

Why Your Password Can Be Stolen Without Ever Being Guessed

Implementing Side-Channel Attack on FPGA

Presented by:

- Shubham Mohanty
- Shlok Mehndiratta
- Prashant Gupta



Defining Side-Channel Attacks

Most attacks try to crack the logic.
Side-channel attacks don't bother.

Traditional Attack

- Tries to reverse-engineer the algorithm
- Guesses keys, breaks encryption mathematically
- Attacks what the system does

Side-Channel Attack

- Ignores the algorithm entirely
- Observes how the system behaves — time taken, power drawn, signals emitted
- The system works perfectly and still leaks your secret



Overview of Two Designs

FSM-Based Sequential Checking

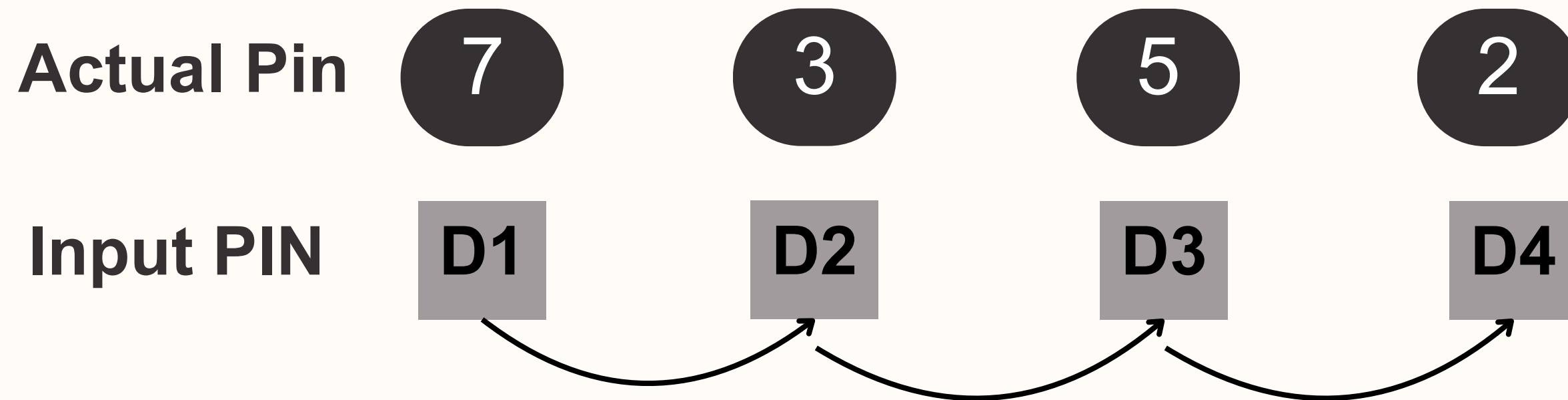
Finite State Machine (FSM) design processes digits one-by-one; advances states on matches, exits early on mismatch; execution time scales with correct prefix length—for PIN 7532, 7-5-3-2 takes max cycles, 0-0-0-0 minimal; vulnerable as time directly correlates to matches.

XNOR-Based Parallel Checking

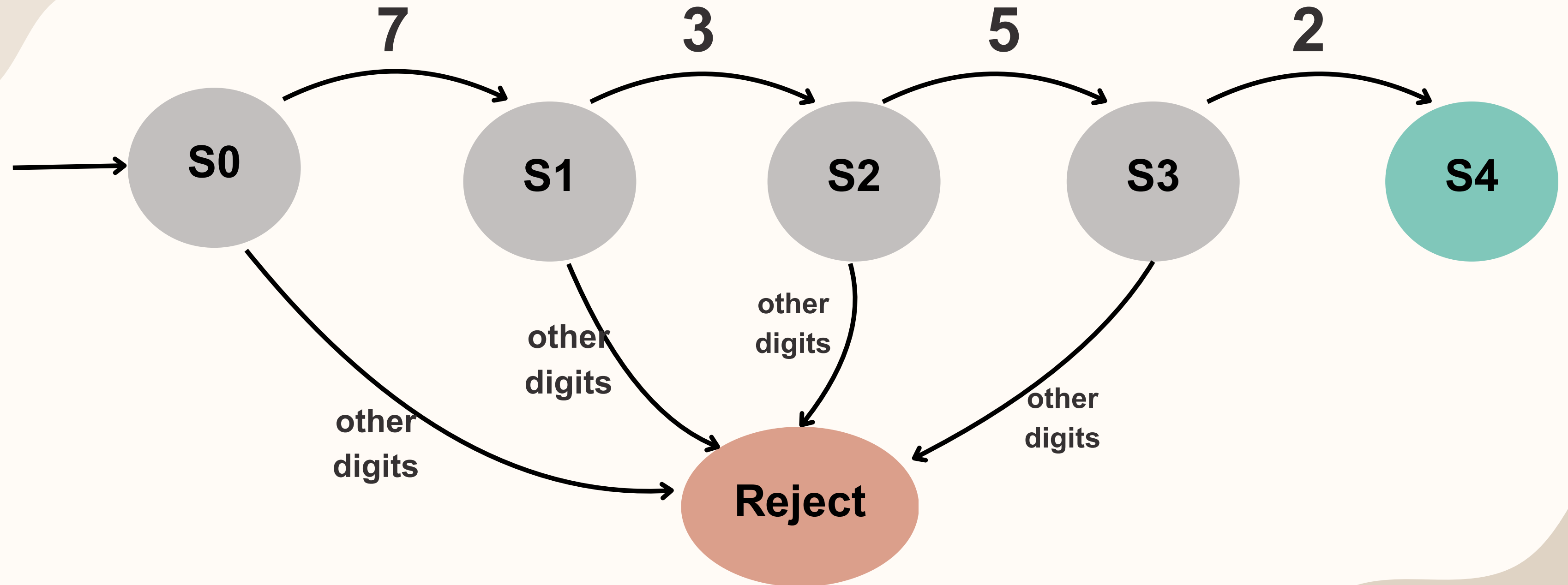
XNOR gates compare all digits simultaneously; outputs ANDed for final access decision; constant cycles regardless of input—0 matches or full match identical; no sequential dependency ensures timing uniformity; ideal for side-channel resistance.

FSM Based PIN Verification

This Mechanism compares the PIN digit by digit and moves to the next place only after a successful comparison.



FSM for PIN 7352



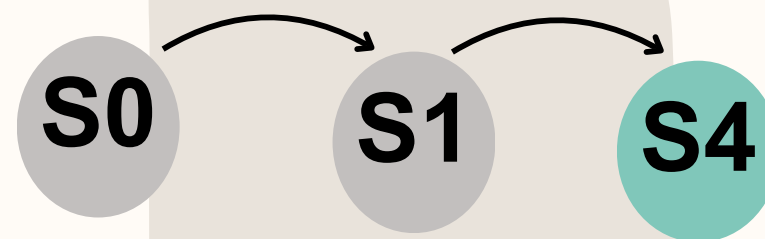
Verilog Module for FSM

```
// =====  
// PIN CHECK FSM  
// =====  
always @(posedge clk) begin  
    case(state)  
        IDLE: begin  
            cycle_count <= 0;  
            success <= 0;  
            state <= D1;  
        end  
  
        D1: begin  
            cycle_count <= cycle_count + 1;  
            if (TEST_D1 == SECRET_D1) state <= D2;  
            else state <= DONE;  
        end  
  
        D2: begin  
            cycle_count <= cycle_count + 1;  
            if (TEST_D2 == SECRET_D2) state <= D3;  
            else state <= DONE;  
        end  
  
        D3: begin  
            cycle_count <= cycle_count + 1;  
            if (TEST_D3 == SECRET_D3) state <= D4;  
            else state <= DONE;  
        end  
  
        D4: begin  
            cycle_count <= cycle_count + 1;  
            if (TEST_D4 == SECRET_D4) begin  
                success <= 1;  
            end  
            state <= DONE;  
        end  
  
        DONE: begin  
            state <= DONE;  
        end  
    endcase  
end
```

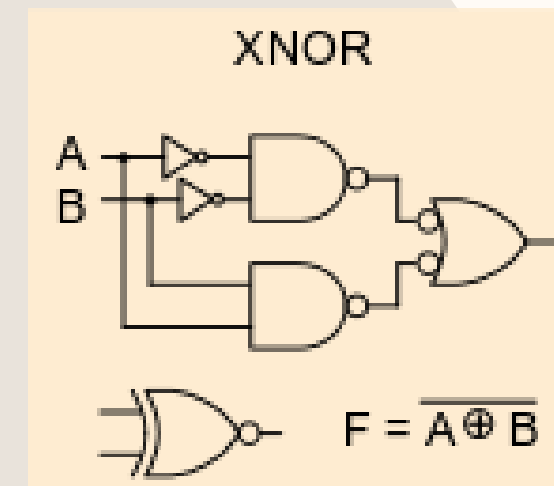
Our Implementation



**Shrike Lite
FPGA Dev Kit**



**FSM Based
PIN
Verification**

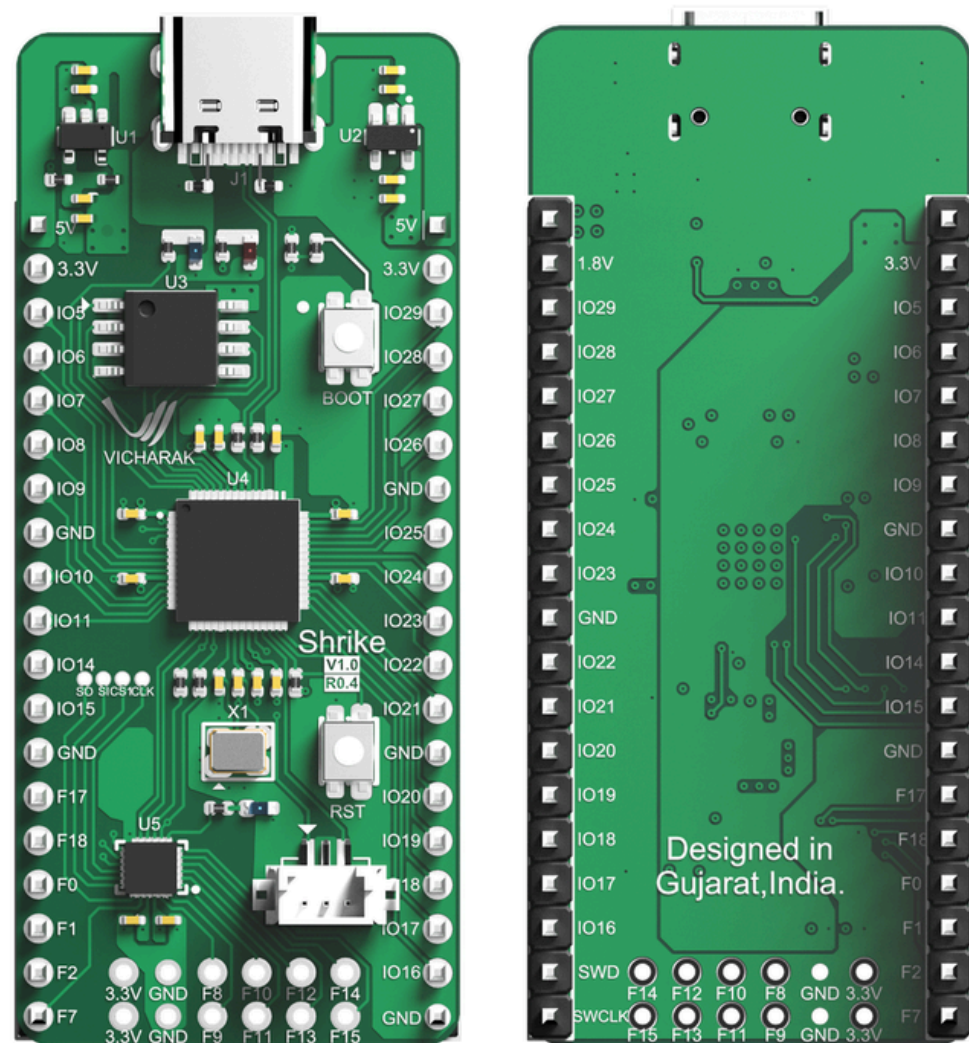


**XNOR Based
Verification**



**Response
Time
Analysis**

Meet the Hardware: Shrike Lite FPGA



What's on the Board

- FPGA with 1120 logic cells — runs our PIN checker hardware
- RP2040 ARM microcontroller — handles programming and interface
- Two onboard LEDs — used as our output channel
- USB Type-C for power and programming
- Breadboard compatible — easy to prototype and wire up

Why We Used It

- FPGA lets us implement custom digital logic in Verilog
- We control exact clock cycles — essential for timing experiments
- Bitstream Flashing software — RenesasGoConfigure
- Compact and real — this is actual hardware, not simulation

Experimental Pipeline

Our Actual PIN is :
7352
(TRUE PIN)



We feed the Input PIN
to the System to verify
PIN



System verifies the PIN
Returns output with a
certain response time if
PIN is correct or not

- 4 Digit PIN to Unlock the System
- Response Time of System = 0.02 microsec = 1 unit
- For n unit Response Time → LED blinks n times

ANALYSE the response time for different input combinations and then crack the Correct Digit based on Observation

```
ers:  
api.statuses.user_timeline(  
    response.data), 'tw  
ata) != 0:  
    onse.data[0]['created_at']  
    tetime.strptime(ltdate, '%a %b %  
    etime.now()  
    (today-ltdate2).days  
    g < daywindow:  
    t i.screen_name, 'has tweeted in th  
    altweets += len(response.data)
```

Observation

Total Combinations

$$\begin{array}{c} \square \\ \times 10 \end{array} + \begin{array}{c} \square \\ \times 10 \end{array} + \begin{array}{c} \square \\ \times 10 \end{array} + \begin{array}{c} \square \\ \times 10 \end{array} = 40 \text{ Combinations}$$

Let us take **INPUT PIN** as 1000

We will measure the response time by the number of blinks of the LED

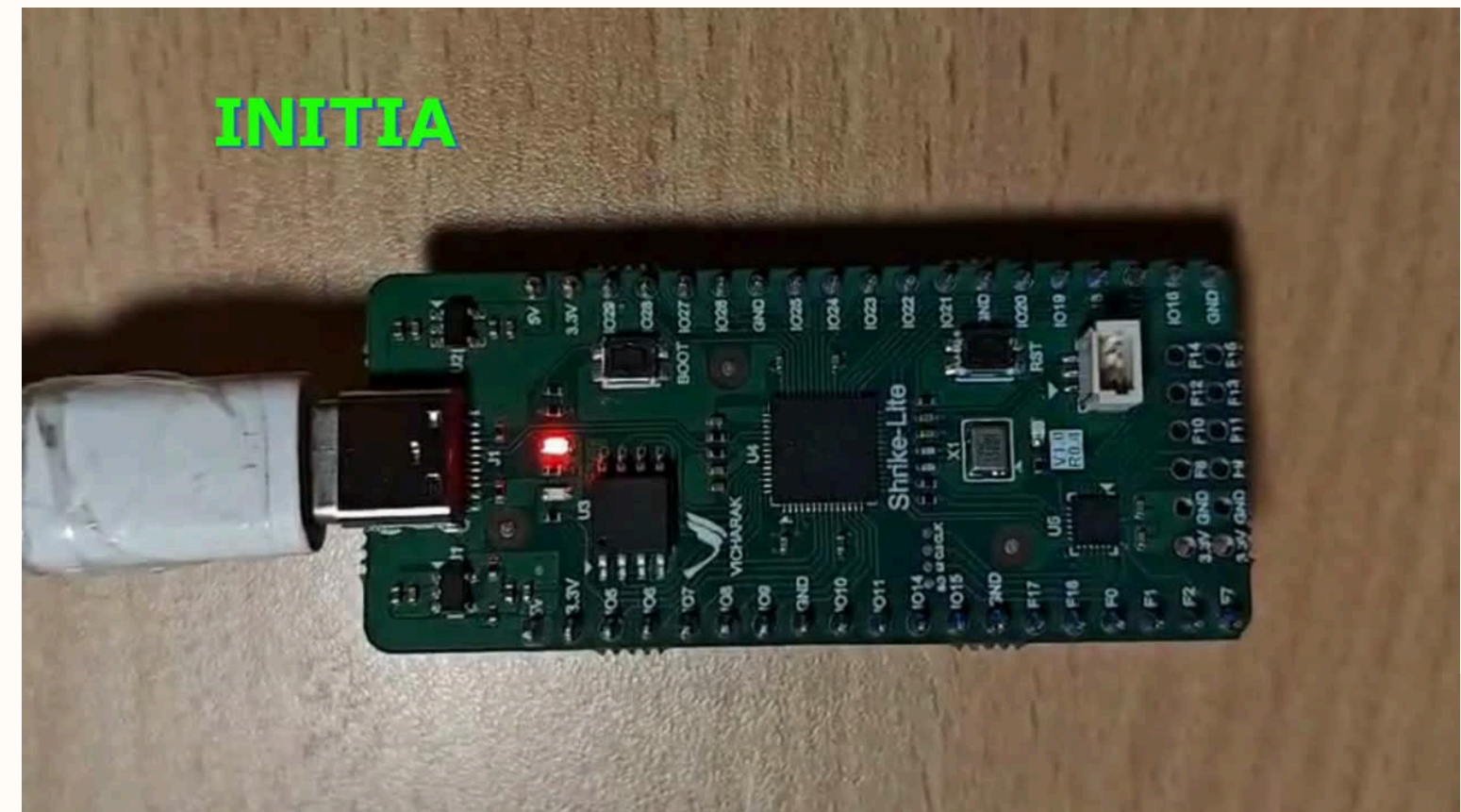


Observation

Same Response Time for all x000 , $x \in [0,9] / \{7\}$

INPUT Combinations	RESPONSE TIME (no. of Blinks)	CORRECT DIGIT
1000, 2000, 3000, 4000, 5000, 6000, 8000, 9000, 0000	1	---

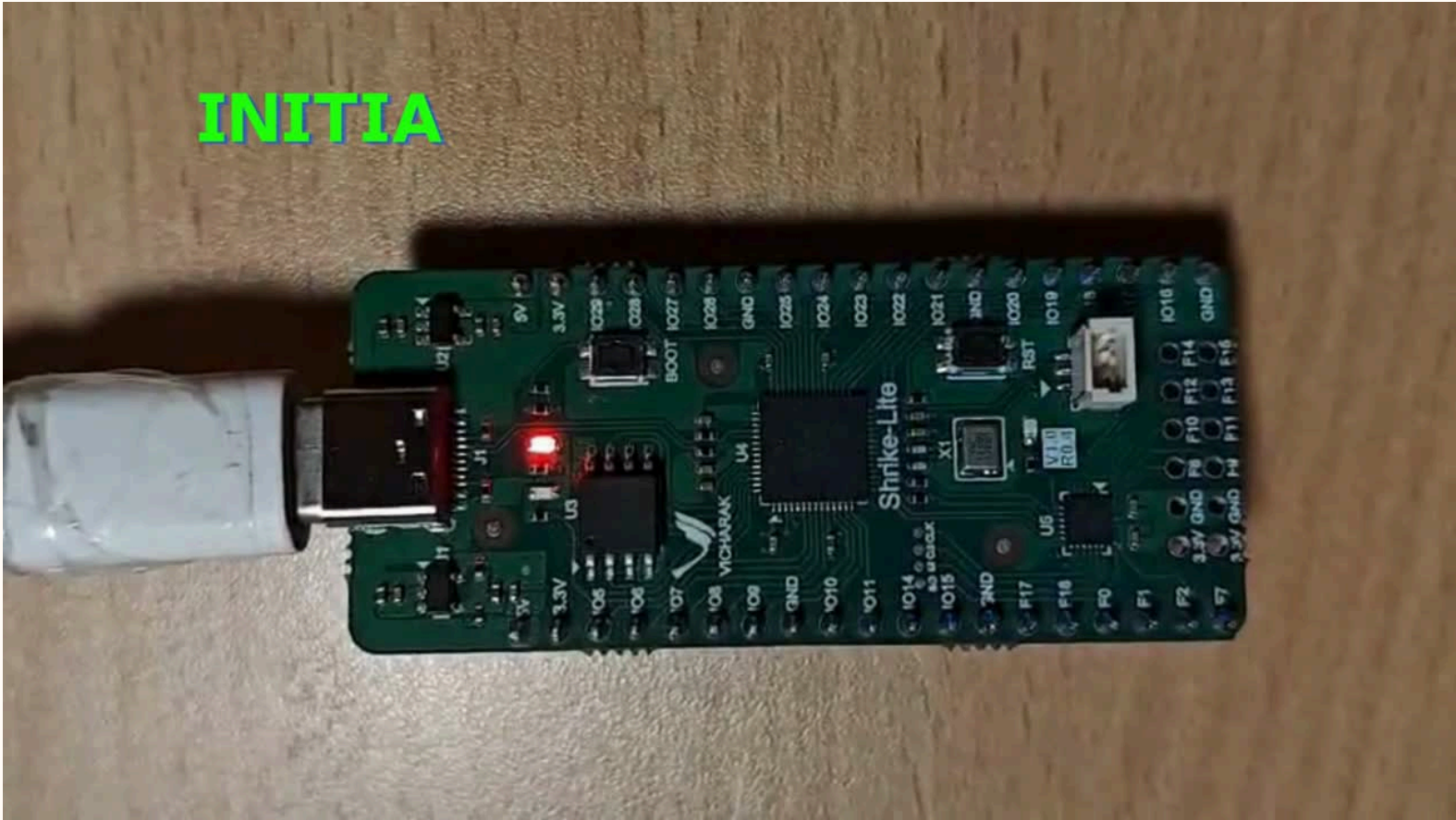
Let us take INPUT PIN as 7000,



Observation

INPUT Combinations	RESPONSE TIME (no. of Blinks)	CORRECT DIGIT
1000, 2000, 3000, 4000, 5000, 6000, 8000, 9000, 0000	1	---
7000	2	7xxx

Let us take INPUT PIN as 7100,



Observation

INPUT Combinations	RESPONSE TIME (no. of Blinks)	CORRECT DIGIT
1000, 2000, 3000, 4000, 5000, 6000, 8000, 9000, 0000	1	---
7000	2	7xxx
7100, 7200, 7400, 7500, 7600, 7700, 7800, 7900	2	7xxx

Same Response Time for all 7x00 , $x \in [0,9] \setminus \{3\}$

Observation

Let us take INPUT PIN as 7300,



7100, 7200, 7400, 7500, 7600, 7700, 7800, 7900	2	7xxx
7300	3	73xx

Observation

Similarly, for all INPUT PINs as 73x0, $x \in [0,9] \setminus \{5\}$

Number of Blinks = 3

7100, 7200, 7400, 7500, 7600, 7700, 7800, 7900	2	7xxx
7300	3	73xx
7310, 7320, 7330, 7340, 7360, 7370, 7380, 7390	3	73xx

Observation

Let us take INPUT PIN as 7350,



7300	3	73xx
7310, 7320, 7330, 7340, 7360, 7370, 7380, 7390	3	73xx
7350	4	735x

Observation

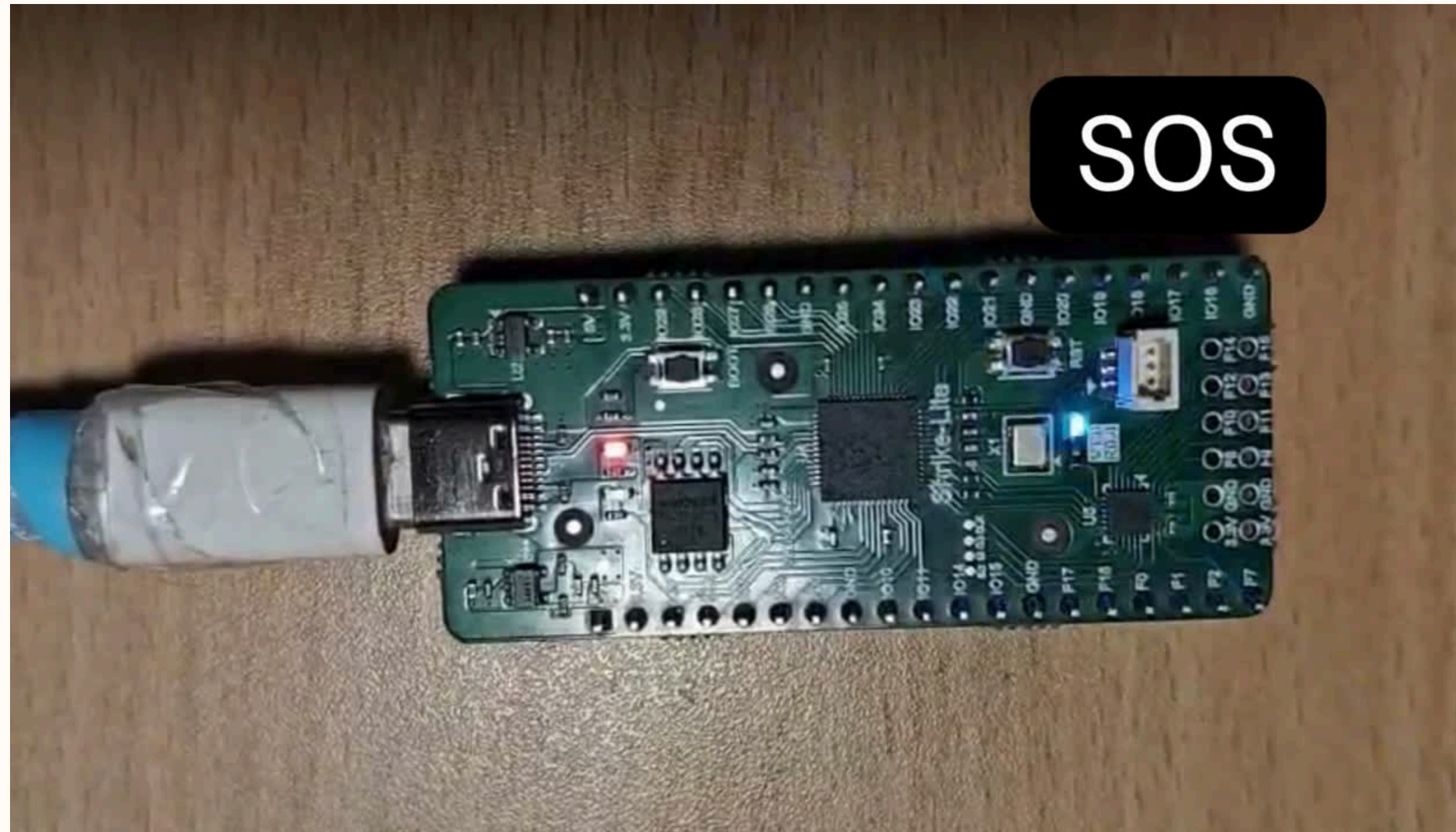
Similarly, for all INPUT PINs as 735x, $x \in [0,9] \setminus \{2\}$

Number of Blinks = 4

7300	3	73xx
7310, 7320, 7330, 7340, 7360, 7370, 7380, 7390	3	73xx
7350	4	735x
7351, 7353, 7354, 7355, 7356, 7357, 7358, 7359	4	735x

Observation

Now, for INPUT PIN as 7352,
The LED will blink continuously, **PIN CRACKED**



Observation

BRUTE Force Method with at max 40 iterations

INPUT Combinations	RESPONSE TIME	CORRECT DIGIT
1000, 2000, 3000, 4000, 5000, 6000, 8000, 9000, 0000	1	---
7000	2	7xxx
7100, 7200, 7400, 7500, 7600, 7700, 7800, 7900	2	7xxx
7300	3	73xx
7310, 7320, 7330, 7340, 7360, 7370, 7380, 7390	3	73xx
7350	4	735x
7351, 7353, 7354, 7355, 7356, 7357, 7358, 7359	4	735x
7532	Continuous	7352

Correct Logic, Broken Security.

What the system does right

- Denies access for wrong PIN
- Grants access only for correct PIN
- No logical flaw exists

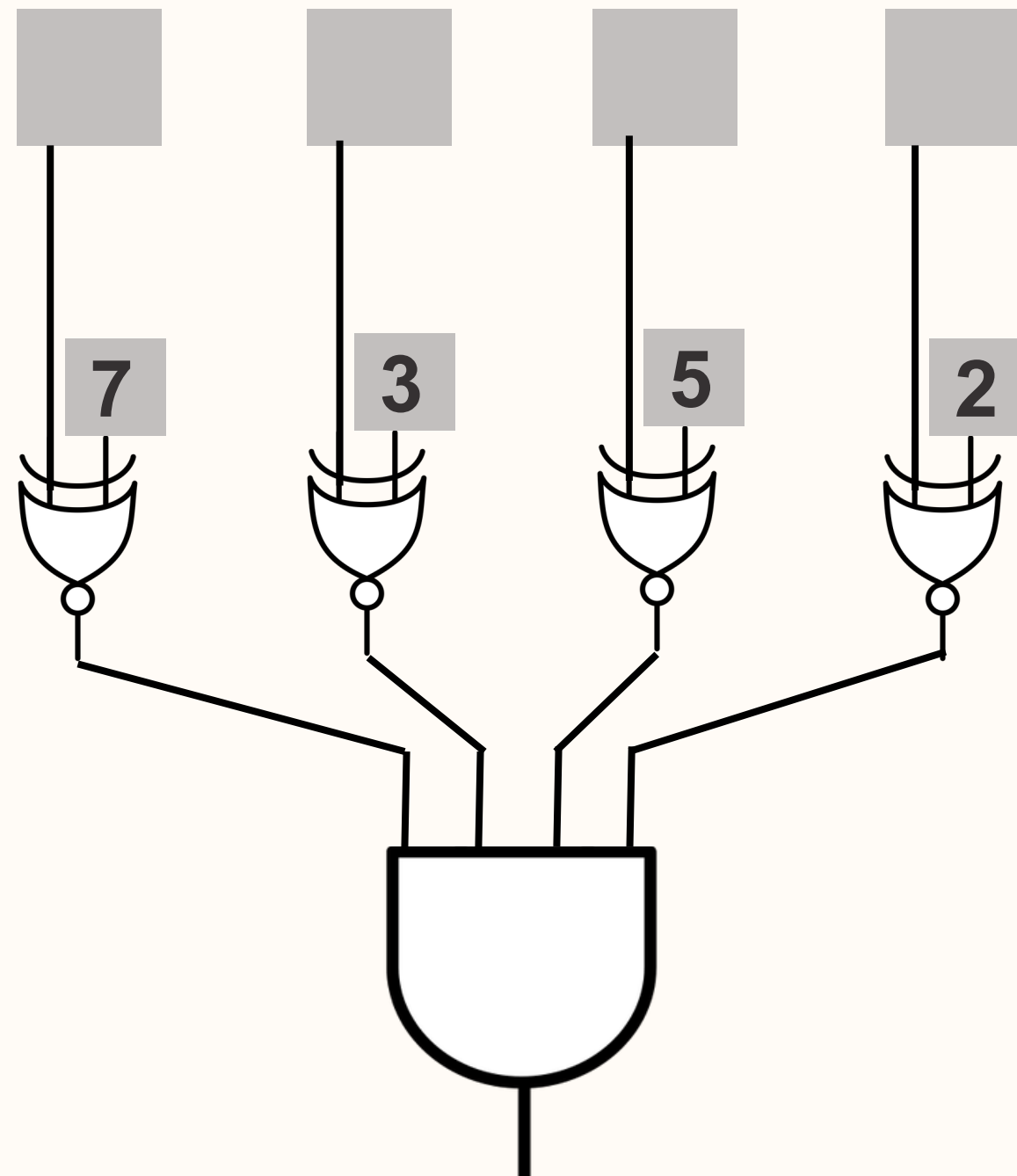
What the system leaks

- Tells you how many digits matched
- Leaks this through timing, not output
- No code change needed to exploit it

“The vulnerability is not in what the system outputs. It is in how long it takes to do so.”

Parallel XNOR-Based PIN Checking

Input PIN:



What Changes:

- No states, no sequence, no FSM
- All comparisons happen in the same clock cycle
- No early exit — all four digits always get checked
- **Timing carries absolutely no information**

How Hard is it to Crack Each Design?

The design choice doesn't just affect security — it determines how much work an attacker has to do.

FSM Sequential Design

Attack Complexity: **$O(n \times b)$**

- Attacker cracks one digit at a time
- For each digit, try $b = 10$ values
- Repeat for $n = 4$ digits
- Total guesses needed: $4 \times 10 = 40$

Timing tells the attacker exactly when to move to the next digit. Each step is independent and cheap.

XNOR Parallel Design

Attack Complexity: **$O(b^n)$**

- No timing information available
- Attacker must brute force all combinations
- Total guesses needed: $10^4 = 10,000$

Every guess looks identical in time. The attacker gets no feedback. Only exhaustive search works.

n = number of digits in the PIN (here, 4)

b = number of possible values per digit (0 through 9, so 10)

It's Not Just Theory - This Has Happened

1. **Intel & AMD:** Spectre & Meltdown (2018) Timing differences in CPU execution allowed attackers to read protected memory. Forced Intel, AMD, and ARM to redesign their processors. Every computer in the world was affected.
2. **OpenSSL:** RSA Timing Attack (2003) RSA decryption took slightly different time depending on the secret key. Attackers measured this remotely and recovered the private key. Emergency patch issued. Millions of web servers were at risk.
3. **Google & Signal:** KyberSlash (2023) Next-generation encryption adopted by Google Chrome and Signal messenger. Timing measurements on crafted inputs revealed secret key pairs. Even post-quantum encryption was not immune.
4. **Apple:** iLeakage (2023) Side-channel attack on Safari across all Apple M-series devices. Allowed attackers to read emails, passwords, and browsing history. No code execution needed — just timing observation.

“Security is not just about what your system does.
It is about what your system reveals while doing it.”

Thank You!

