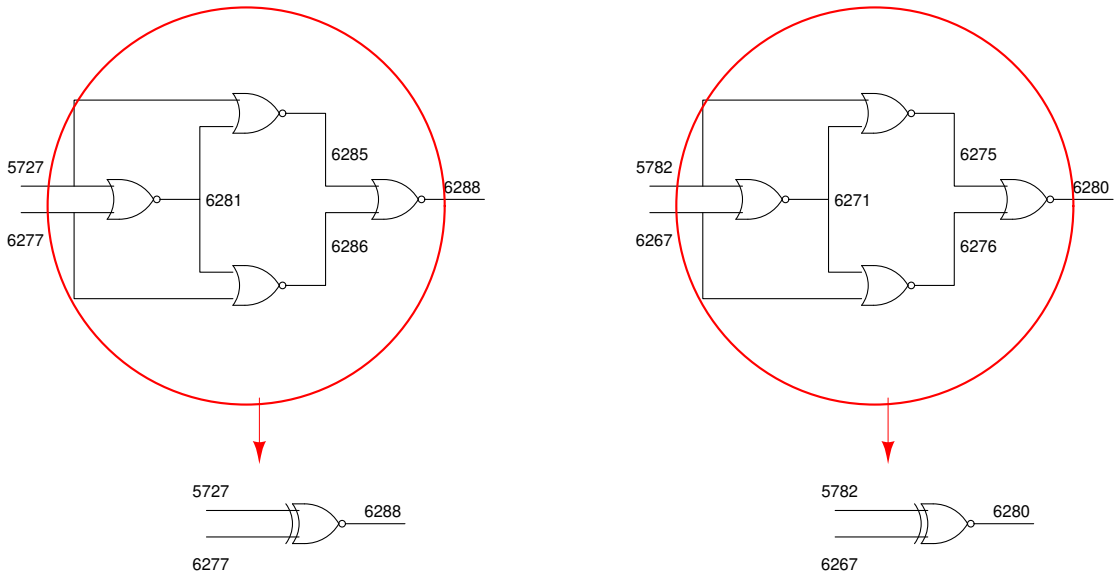
c880 XOR chains

Figure: c880 XOR chains at primary outputs



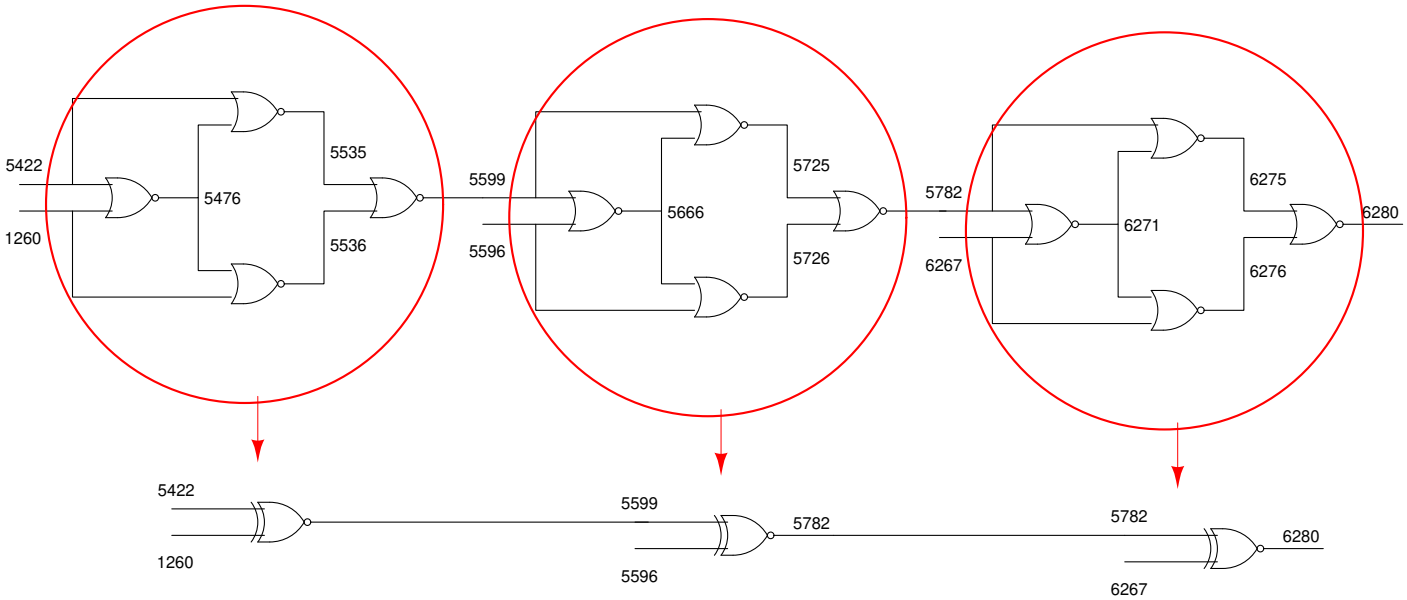
c6288 XOR chains

Figure: c6288 XOR chains at primary outputs



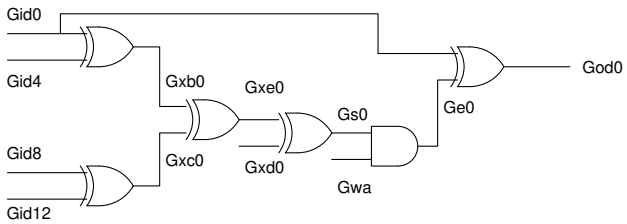
c6288 XOR chains

Figure: c6288 XOR chains at primary outputs



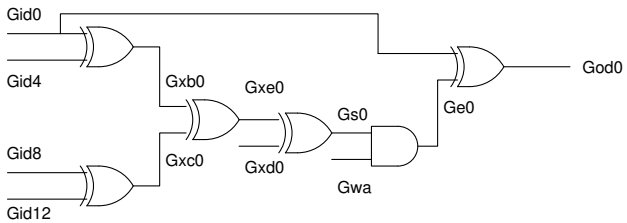
c499 XOR chains

Figure: c499 XOR chains at primary outputs



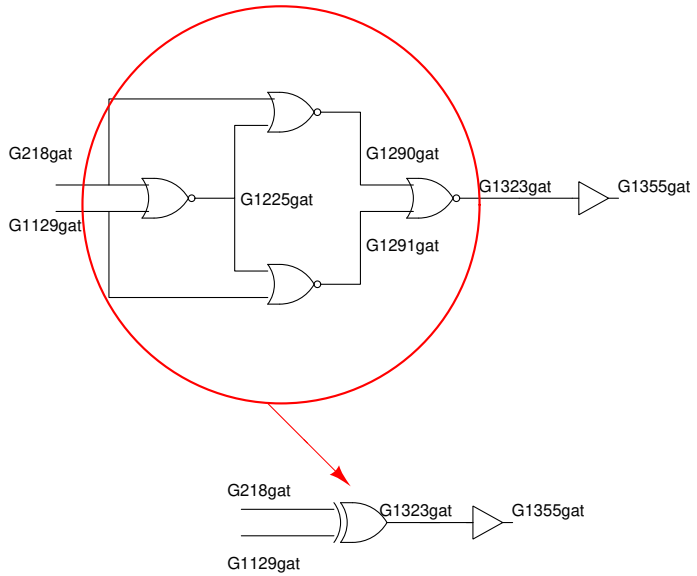
c499 XOR chains

Figure: c499 XOR chains at primary outputs



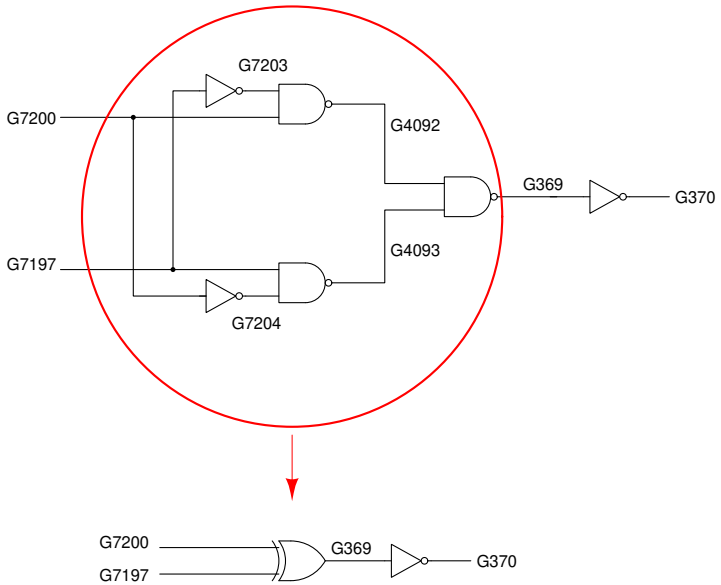
c1355 XOR chains

Figure: c1355 XOR chains at primary outputs



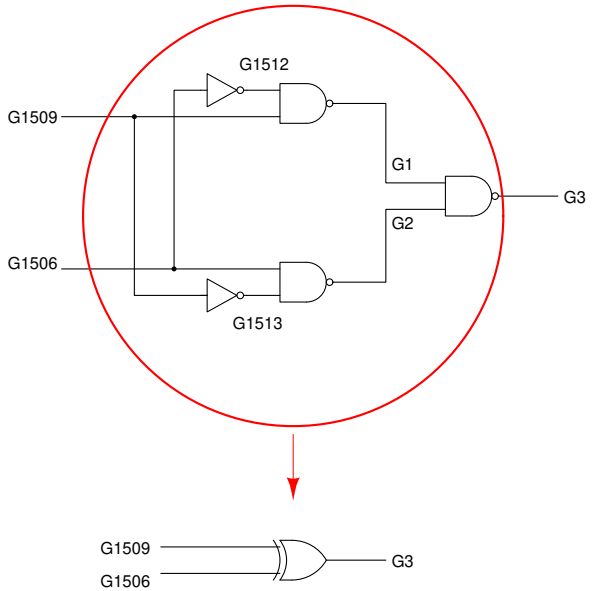
c7552 XOR chains

Figure: c7552 XOR chains at primary outputs



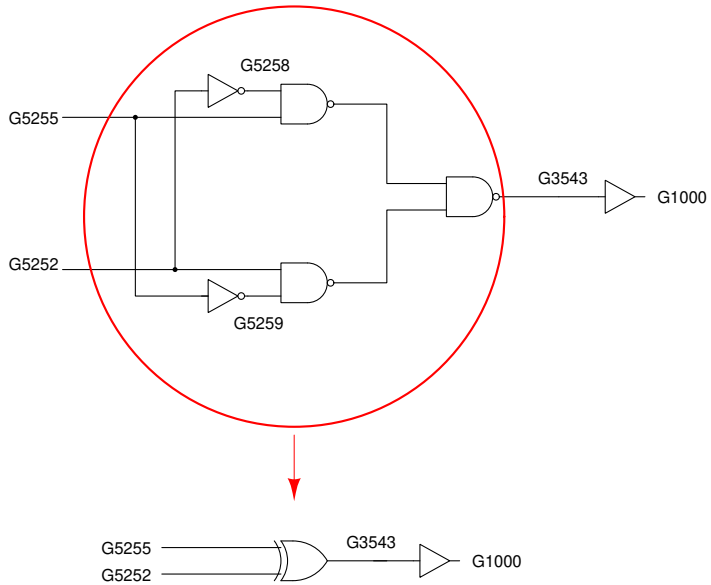
c1908 XOR chains

Figure: c1908 XOR chains at primary outputs



c5315 XOR chains

Figure: c5315 XOR chains at primary outputs



Akash feedback on 20.09.2018

- Algorithm to automatically identifying combinational XOR-chains at primary outputs, and encrypting them
- Algorithm to encrypt the scan-chain
- Quantify runtime and functional output corruption
- Implementation on ISCAS'85 and OpenSPARC circuits
- Idea: Since there are primary output XOR-chains for ALUs (used for PC, Branch target calculator, Execution units etc.), this idea will lock the processor successfully
- Algorithm for selective flip-flop locking, to minimize area overhead due to sequential locking

Next steps

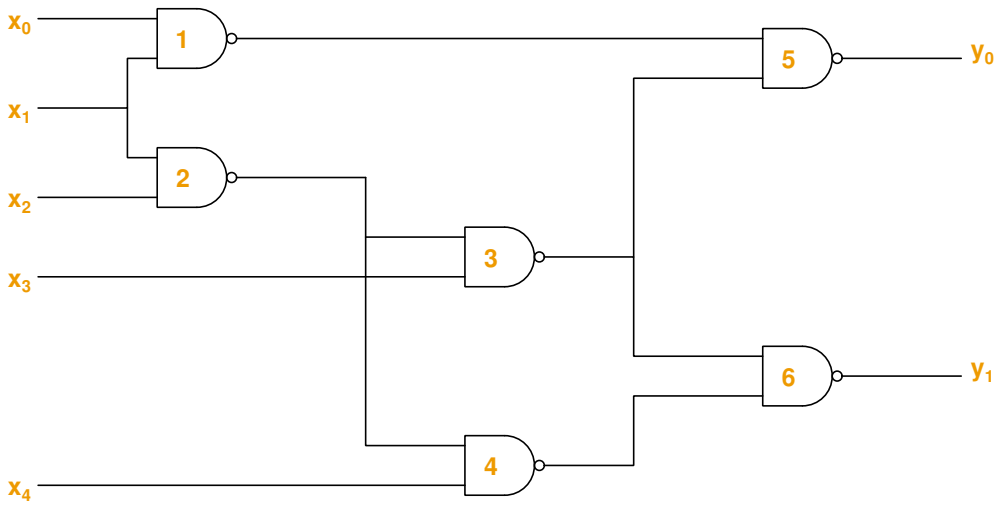
- Algorithm to decode correct combinational key, given sequential key

Algorithm to decrypt correct combinational key

- Step 1: Run SAT solver on scan-unrolled combinational circuit, to find sequential key
- Step 2: Isolate combinational circuit, apply combinational portion of key, and add virtual key gates at POs, and run SAT solver
- Step 3: Apply virtual key values, and make combinbational key unknown again, and run SAT-solver

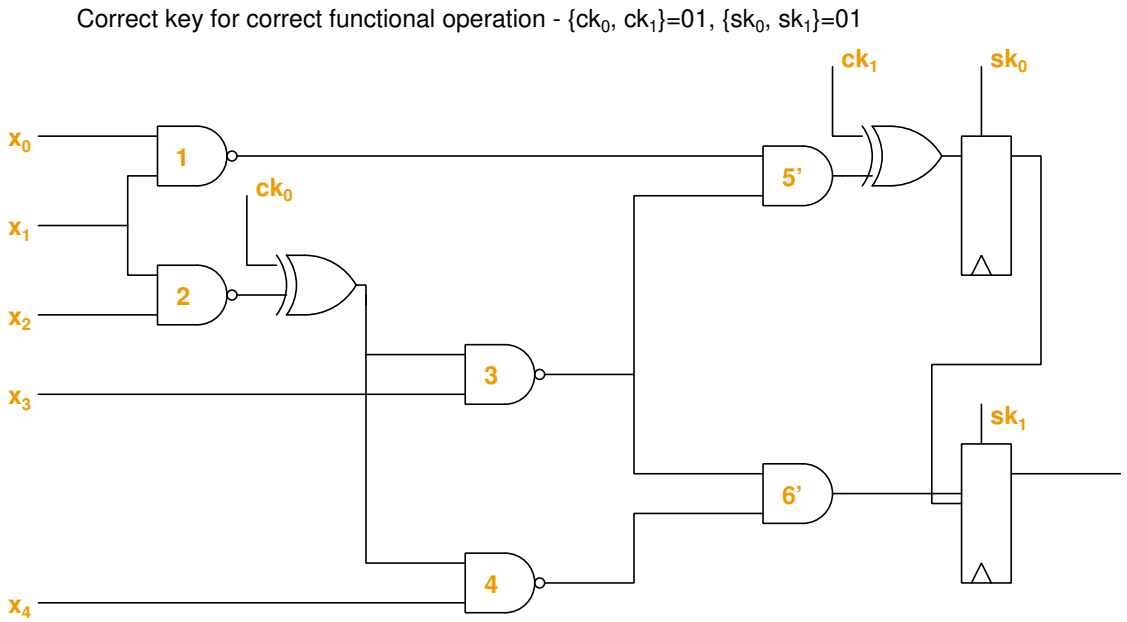
c17 circuit

Figure: c17 combinational circuit



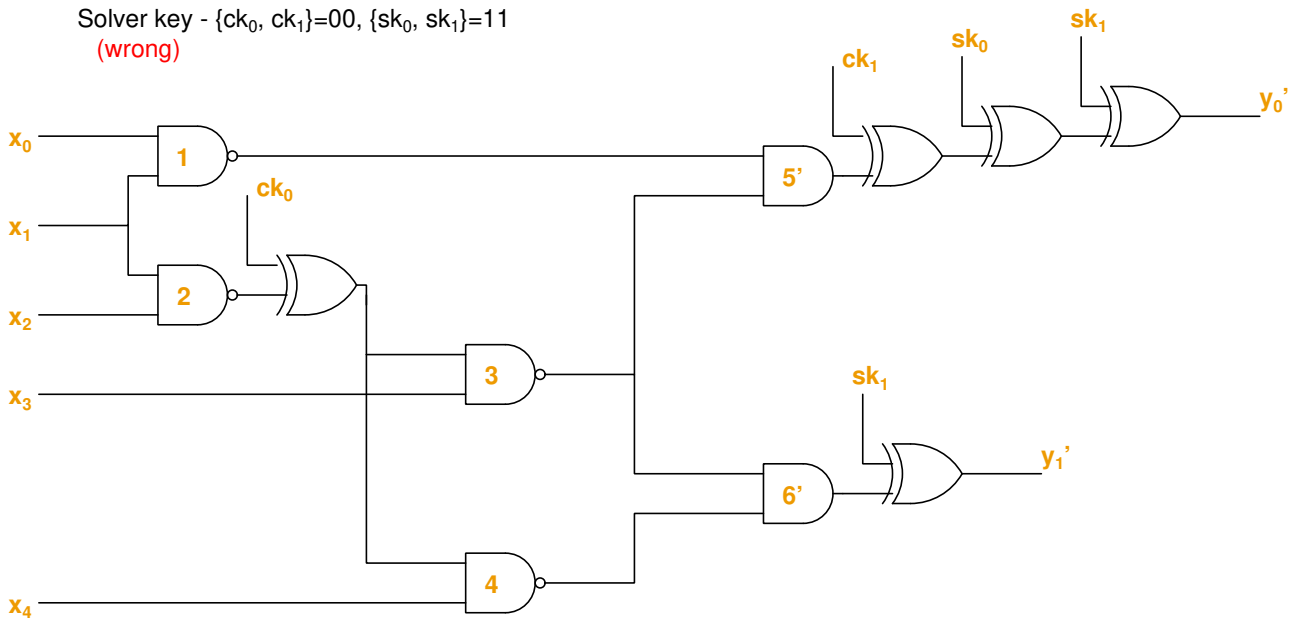
c17 circuit with sequential locking

Figure: c17 circuit with sequential locking



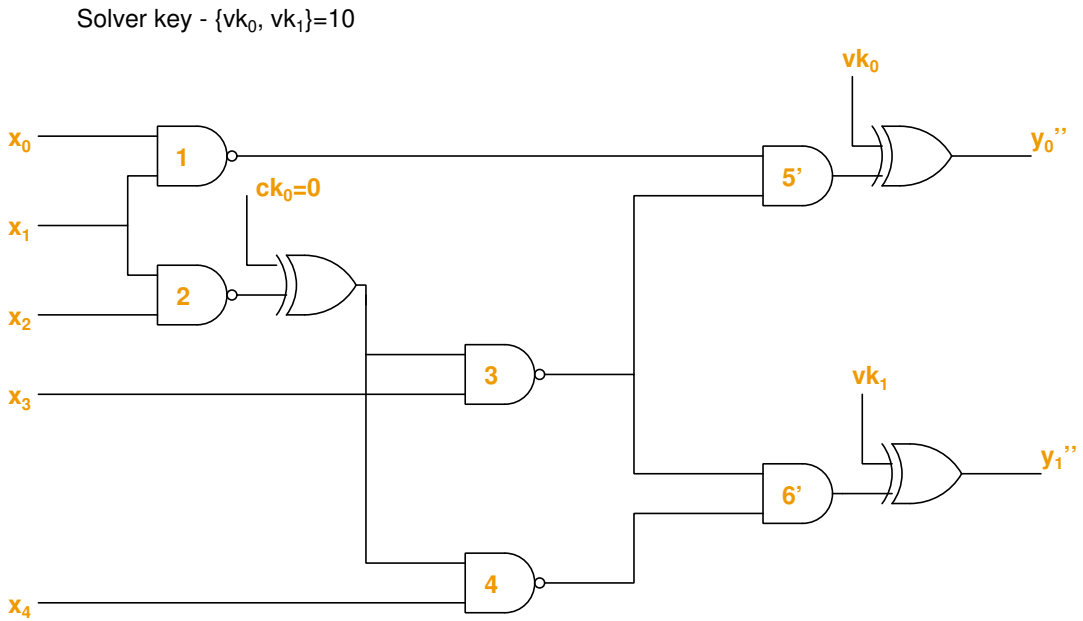
c17 circuit with scan unroll and Step 1

Figure: c17 circuit with scan unroll and Step 1



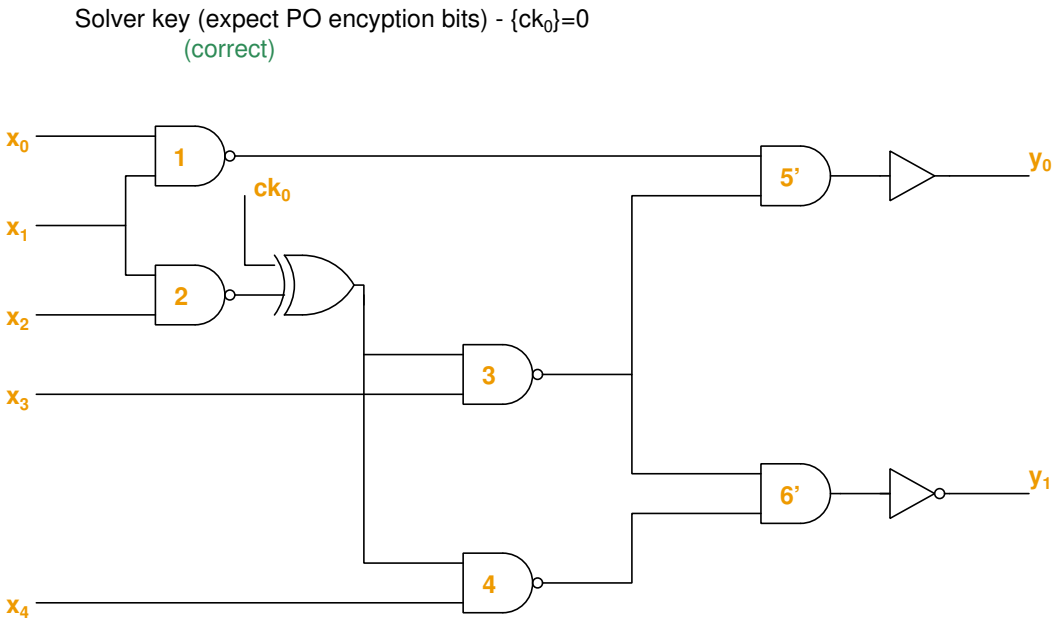
c17 circuit with scan unroll and Step 2

Figure: c17 circuit with scan unroll and Step 2



c17 circuit with scan unroll and Step 3

Figure: c17 circuit with scan unroll and Step 3



Observation

- In principle, the Algorithm can solve for the correct combinational key
- However, we do not have the reference correct combinational circuit for Step 2, since we only have access to scan data on the activated IC
- Hence Algorithm is not feasible
- Thus, we need to make 2^M (for circuit with M flip-Flops) observations.
- For `c880` circuit, $M = 26$, thus running time = $2^{26} * 0.04s = 3,355,433s \approx 1 month$
- `c880` has 105 primary inputs (PIs); if we add flip-flops at PIs as well, running time increases $2^{105}\times$ further.

Experimental evaluation

- Formulate the sequential locking problem, as an instance of the logic locking problem, and solve for the sequential key
- Extract the combinational portion of the key, and apply to the combinational portion of the circuit, and check if it is formally equivalent to original circuit
- If YES, it means decrypted decrypted key produces correct functional outputs when applied to the encrypted circuit
- If NO, it means decrypted decrypted key produces corrupt functional outputs when applied to the encrypted circuit

Results for 5% combinational (random) encryption and encrypting all output flip-flops

- Decrypted key produces correct functional outputs : `apex2`, `apex4`, `c432`, `c499`
- Decrypted key produces corrupt functional outputs : `c880`, `c7552`, `c1908`, `c1355`, `c5315`, `c2670`, `c3540`, `dalu`, `des`, `ex1010`, `ex5`, `i4`, `i7`, `i8`, `i9`, `k2`, `seq`
- Success rate : $\frac{17}{21}$ cases $\approx 81\%$

Results for 5% target combinational (random and PO) encryption and encrypting all output flip-flops

- Decrypted key produces correct functional outputs : Nil
- Decrypted key produces corrupt functional outputs : c880, c7552, c1908, c1355, c5315, c2670, c3540, dalu, des, ex1010, ex5, i4, i7, i8, i9, k2, seq, c432, c499, apex2, apex4
- Success rate : $\frac{21}{21}$ cases $= 100\%$

Results for 5% combinational (DAC'12 fault analysis based) encryption and encrypting all output flip-flops

- Decrypted key produces correct functional outputs : `i4`, `i7`, `i8`, `i9`, `apex2`, `apex4`, `seq`, `k2`, `dalu`, `ex1010`, `ex5`, `des`, `c432`, `c499`, `c7552`, `c1908`, `c1355`, `c3540`
- Decrypted key produces corrupt functional outputs : `c880`, `c5315`
- `c2670` run in progress ($>$ 48 hours). The SAT-attack paper mentions they could not decrypt this benchmark.
- Success rate : $\frac{2}{20}$ cases $= 10\%$

Results for 5% combinational (DAC'12 Fault analysis + PO) encryption and encrypting all output flip-flops

- Decrypted key produces correct functional outputs : `apex2` (only 3 POs),
- Decrypted key produces corrupt functional outputs : `i4`, `i7`, `i8`, `i9`, `apex4`, `k2`, `seq`, `dalu`, `des`, `ex1010`, `ex5`, `c432`, `c499`, `c880`, `c1355`, `c3540`, `c5315`, `c1908`, `c7552`
- Success rate : $\frac{19}{20}$ cases $= 95\%$

Results for 5% combinational (IOLTS'14) encryption and encrypting all output flip-flops

- Decrypted key produces correct functional outputs : `apex2`, `apex4`, `i4`, `i7`, `i8`, `i9`, `ex1010`, `ex5`, `k2`, `seq`, `dalu`, `des`, `c432`, `c499`, `c1908`, `c1355`, `c5315`, `c7552`, `c3540`, `c880`, `c2670`
- Decrypted key produces corrupt functional outputs : Nil
- Success rate = $\frac{0}{21}$ cases = 0%

Results for 5% combinational (IOLTS'14 and PO) encryption and encrypting all output flip-flops

- Decrypted key produces correct functional outputs : Nil
- Decrypted key produces corrupt functional outputs : `c2670`, `c3540`, `c5315`, `c7552`, `k2`, `seq`, `ex5`, `ex1010`, `apex2`, `apex4`, `c432`, `c499`, `c880`, `i4`, `i7`, `i8`, `i9`, `c1908`, `c1355`, `dalu`, `des`
- Success rate = $\frac{21}{21}$ cases = 100%

Results for 5% combinational (TOC13'XOR) encryption and encrypting all output flip-flops

- Decrypted key produces correct functional outputs : `i9`, `des`, `c432`, `c499`, `c3540`, `c880`, `c1908`
- Decrypted key produces corrupt functional outputs : `apex2`, `apex4`, `i4`, `i7`, `i8`, `ex1010`, `ex5`, `k2`, `seq`, `dalu`, `c1355`, `c5315`, `c2670`, `c7552`
- Success rate = $\frac{14}{21}$ cases $\approx 67\%$

Results for 5% combinational (TOC13'XOR and PO) encryption and encrypting all output flip-flops

- Decrypted key produces correct functional outputs : Nil
- Decrypted key produces corrupt functional outputs : `apex2`, `apex4`, `i4`, `i7`, `i8`, `ex1010`, `ex5`, `k2`, `seq`, `dalu`, `c1355`, `c5315`, `c2670`, `c7552`, `i9`, `c432`, `c499`, `c1908`, `c880`, `c3540`, `des`,
- Success rate = $\frac{21}{21}$ cases $=$ 100%

Results for 5% combinational (TOC13'MUX) encryption and encrypting all output flip-flops

- Decrypted key produces correct functional outputs : `apex2`, `i4`, `i7`, `i8`, `i9`, `ex1010`, `ex5`, `k2`, `seq`, `dalu`, `des`, `c432`, `c499`, `c1355`, `c5315`, `c7552`, `c3540`, `c2670`, `c880`, `c1908`
- Decrypted key produces corrupt functional outputs : `apex4`
- Success rate = $\frac{1}{21}$ cases $\approx 5\%$

Results for 5% combinational (TOC13'MUX and PO) encryption and encrypting all output flip-flops

- Decrypted key produces correct functional outputs : Nil
- Decrypted key produces corrupt functional outputs : `apex4`, `apex2`, `i4`, `i7`, `i8`, `i9`, `ex1010`, `ex5`, `k2`, `seq`, `dalu`, `des` ($\approx$ 2 hours), `c432`, `c499`, `c1355`, `c5315`, `c880`, `c1908`, `c3540`, `c7552`, `c2670`,
- Success rate = $\frac{21}{21}$ cases $\approx$ 100%

Theorem

Using virtual gates (shown in slide 62), it is possible to find portion of correct combinational key (except the encryption bits at POs)

Theorem

It is not possible to find the encryption bits at POs

Conclusion

Hence, our methodology in adding XOR-based encryption gates at POs, is ROBUST (in the sense that it is not possible to reformulate and solve for the PO encryption key bits, hence not possible to decrypt the complete combinational key).

Boundary Encryption- Experiment 1

- I have added XOR-type encryption key gates to all 7 outputs of `c432` circuit and encrypted all output flip-flops
- The *functionally correct key* is 0000000 (singleton equivalence class)
- However, running ScanSAT on scan unrolled version of above boundary encrypted `c432` circuit, gave 0010000, hence corrupting functional output
- Thus, boundary encryption produces SAT-attack Resilient Circuits.
- No need to add key gates deep inside combinational logic.

Boundary Encryption- Experiment 2

- Everything same as experiment 1, however number of encrypted POs is variable
- 7, 6, 5, 4, 3, 2: decrypted functionally incorrect key
- 1 : decrypted functionally correct key
- So, minimum 2 POs need to be encrypted. This is what I have manually done in all the experiments, across the 21 benchmarks, spanning the 5-encryption-schemes. Hence, the success rate reached 100 % by just encrypting just 2-3 POs.

Advantages of Boundary Encryption

- We need to add few XOR/XNOR-type encryption key gates, just at POs, and no need to add them deep inside combinational logic.
- This makes the encryption scheme very area-efficient. The key gates can be added to timing-uncritical POs.
- Area, Timing and Power friendly.
- This makes this method seamless integrable to industry practice
- The method scales easily from IP-level to SoC-level.

Advantages of Boundary Encryption/Locking

- Logic Locking techniques lock only the combinational logic, hence vulnerable to scan-based SAT-attack, EDT-Bypass mode;
- Scan Locking locks only the scan chains, using Encrypt-Flip-Flop. However, ScanSAT models this as an instance of logic locking problem, hence also vulnerable to SAT-attack;
- Boundary Locking combines both intelligently, to thwart SAT-attack.
- Compare area overhead for all circuits reported in Table II of Encrypt Flip-Flop paper.
- Report running times

Thank you!