PowerAlert: An Integrity Checker using Power Measurement

Ahmed Fawaz

11/16/2016

Submitted to IEEE S&P’17

Problem Description

How to validate the integrity of software against a slow and stealthy attacker without any trusted components in the machine?

- Trusting data from monitors such as Kobra

Solution Approach

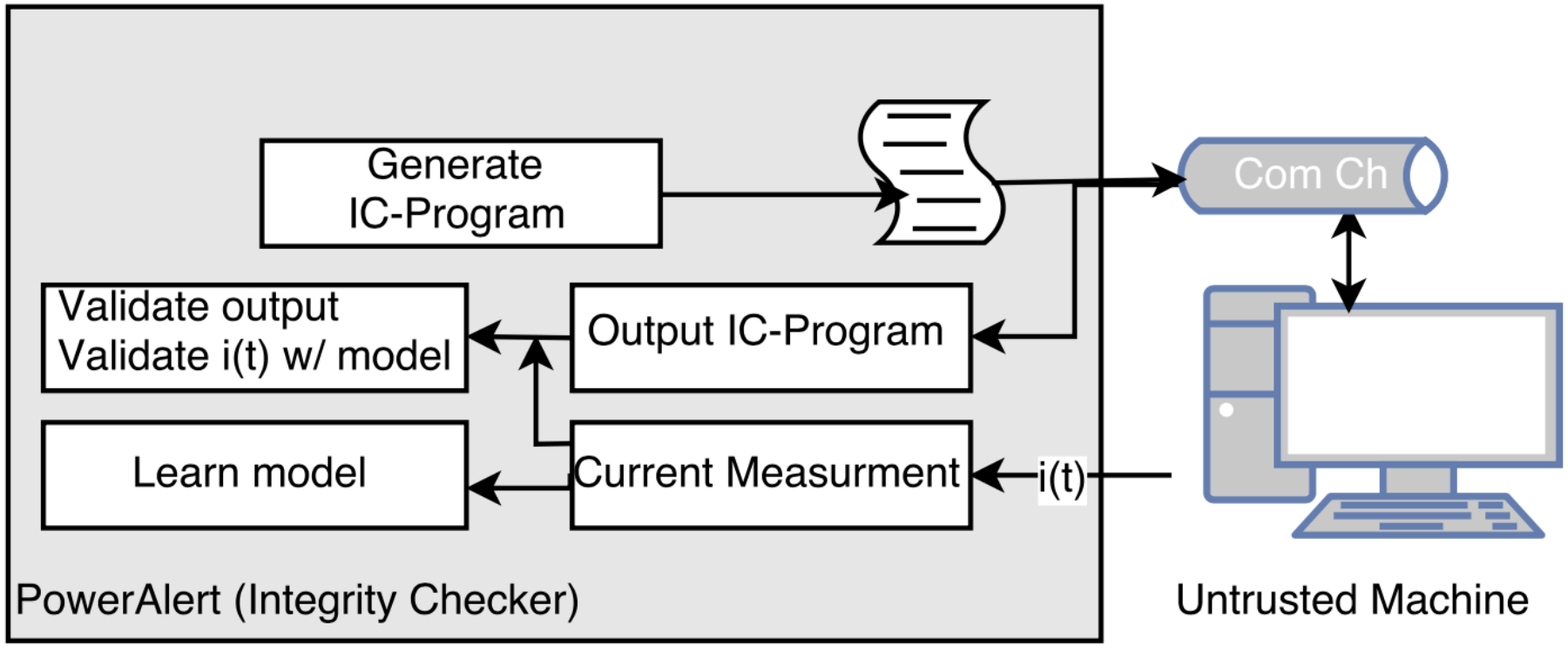


Fig. 1: The components of PowerAlert.

Threat Model

- Host machine is untrusted with attacker changing the kernel (rootkit)
- Attacker does not modify the hardware (clock rate, firmware,…)
- Attacker will attempt to deceive PowerAlert

Proxy Attack	Data Pointer Redirection	Static Analysis	Active Analysis	Attacker Hiding	Force Retraining
Use a remote machine to compute	Keep a clean copy of memory	Attempt to precompute	Learn how the program works	Hide when the checker runs	Learn a tampered model

Solution

- Diversity of the IC-program to thwart adaptation
- Measure current independently
- Run the checker at random times

PowerAlert Protocol

Verifier:

1. Start protocol
2. Generate the IC-program
3. Start measurement
4. Verify the output and the current measurements

Prover:

1. Pause all tasks
2. Load the IC-program
3. Run the IC-program
4. Return the result

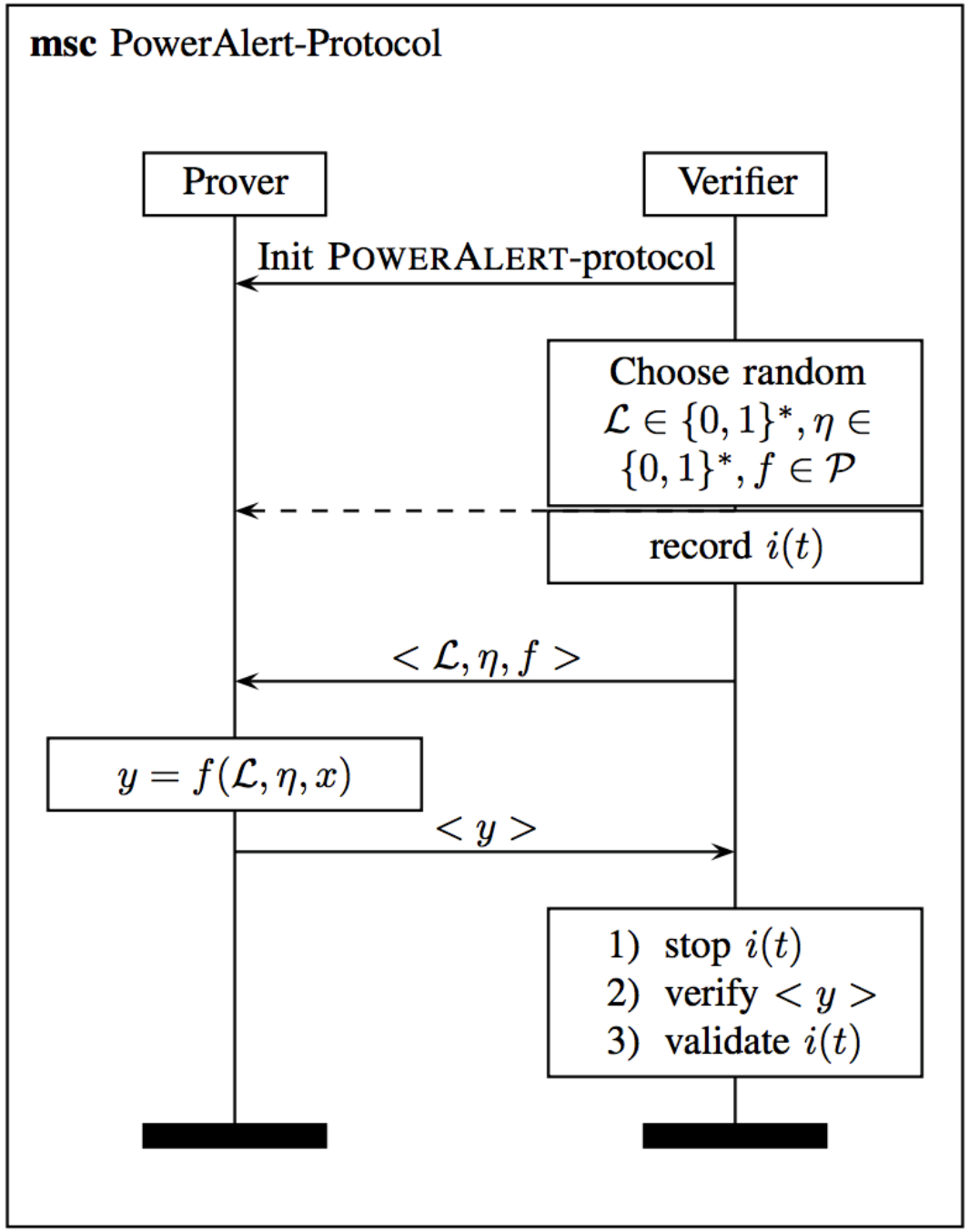


Fig. 2: Description of the POWERALERT-protocol

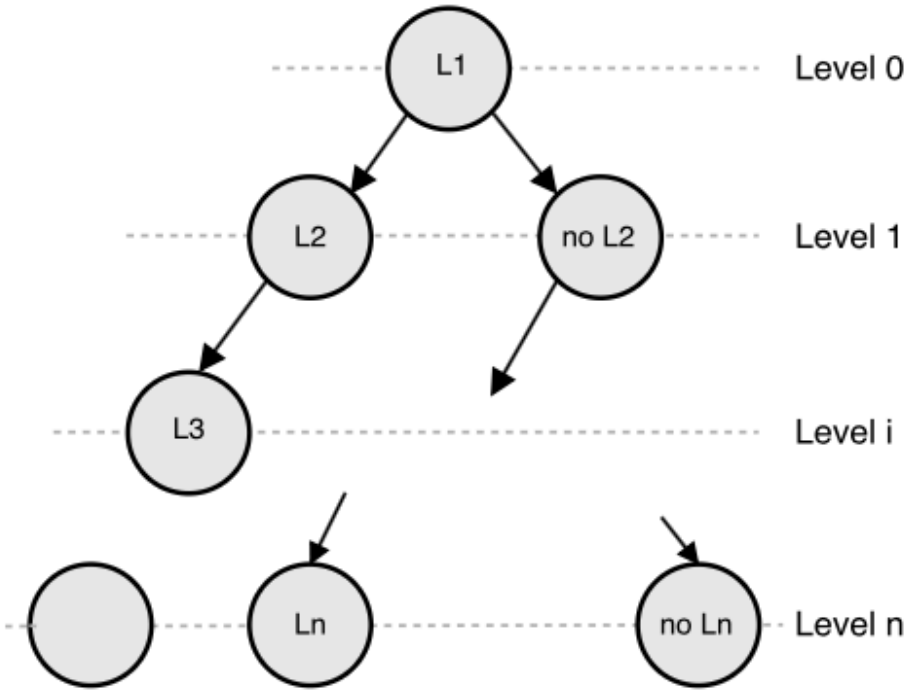


Fig. 4: Control Structure

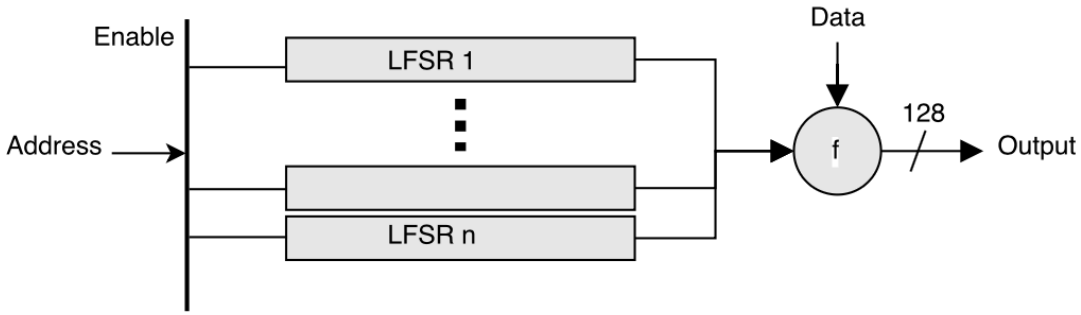


Fig. 3: General Architecture of the Hash Function

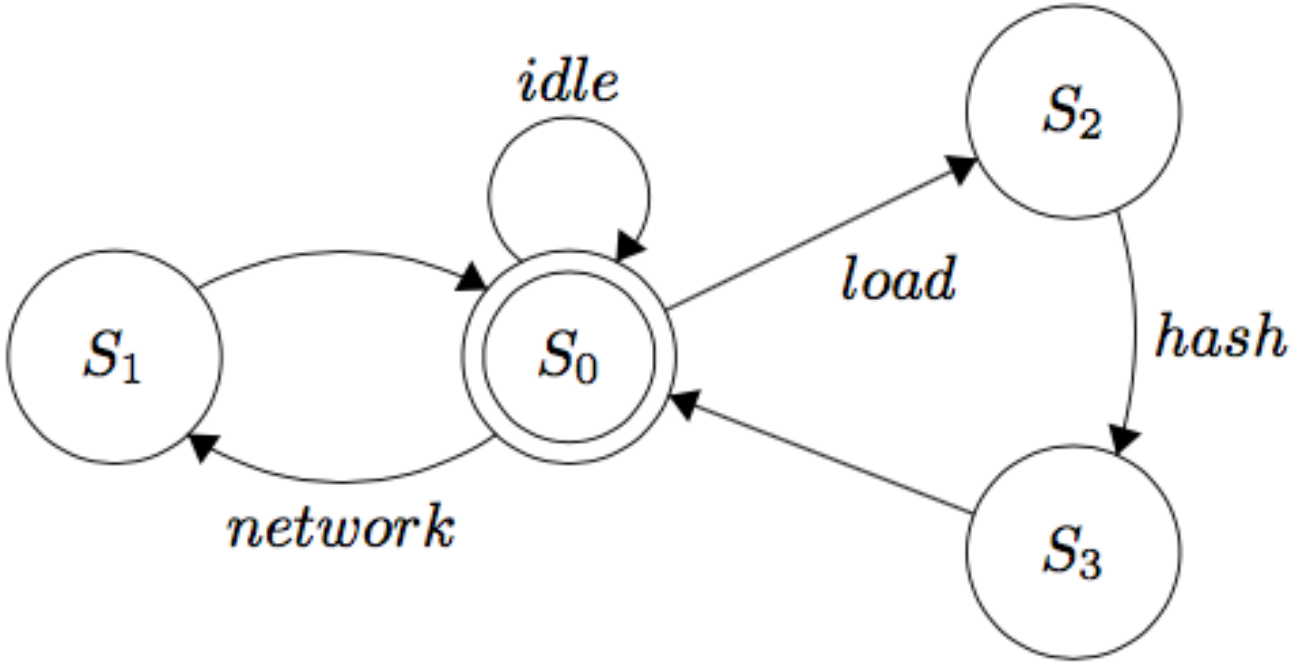
Generation of IC-programs

LFSR Generation

Algorithm V.2 Irreducible polynomial generation using Ben-Or Irreducibility Test

```
while true do
    Generate poly p(x) ∈ GF(2) of degree at most n
    for i := 1 to n/2 do
        g := gcd(p, x^(2^i) − x  mod p);
        if g ≠ 1 then
            'p is reducible'
            break
        end if
    end for
    return 'p is irreducible'
end while
```

Power Finite State Machine



Power Measurement

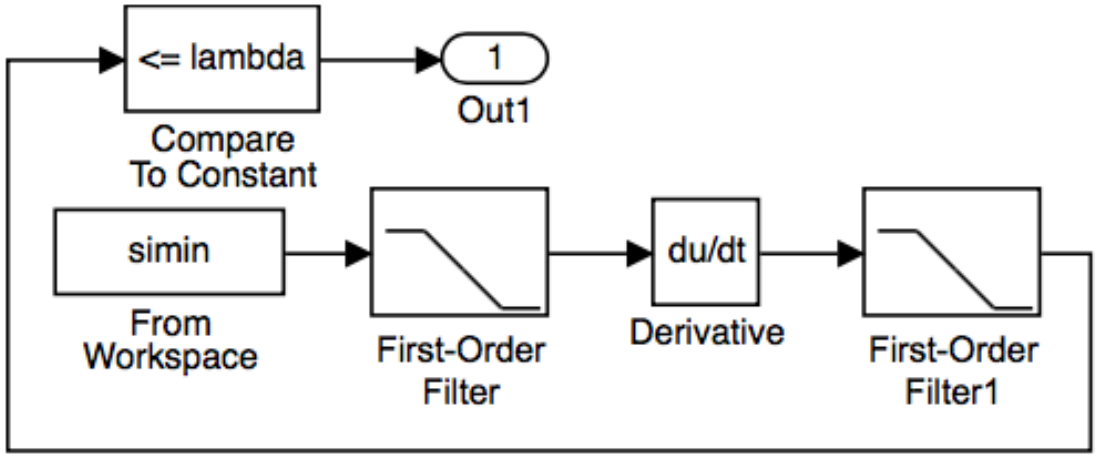


Fig. 6: Block diagram for power state extraction.

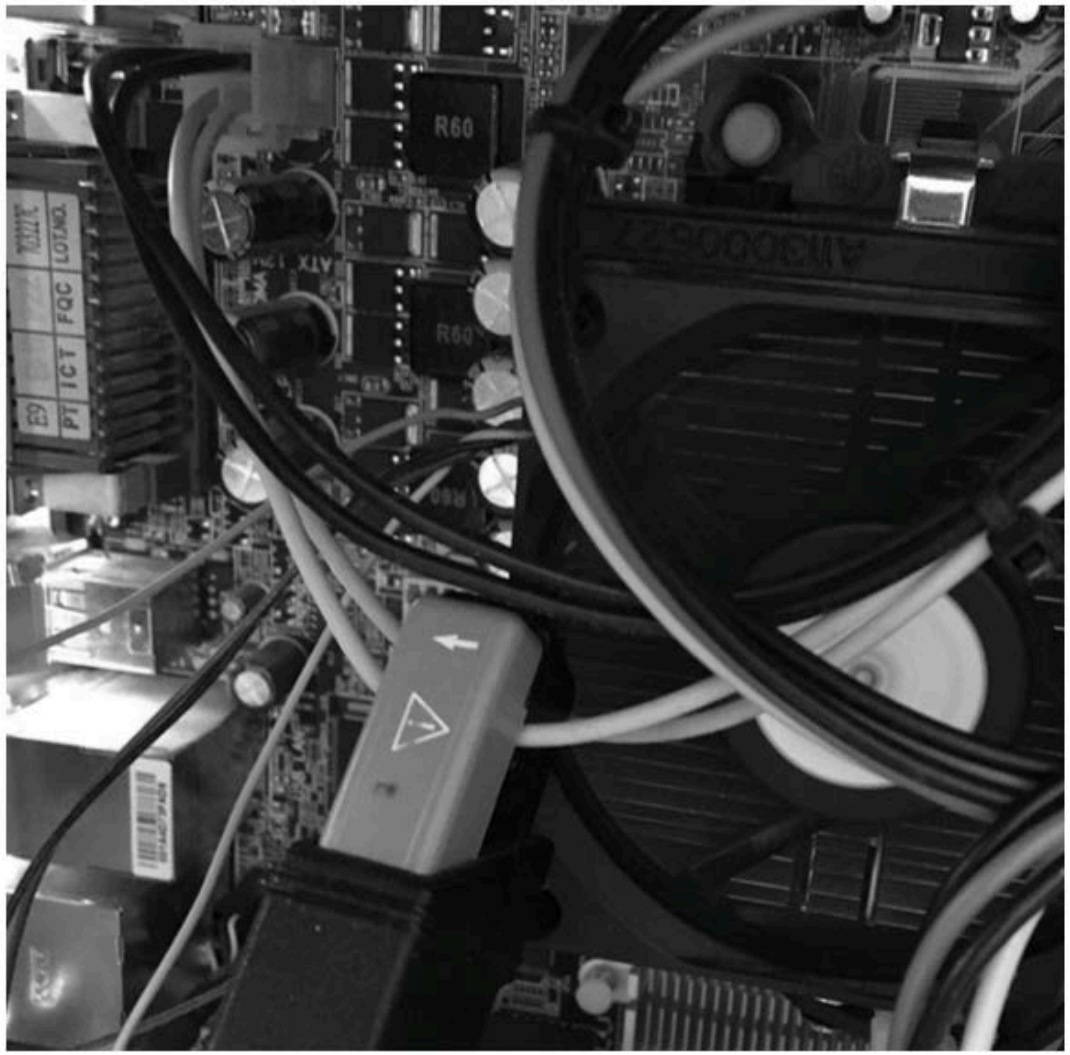
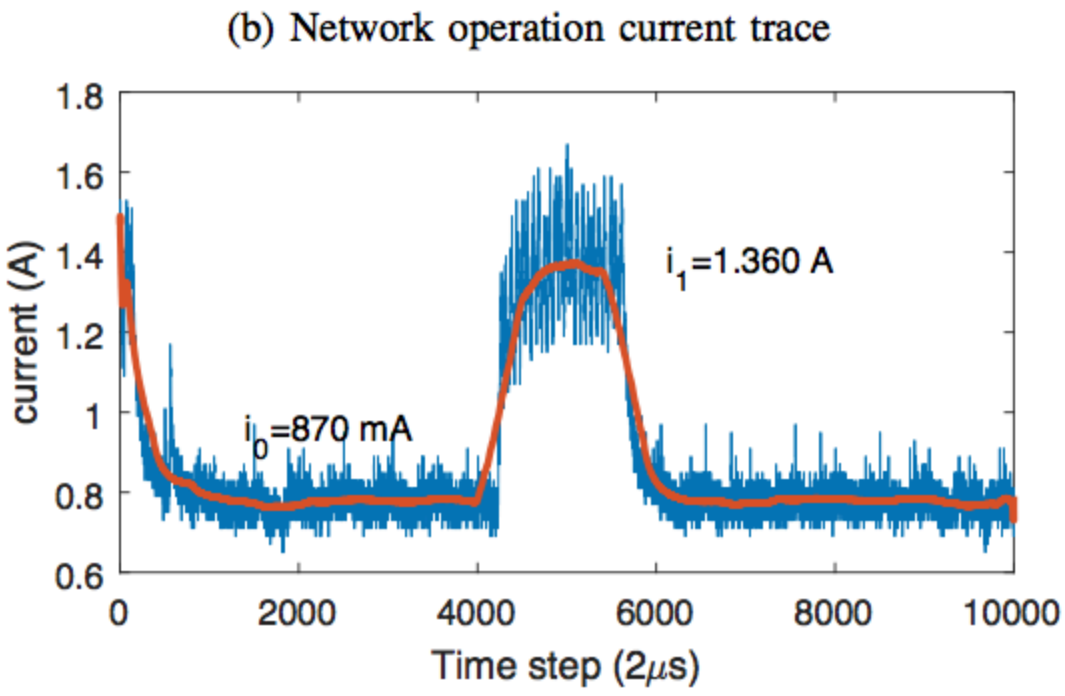
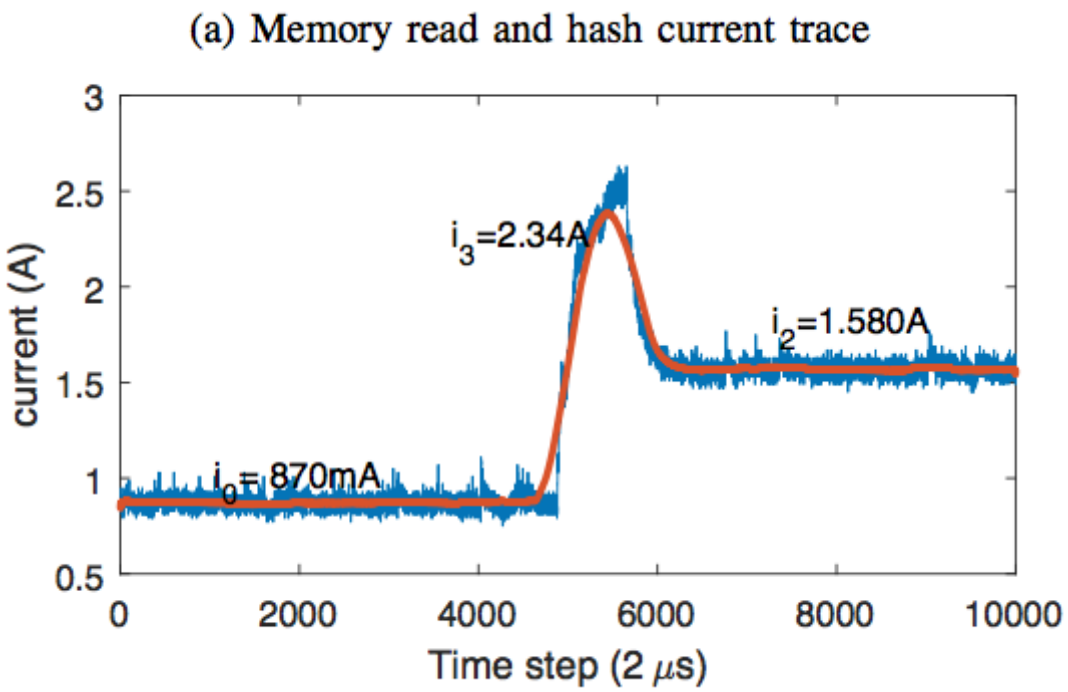


Fig. 5: The current measurement loop place around the CPU power line.





Measurements

Timing Model

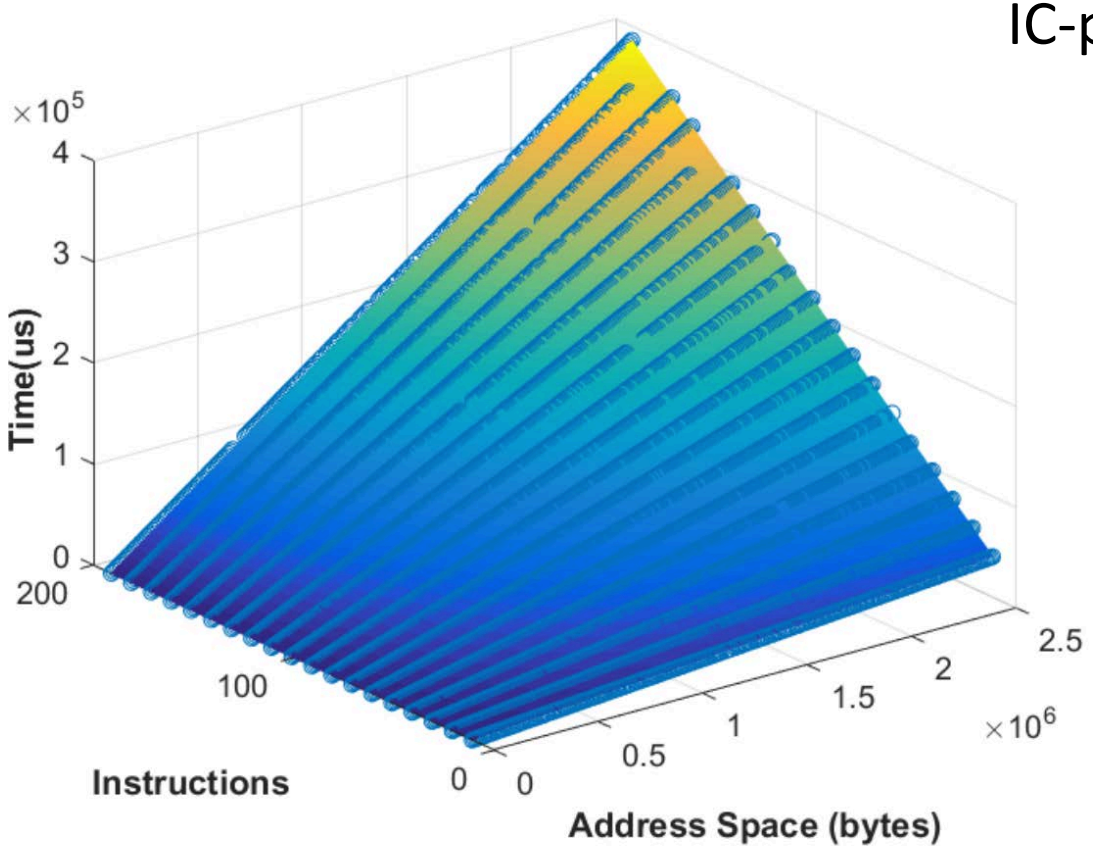


Fig. 9: Power State Timing model for hashing phase.

IC-program runtime:

$$x = [N, \ \|f\|_c]$$

$$y = 1.3958 + 0.081x(1) - 0.017x(2) + 0.008x(1) \times x(2)$$

$$\sigma = 5.4542\mu s$$

Network runtime:

$$y_n = 0.129 \times x + 12.48$$

$$\sigma_n = 1.902\mu s$$

Verification Process

- Power Language:

$$L = (S_0, S_1)^+(S_0, S_2, S_3, S_0)(S_0, S_1)$$

- Timing verification:

$$\delta \geq \gamma \max(\sigma_m + \sigma_s), \qquad \delta_n \geq \gamma \max(\sigma_n, \sigma_s)$$

Example

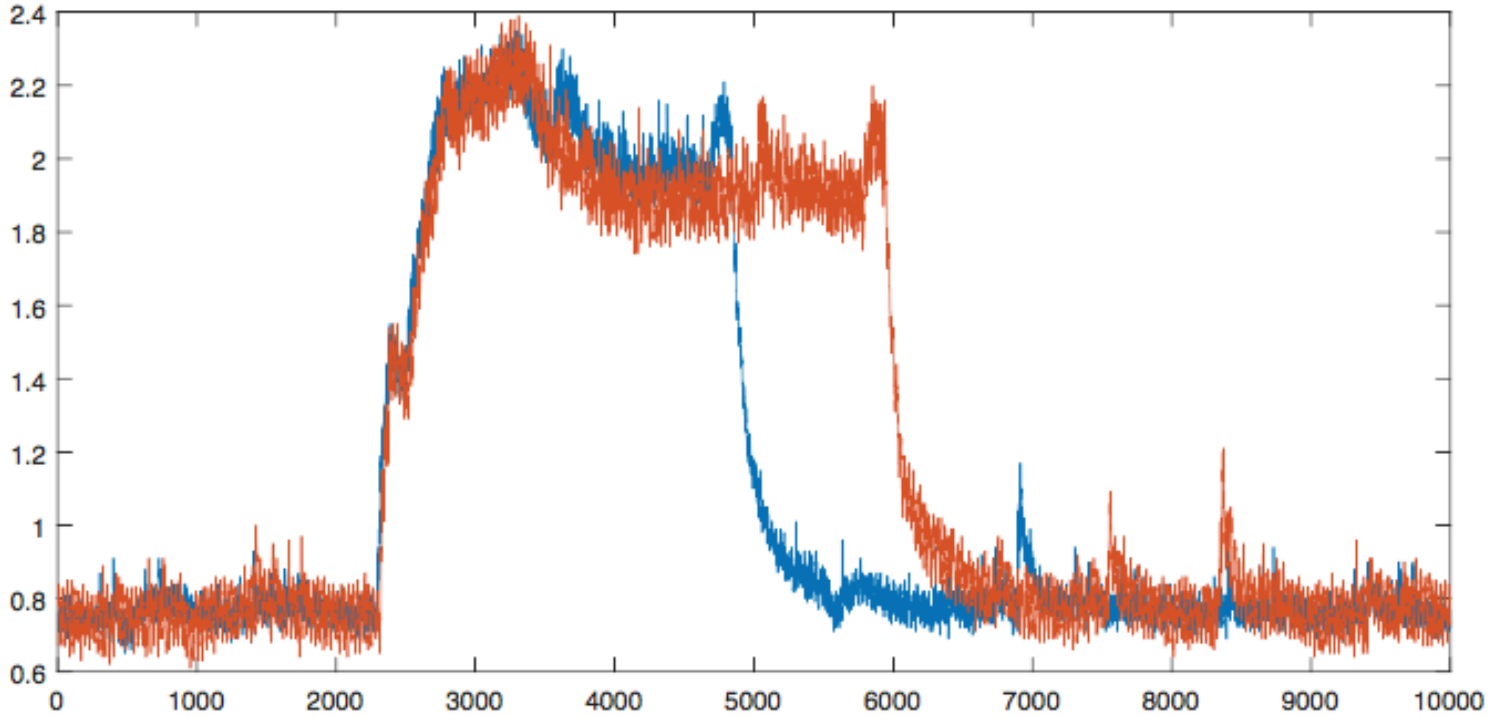


Fig. 10: Timing difference in current signal due to tampering.

Design of IC-program

$$\underset{N,\|f\|_p}{\text{minimize}} \quad y(N, \|f\|_p)$$

$$\text{subject to} \quad y(N, \|f\|_p + k) - y(N, \|f\|_p) > \gamma \cdot \max(\sigma_m, \sigma_s)$$

$$y_n(\|f\|_p) > \gamma \cdot \max(\sigma_n, \sigma_s)$$

$$\|f\|_p < cost, \;\; N/N_{total} > coverage.$$

Sampling Rate	Error Tolerance (μs)	Coverage (bytes)	Program Size
1 MHz	64.542	2,019	40
500 KHz	74.542	2,331	40
250 KHz	94.542	2,956	40
200 KHz	104.542	3,269	40
54 KHz	239.727	7,493	40

Attacker-Verifier Games

- Players:
  - Attacker
  - Defender
- Actions:
  - Attacker hide @ {ta1, ta2, tan, …}
  - Defender verify @ {td1, td2, tdn, …}
- Strategy: pick the time instances

Strategy selection

- Defender is assumed to have a renewal strategy with rate \lambda

Theorem 1. *If the verifier is playing with a renewal strategy, then the attacker's best strategy is to play periodically with a fixed period* T^*.

Detection probability

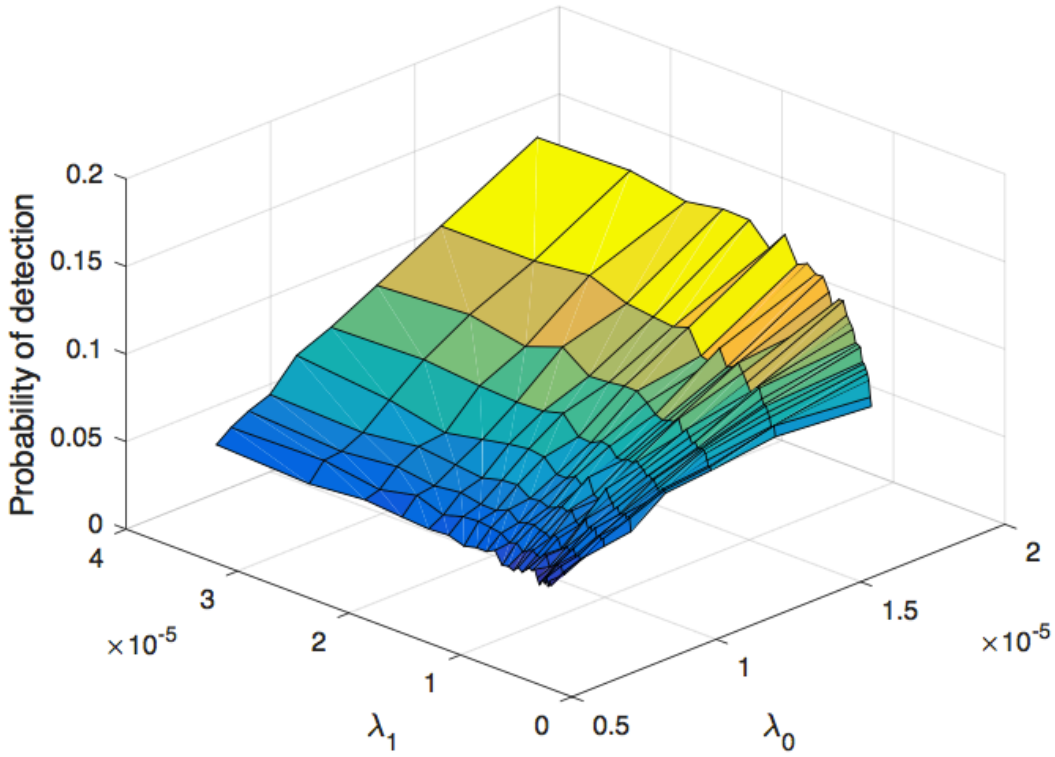


Fig. 11: The average probability of the attacker's detection as a function of the attacker's and the verifier's play rates.

Attacker Utility

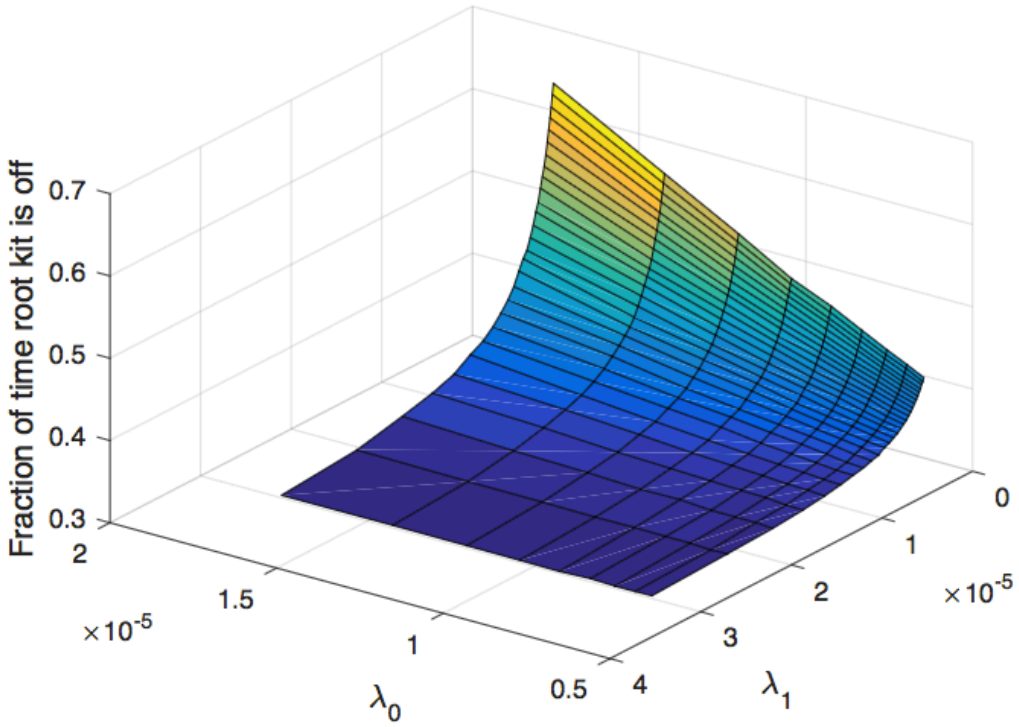


Fig. 12: The average fraction of the time the attacker's malicious activity is hidden as a function of the attacker's and the verifier's play rates.

Evasion Rate

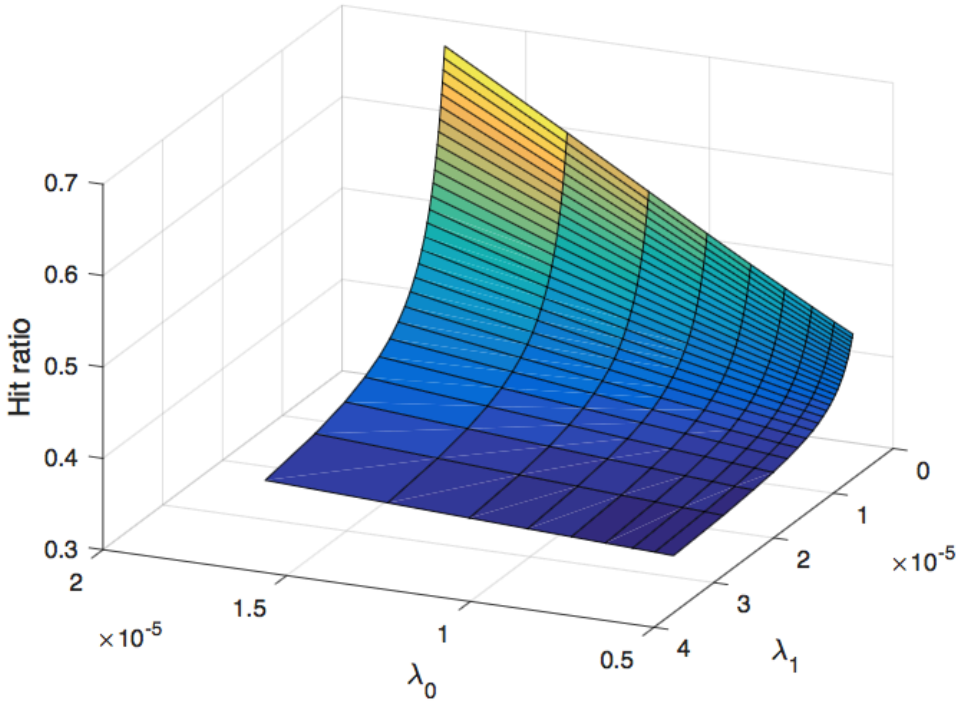


Fig. 13: The average ratio of attestation tasks that were evaded by the attacker's actions as a function of the attacker's and the verifier's play rates.

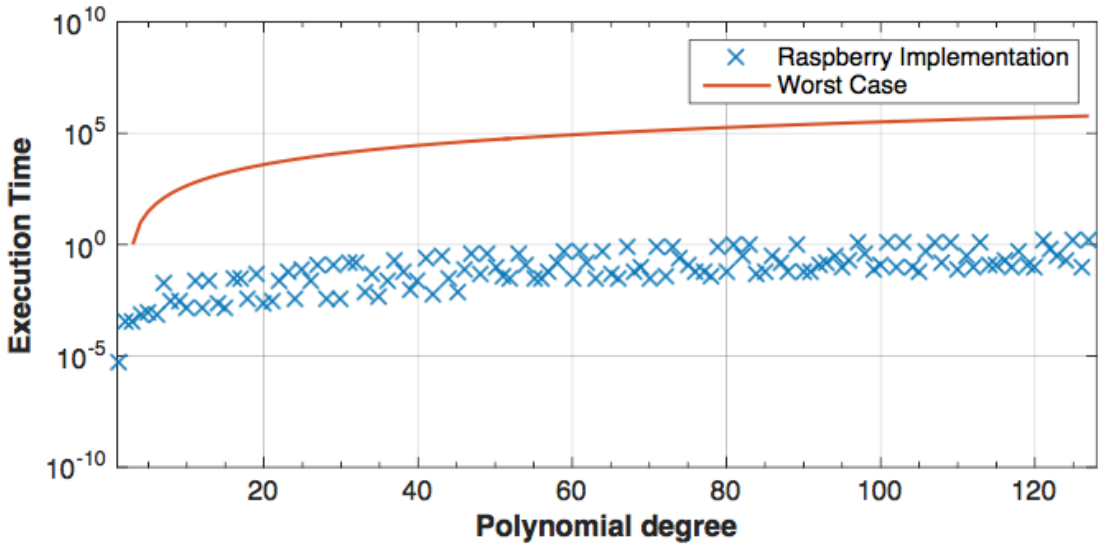


Fig. 14: Average time in seconds for generating a random irreducible polynomial for degree d

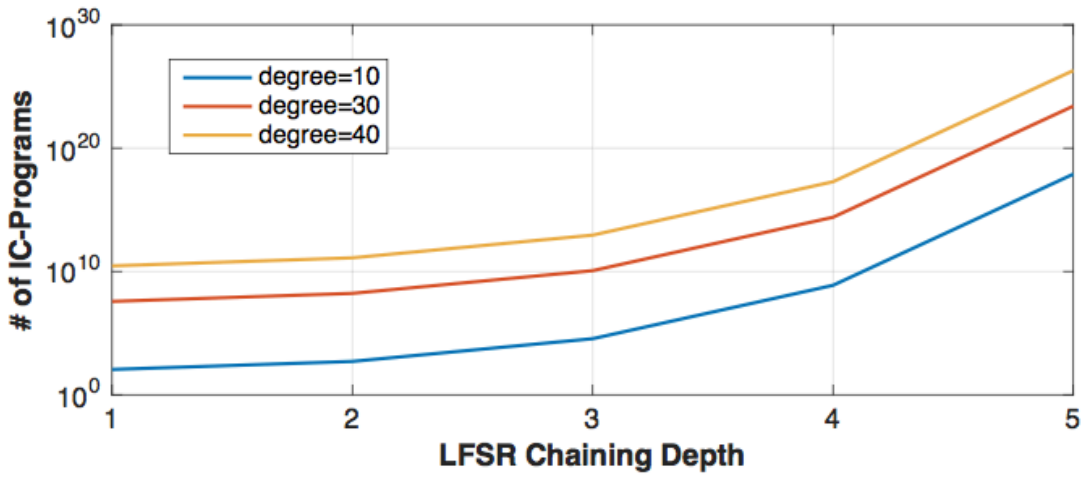


Fig. 15: Maximum Number of IC-Programs a varying tree depth and polynomial degree.

Performance

Security Analysis

Proxy Attack	Data Pointer Redirection	Static Analysis	Active Analysis	Attacker Hiding	Force Retraining
Use a remote machine to compute	Keep a clean copy of memory	Attempt to precompute	Learn how the program works	Hide when the checker runs	Learn a tampered model
Network timing will detect redirection	**IC-program optimized to detect k=4**	**The structure is flattened thus NP-Hard**	• **Change of the IC-program prevents adaptation.** • **Space of programs big.**	**Defender strategy leads to attacker slow down**	**Retraining is only done with a clean HDD**

Conclusion

- We use power measurements as a trust base
- The IC-program diversity prevents the attacker from adapting
- The defender's strategy results in the attacker detection