

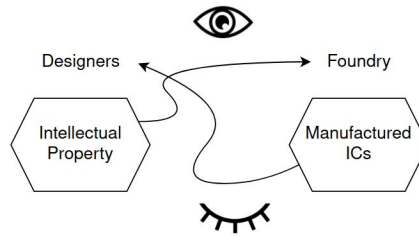


IP Protection for Logic-in-Memory

How to protect your memory?



Motivation



We are relying on Zoom, and not sitting in front, because we trust our chips. It will work, that's what we assume.

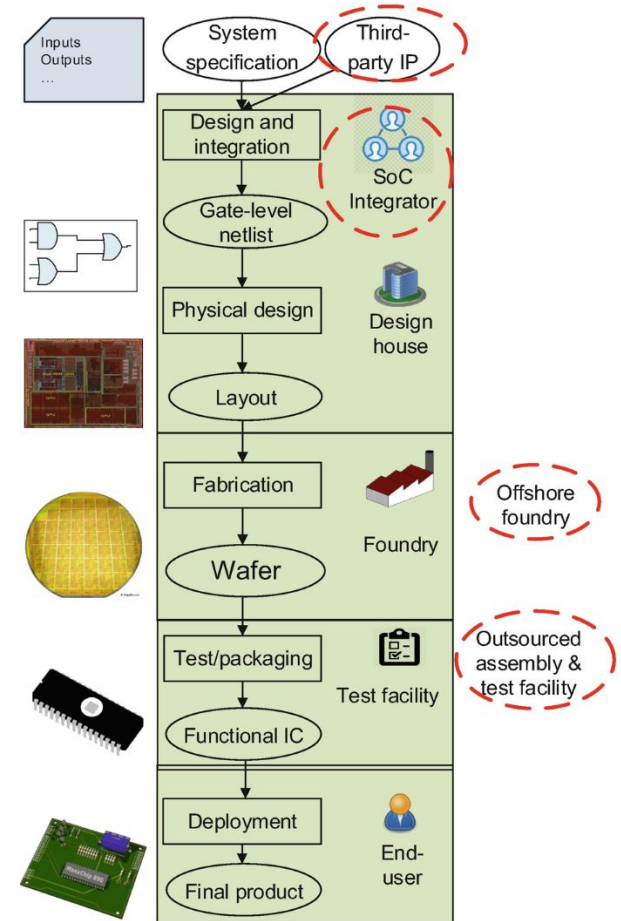
Suppose it doesn't. Why? **Reliability**

Maybe you bought from a bad dealer, or your dealer gave you an unreliable device. Maybe it was unintended design fault, or manufacturing defects, or exchange from a worse but similar looking product. **Integrity**

The higher world assumes ICs would do and only do what it's supposed to, but maybe the foundry inserted a trojan, or did the testing facility? **Security**

Who can we trust? Apparently no one except the designer

Therefore the field of **IP protection**. I will focus specifically for Logic-in-Memory[2] architecture, after some background.



Outline

Existing Wars

Will discuss defenses:

- Watermarking
- Active Metering
- Camouflaging
- Logic Locking

Attacks:

- Brute Force
- Sensitization
- SAT Attack

LIM Architecture

Know how your flash memory chips can act as computing units.

Discuss about merits and shortcomings

WFSM Strategy

Introduce a novel technique of implementing locking in LIM.

Includes abstract implementation of the defense and a brute force attack analysis

Circuit Wars: Defenses

Watermarking & Metering

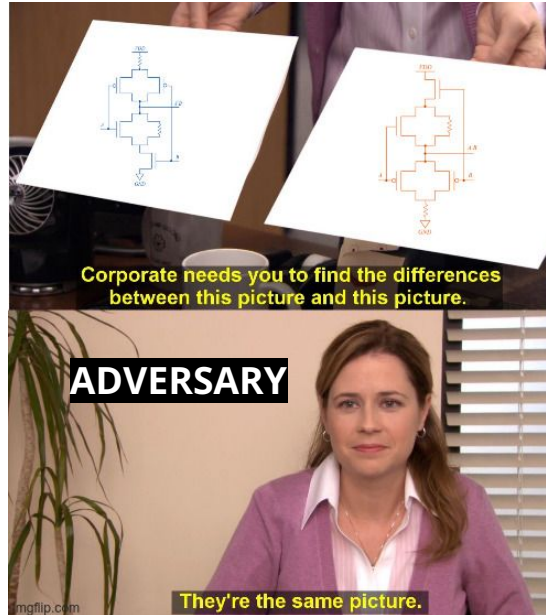
Watermark

IP with set of constraints, solve optimisation problem to prove originality

Metering

Keep track of functional ICs out there

Camouflaging



Logic Locking

Lock-and-Key

There are key bits as extra input from user. The circuit works correctly **always** only when the correct key is inserted

Straightforward:
forge, tamper, try
entire key space

Find input patterns
that make key bits
primary outputs

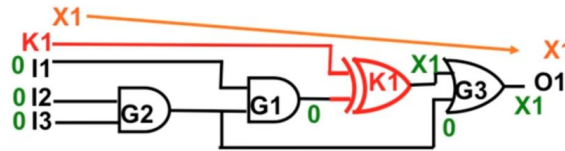
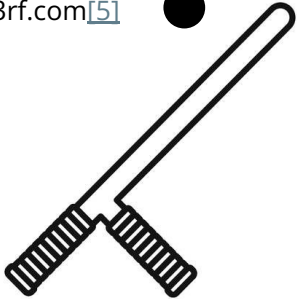
Iterate over key
space, partition it
everytime

BruteForce

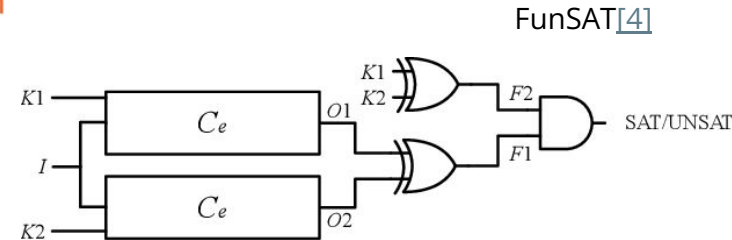
Sensitizaⁿ

SAT Attack

123rf.com[5]



Advances in Logic Locking[3]



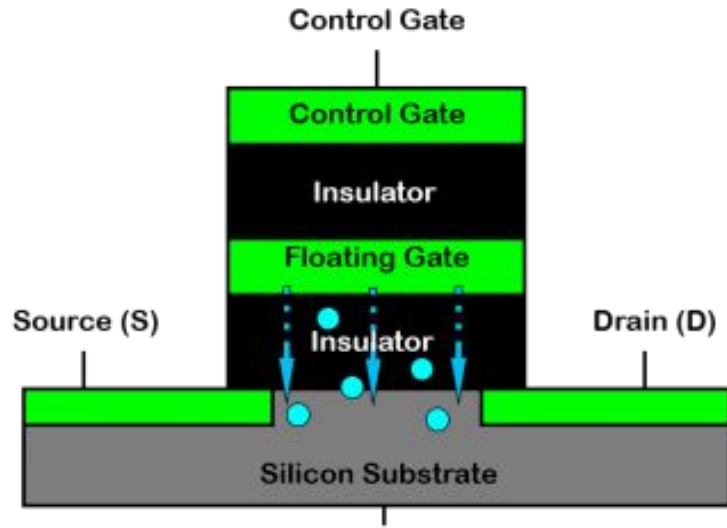
FunSAT[4]

Logic-in-Memory

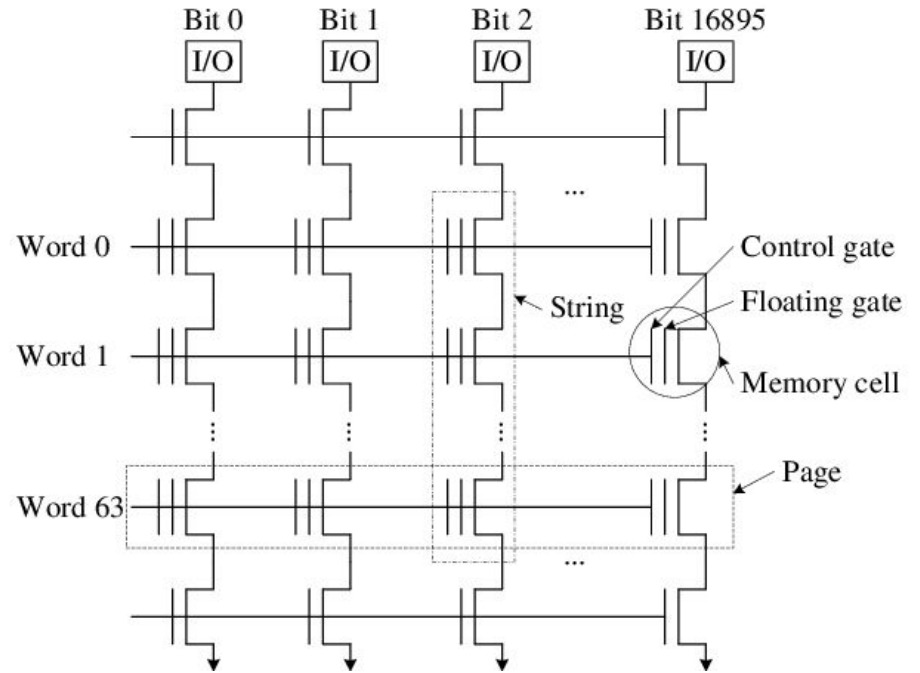
NAND Flash Architecture

Use the similarity of NAND strings to, NAND gates, treating a single string as multipliers and parallel strings as adders.

One can convert any function to Sum-of-product form.



Cactus-tech[7]



Research Gate[8]

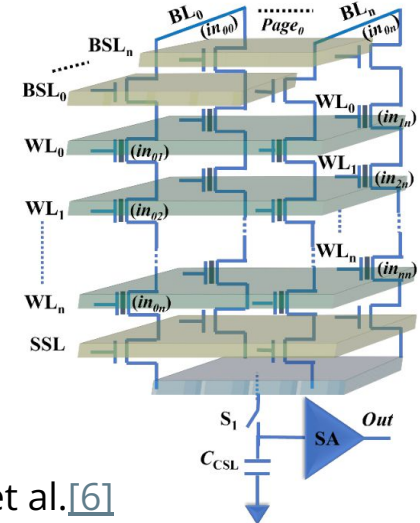
$$F = AB'C + DC' \quad A=0; B=1; C=1; D=0$$

Merits

- Capable of calculating an SOP form with 177 literals and 2^{14} minterms, under some constraints
- Read time of the function that's already stored is in nanoseconds
- Once a protocol is invented that streams bits into NAND cells that's compatible to be used in SOP form, this will lead to reason for inputting bits and storing them in such an order
- Performing computation with no side overhead, other than the implementing protocol above
- Is intrinsically camouflaged

Demerits

- Can't be used as a full fledged processing unit, lifecycles of NAND cell is limited to 10^5 writes, compared to 10^9 cycles of instruction execution done on modern processors
- Has some constraints including keeping track of input bits, and entire page sharing single read lines




Sahay et al.[6]

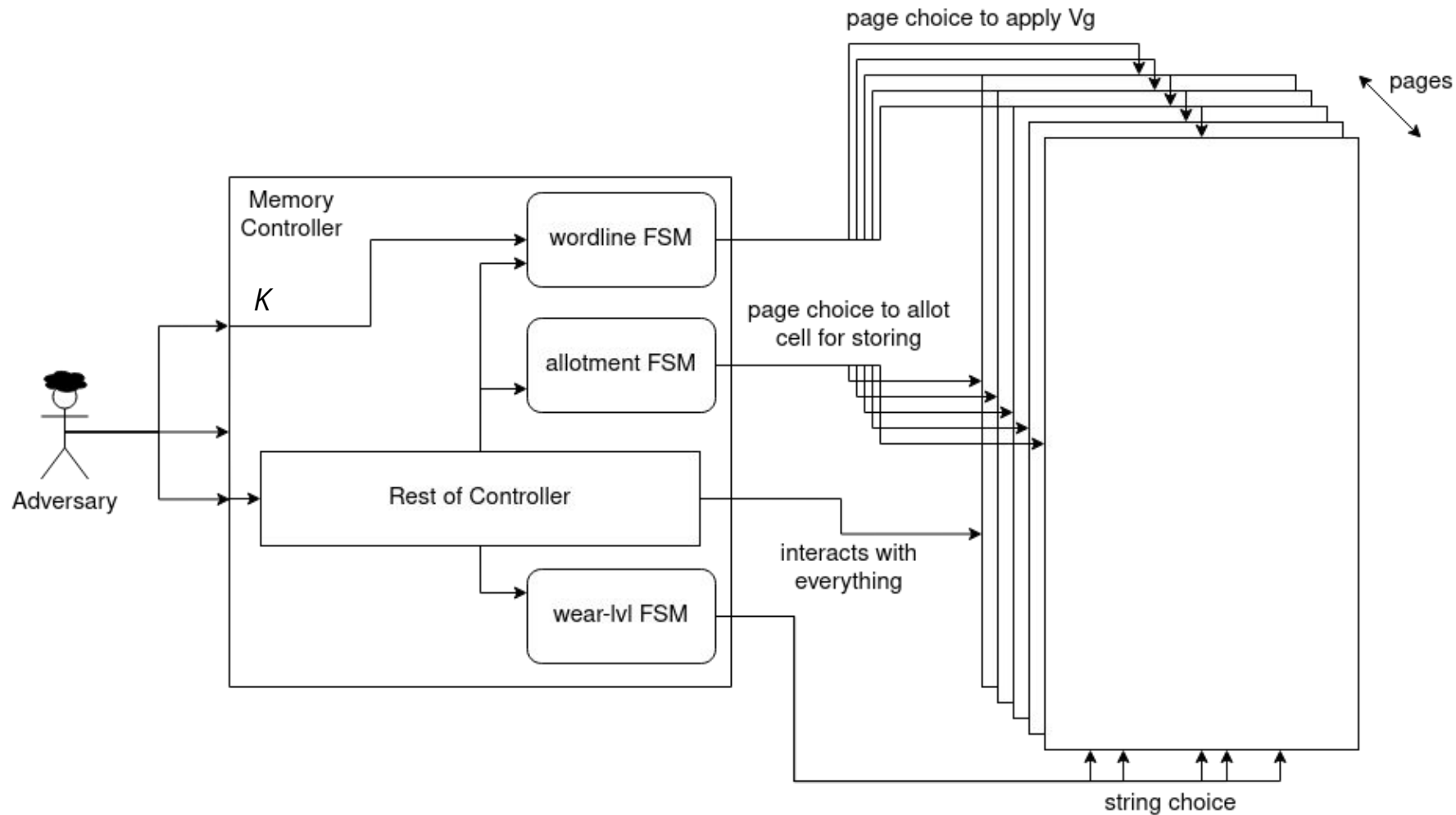


WFSM Strategy

Use WordLine voltages to corrupt the output

Difference lies in allotment state having initialization from the designer, wordline state having init from user





Experimenting with Simulations

Implemented the architecture with the defense, can be used as module by any attacker

→ **Simulation** `python test.py -correct_key 400`

For function 0:

```
[81, 3, 0, 8, 0, 3, 0, 27, 1, 7, 19, 8, 8, 5, 0, 6, 1, 3, 2, 0, 1, 4, 5, 2, 3, 13, 0, 0, 0, 0, 0, 8, 1, 7, 6, 17, 3, 4, 19, 1, 1, 12, 0, 1, 8, 0, 13, 8, 0, 5, 5, 18, 0, 5, 0, 0, 9, 0, 3, 0, 4, 4, 1, 3, 3, 1, 14, 1, 1, 0, 6, 0, 8, 2, 3, 6, 30, 0, 1, 0, 2, 0, 2, 0, 7, 8, 8, 1, 4, 5, 0, 12, 0, 5, 0, 0, 1, 10, 1, 0, 178, 2, 2, 0, 1, 0, 0, 1, 2, 2, 0, 1, 0, 1, 0, 4, 5, 2, 1, 0, 0, 4, 0, 12, 6, 3, 2, 0, 0, 0, 0, 0, 1, 6, 0, 1, 0, 0, 0, 3, 1, 2, 1, 0, 0, 0, 0, 0, 0, 3, 0, 0, 1, 2, 1, 0, 0, 5, 4, 0, 1, 3, 1, 0, 1, 0, 2, 0, 4, 1, 0, 2, 0, 5, 0, 0, 0, 2, 9, 2, 2, 4, 0, 0, 3, 4, 1, 0, 1, 0, 0, 2, 0, 0, 0, 4, 4, 6, 2]
```

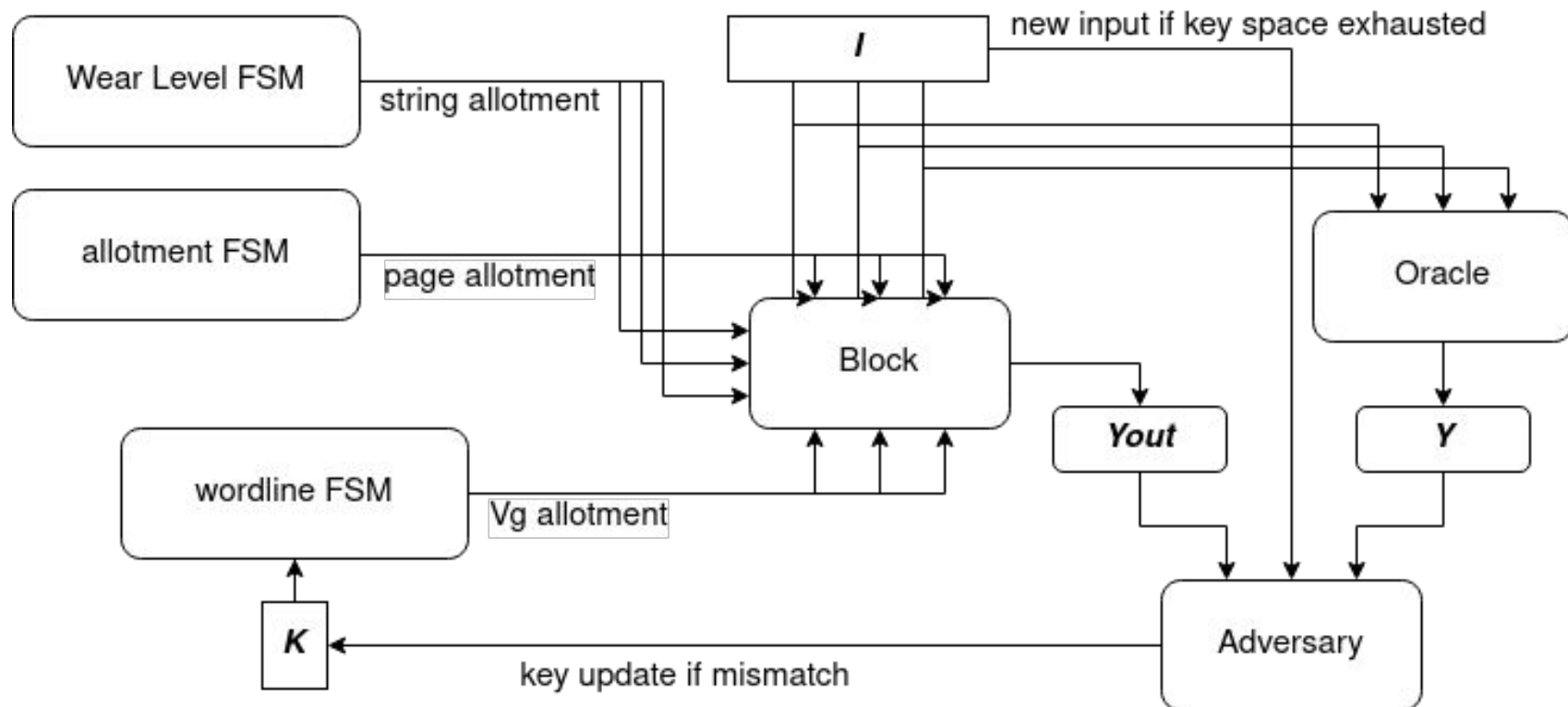
For function 1:

```
[3, 1, 178, 59, 1, 37, 122, 65, 102, 75, 112, 178, 106, 61, 66, 84, 46, 178, 71, 78, 0, 178, 29, 178, 5, 178, 95, 100, 75, 32, 69, 40, 47, 83, 74, 28, 178, 178, 8, 19, 8, 11, 0, 53, 58, 63, 84, 41, 62, 84, 8, 178, 178, 40, 29, 18, 7, 78, 81, 178, 75, 109, 53, 58, 15, 36, 74, 32, 54, 10, 30, 19, 8, 63, 116, 55, 28, 81, 24, 13, 75, 53, 178, 178, 55, 178, 33, 22, 9, 64, 35, 153, 77, 68, 120, 111, 67, 24, 56, 0, 178, 45, 178, 23, 10, 127, 70, 154, 62, 37, 58, 15, 4, 39, 77, 35, 178, 46, 35, 22, 11, 0, 53, 26, 15, 20, 178, 102, 35, 137, 78, 178, 58, 63, 34, 23, 12, 1, 167, 59, 64, 55, 12, 1, 38, 59, 32, 5, 10, 15, 178, 24, 77, 66, 7, 12, 178, 135, 124, 64, 53, 58, 15, 4, 25, 49, 70, 3, 48, 149, 120, 178, 178, 87, 14, 83, 105, 83, 0, 178, 28, 66, 6, 4, 15, 6, 170, 126, 150, 137, 128, 84, 25, 178, 98, 178, 178, 178, 178, 29]
```

Usage count per string: [5440. 5440. 5441. 5439. 5441. 5441. 5440. 5440. 5438. 5438.]

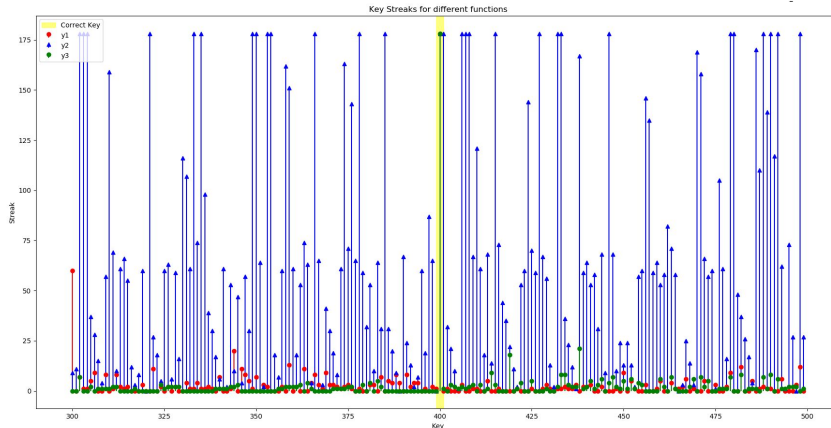
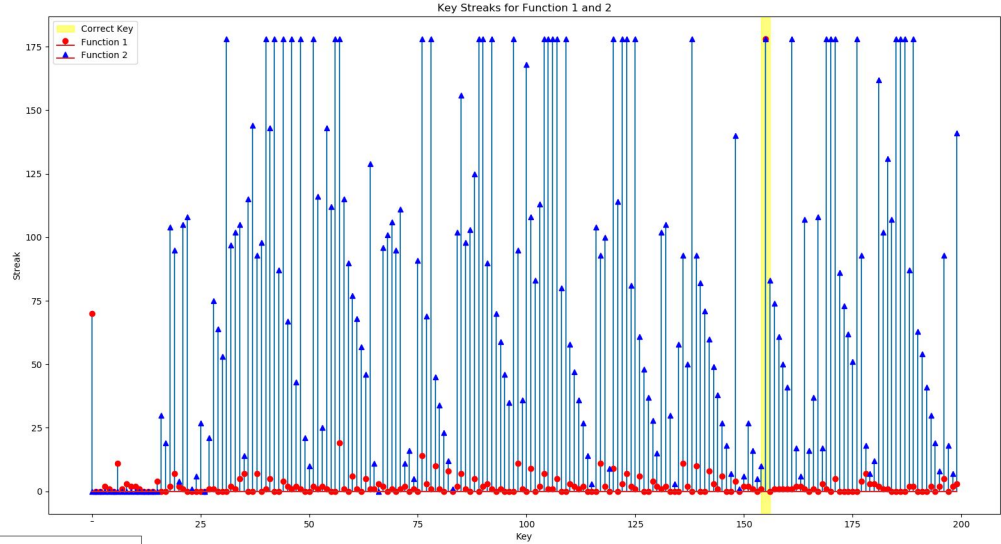
Each flip requires 3ms to complete, therefore total time taken: 163194.0 ms

Brute-forcing



Results:

The above shows a keyspace of 200, with number of flips in total equal to 54398, for trying two functions of form $[(3, 2), (0, 1)]$ & $[(2, 4), (5, 6)]$



Implementation of Linear Congruence Generator as FSM. Generates a sequence of numbers (states) of period 177 (corresponding to each cell). Therefore the key repeats in every 177 key range. Improve FSM to improve key space

3 functions search through key space

Conclusion

This project concludes with introduction of a locking defense strategy WFSM for LIM, inspired by Active Metering and Watermarking techniques.

I've benchmarked the performance of this method using a basic model of a markov chain and small key space, against a brute-force attack.

It has also been shown that existing algorithmic attacks like SAT attack or it's any existing variation would be unable to work on this due to it's requirement of CNF, which can be made only for combinational circuits.

There is a possibility of extending this idea to domain of Memory protection against unauthorised use, being a prospective future work.

Acknowledgements

Extremely thankful to Bhogi Satya Swaroop for providing relevant papers and helping me understand the LIM architecture.

I would like to thank the people at NeuroCHaSe lab for being ever inviting.

I thank Professor Shubham Sahay for providing an approachable problem statement, and correcting me on the way.

I finally thank you, the panel for listening through this project report and for your criticism.

This has been presented by B.Anshuman [200259], EE-UG.