IDA: Hybrid Attestation with Support for Interrupts and TOCTOU

Fatemeh Arkannezhad
Department of Electrical and Computer Engineering
University of California, Los Angeles

31st NDSS Symposium
February 29, 2024
San Diego, California, USA

The Team

Fatemeh Arkannezhad
UCLA

Justin Feng
UCLA

Nader Sehatbakhsh
UCLA

UCLA Samueli School of Engineering

Outline

Introduction

Motivation

Our Work

System Architecture

Implementation

Evaluation

Conclusions

Outline

Introduction

Motivation

Our Work

System Architecture

Implementation

Evaluation

Conclusions

Embedded devices

Introduction: Embedded Devices

Wide range of applications and interactions

Vulnerable to attacks

Ensuring integrity and security

Introduction: Embedded Devices

Wide range of applications and interactions

Vulnerable to attacks

Ensuring integrity and security

Low-end devices constraints

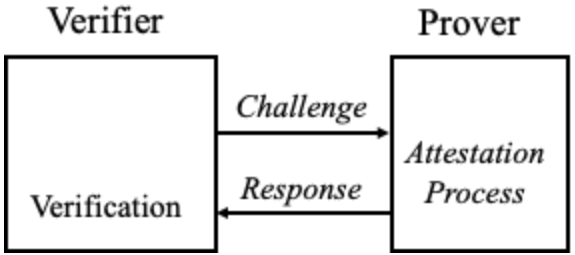
- Hardware
- Power

Outline

Motivation: Embedded Devices

Remote Attestation

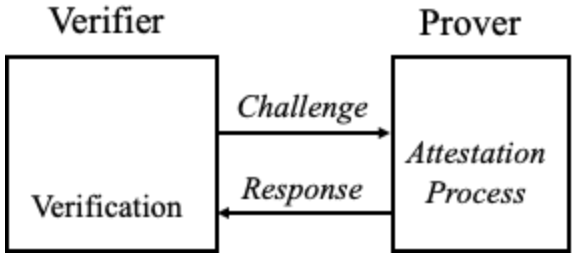
High-end -> Hardware (TEE)



Motivation: Embedded Devices

Remote Attestation

- High-end -> Hardware (TEE)
- Low-end
  - Examples
    - Software: ACES, RealSWATT, …
    - Hybrid: GAROTA, VRASED, …



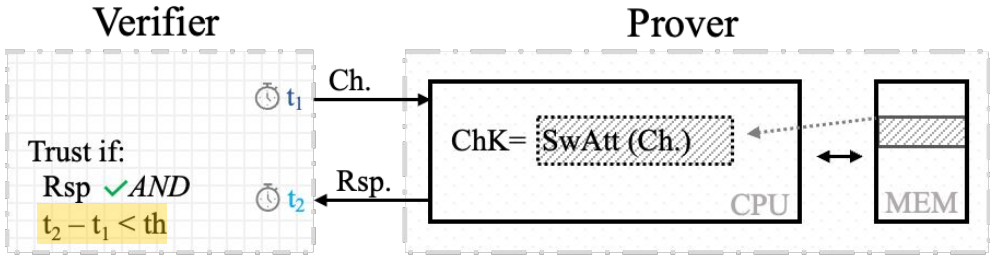
Motivation: Embedded Devices

Software: ✖ Strict assumptions

Request-to-Response time =>

✖ Low-latency attacks

✖ Network limitations



Motivation: Embedded Devices

Software: ✖ Strict assumptions

Request-to-Response time =>

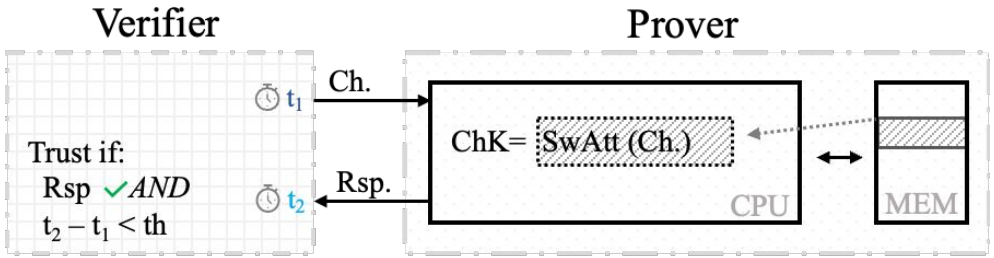
✖ Low-latency attacks

✖ Network limitations

✔ Overhead

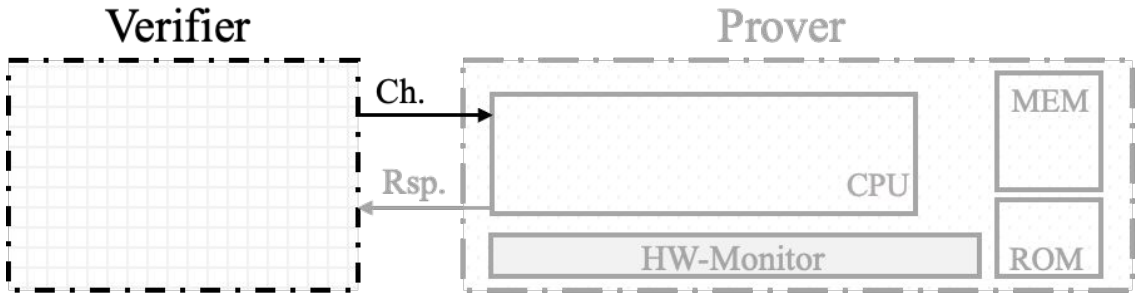
✔ Flexibility

✖ Robust



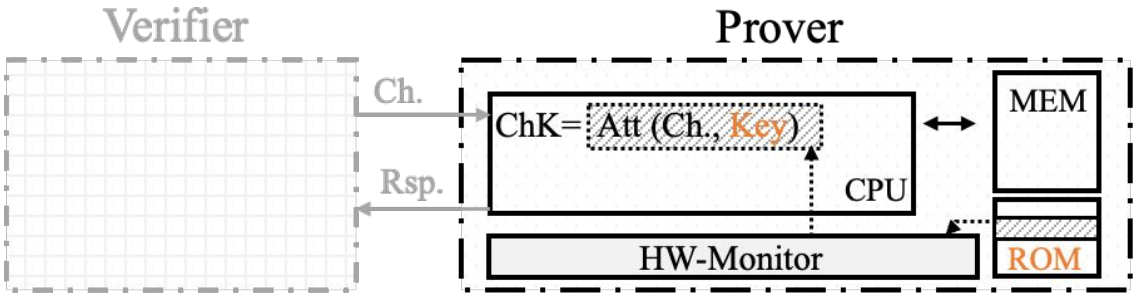
Motivation: Embedded Devices

Hybrid:



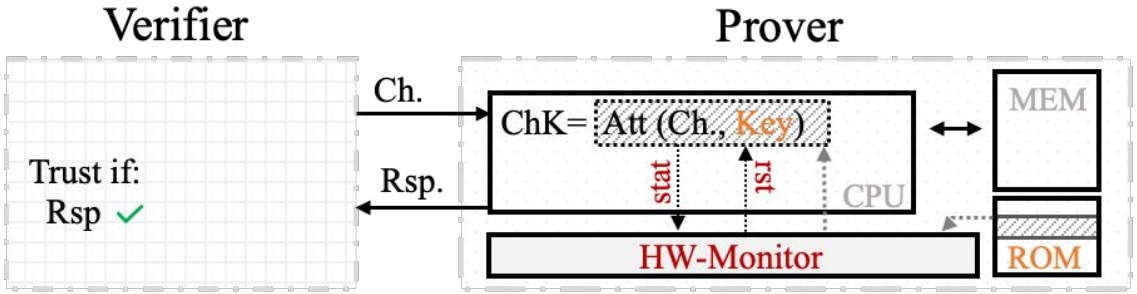
Motivation: Embedded Devices

Hybrid:



Motivation: Embedded Devices

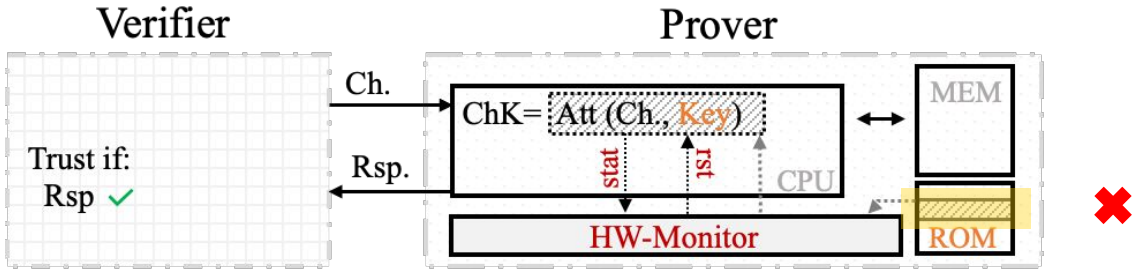
Hybrid:



Motivation: Embedded Devices

Hybrid:

- ✖ Secure key storage and access control =>



Motivation: Embedded Devices

Hybrid:

- ✖ Secure key storage and access control =>

  Taint, track, and sanitize key-related operations

Motivation: Embedded Devices

Hybrid:

- ✖ Secure key storage and access control =>
  - Taint, track, and sanitize key-related operations
  - ✖ No support
    - Interrupts
    - DMA requests

Motivation: Embedded Devices

Hybrid:

✔Robust

Motivation: Embedded Devices

Hybrid:

- ✔ Robust
- ✖ Protect code and cryptographic keys
  - ✖ No interrupt
  - ✖ No DMA requests

Motivation: Embedded Devices

Hybrid:

- ✔ Robust
- ✖ Protect code and cryptographic keys
  - ✖ No interrupt
  - ✖ No DMA requests

✖ TOCTOU

Introduction: TOCTOU

Static Remote Attestation

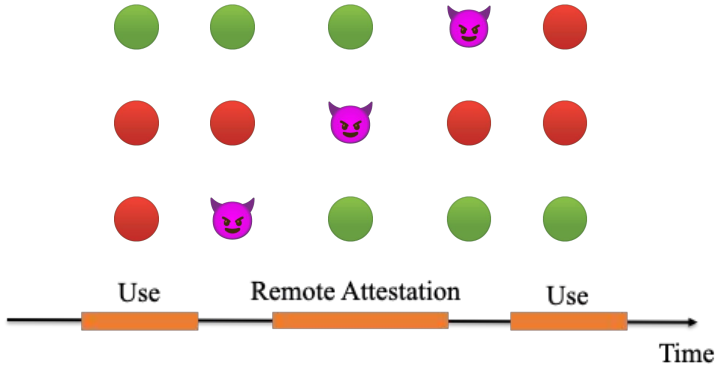
- ✖ Time-Of-Check-Time-Of-Use attack

No information about prover before or after consecutive RAs

Introduction: TOCTOU

Static Remote Attestation

- ✖ Time-Of-Check-Time-Of-Use attack
- 😈 Adversary:
  - Modifies memory after RA
  - Hides the malware during RA
  - Erase itself before RA

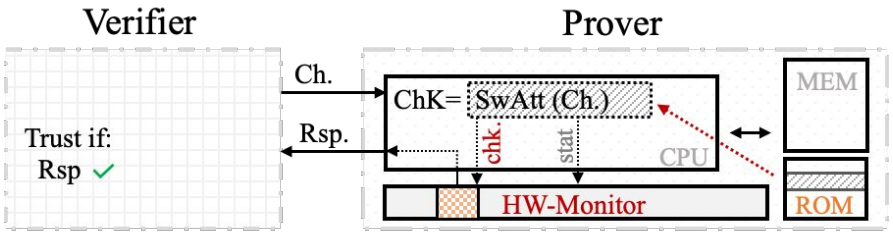


Outline

Introduction: IDA

Hybrid Remote Attestation

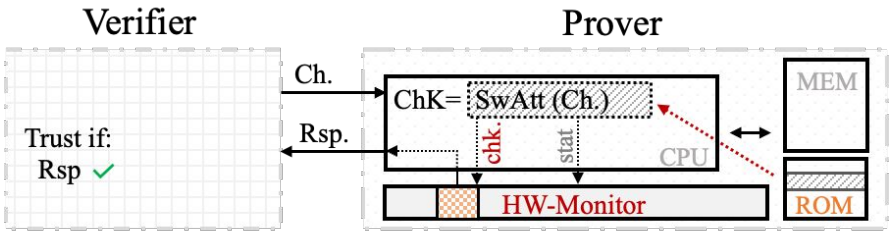
- Flexibility (SW-based)
- Security (Hybrid-based)



Introduction: IDA

Hybrid Remote Attestation

- Flexibility (SW-based)
- Security (Hybrid-based)

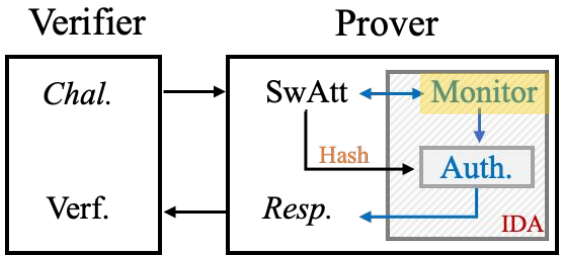


No secure access control for the key

- ✔Interrupts
- ✔DMA requests

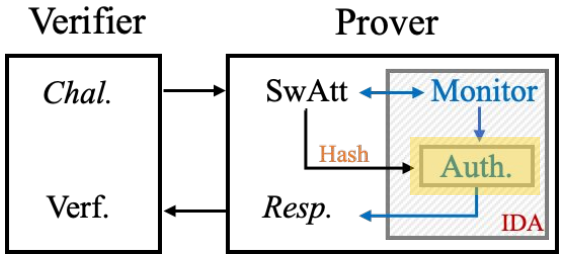
Introduction: IDA

1) Hardware-monitored hash computation
   - No key during attestation
   - Correct execution of the software
   - Secure hash computation => Interrupts



Introduction: IDA

1) Hardware-monitored hash computation
   - No key during attestation
   - Correct execution of the software
   - Secure hash computation => Interrupts



2) Authenticate response (HMAC) => Authenticity, Correctness

Attack Model

Collision Attack

- Hash computation security
- Unique nonces
- Authentication forgery resistance

Attack Model

Collision Attack
- Hash computation security
- Unique nonces
- Authentication forgery resistance

Substitution Attack
- ROM residency of SW-Att
- Correct implementation of hardware
- Authentication forgery resistance

Motivation: Improve IDA

Resets process upon completion

Reducing reset overhead

Time-of-Check-to-Time-of-Use (TOCTOU)

Introduction: IDA+

✔**Handling interrupts efficiently:**

Tracks changes to program memory during interrupt handling

Avoid resetting

Critical values in Sw-Att are saved before servicing an interrupt

Introduction: IDA+

✔**Handling interrupts efficiently:**

Tracks changes to program memory during interrupt handling

Avoid resetting

Critical values in Sw-Att are saved before servicing an interrupt

✔**TOCTOU problem mitigation:**

Hardware Moule tracks changes to program memory

Outline

Introduction

Motivation

Our Work

System Architecture

- Properties
- Modules
  - Hardware Module
  - Software Attestation
- IDA+

Implementation

Evaluation

Conclusions

Properties

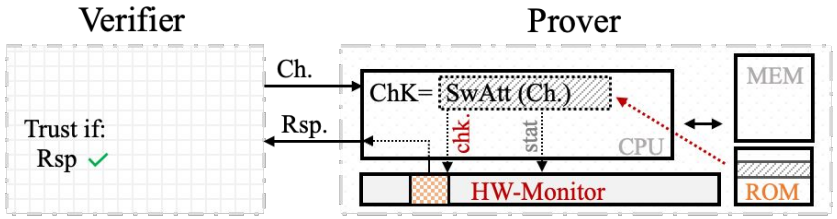
Relaxed

Atomic execution

Invocation flexibility

Flexible reset

DMA monitoring



Properties

Relaxed

Atomic execution

Invocation flexibility

Flexible reset

DMA monitoring

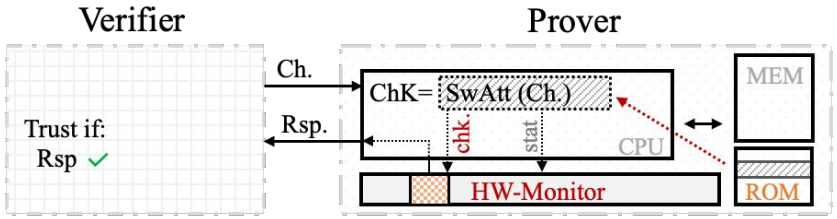
Security

Immutability

Functional correctness

Interrupt / DMA -> Reset

Implementation correctness



System Architecture: IDA

Hardware Module (HWM)

- Monitoring the CPU
- Ensuring security during attestation

Memory System (ROM, program memory, data memory)

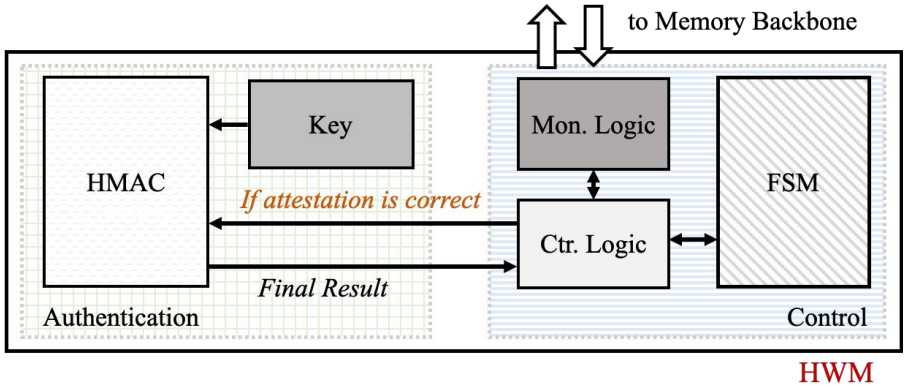
CPU Unit

Memory Backbone

System Architecture: IDA

Hardware Module (HWM)

Secure execution of Sw-Att



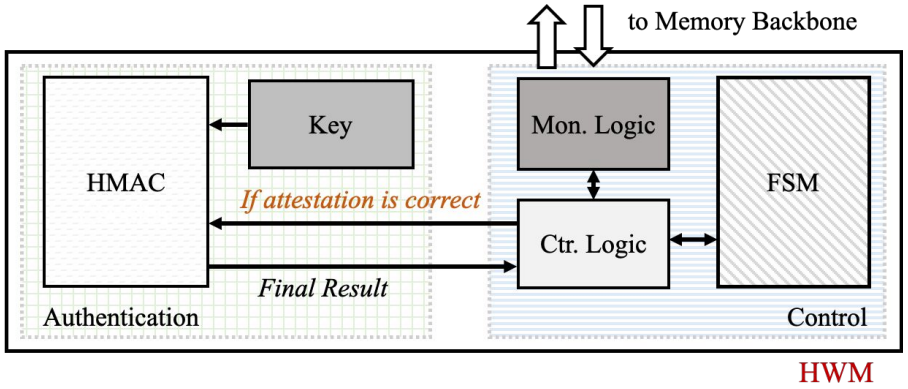
System Architecture: IDA

Hardware Module (HWM)

Secure execution of Sw-Att

Monitor

- PC
- Memory Address
- Interrupt Request
- DMA Write
- Shared Status Register



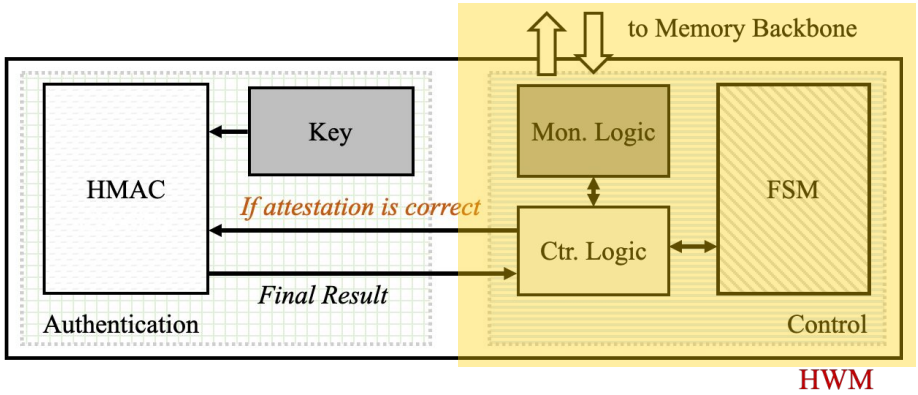
System Architecture: IDA

Controller Unit

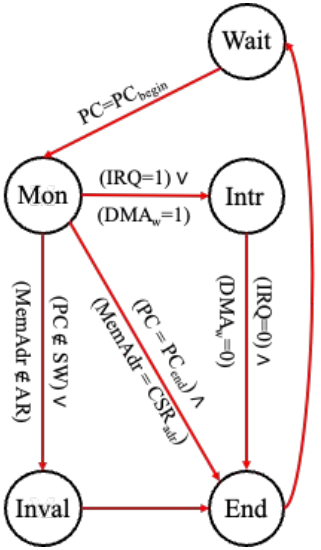
Finite State Machine

Monitor

- PC
- Memory Address
- DMA Write
- Interrupt Request



System Architecture: IDA



Monitor PC and memory address

Interrupt

Invalid

End as hash computation is completed without interruptions

System Architecture: IDA

Receiving Challenge

Initializes hash

- Nonce
- Attestation range (optional)

Algorithm 1: IDA: Algorithm and SW-Att function

```
1  input: Ch. = {nonce, AR*}, Repeat;
   /* AR: address range to be attested.
      */
2  while Repeat do
3  |  Resp. = Sw-Att(Ch.);
4  end
5  return Resp.;
1  Function SW-Att
2  |  input: nonce, AR, inter;
3  |  CSR = CSR_adr;
   |  /* Address for accessing HWM.    */
4  |  cHash = nonce;
5  |  i = AR_begin;
6  |  while (i < AR_end) do
7  |  |  cHash = hash(Mem[i], cHash);
8  |  |  i++;
9  |  |  if (inter == 1) then break;
10 |  end
11 |  Mem[CSR] = cHash;
12 |  IdleLoop();
   |  /* To allow time for HWM to
   |     calculate the response.       */
13 |  cHash = Mem[CSR];
14 |  return cHash;
```

System Architecture: IDA

Executing Sw-Att

Compute hash over program memory

Algorithm 1: IDA: Algorithm and SW-Att function

```
1  input: Ch. = {nonce, AR*}, Repeat;
   /* AR: address range to be attested.
      */
2  while Repeat do
3  |  Resp. = Sw-Att(Ch.);
4  end
5  return Resp.;
1  Function SW-Att
2  |  input: nonce, AR, inter;
3  |  CSR = CSR_adr;
   |  /* Address for accessing HWM.    */
4  |  cHash = nonce;
5  |  i = AR_begin;
6  |  while (i < AR_end) do
7  |  |  cHash = hash(Mem[i], cHash);
8  |  |  i++;
9  |  |  if (inter == 1) then break;
10 |  end
11 |  Mem[CSR] = cHash;
12 |  IdleLoop();
   |  /* To allow time for HWM to
   |     calculate the response.       */
13 |  cHash = Mem[CSR];
14 |  return cHash;
```

System Architecture: IDA

Executing Sw-Att

Compute hash over program memory

Interrupts and DMA requests

Halt and restart hash computation

Algorithm 1: IDA: Algorithm and SW-Att function

```
1  input: Ch. = {nonce, AR*}, Repeat;
   /* AR: address range to be attested.
      */
2  while Repeat do
3  |  Resp. = Sw-Att(Ch.);
4  end
5  return Resp.;
1  Function SW-Att
2  |  input: nonce, AR, inter;
3  |  CSR = CSR_adr;
   |  /* Address for accessing HWM.    */
4  |  cHash = nonce;
5  |  i = AR_begin;
6  |  while (i < AR_end) do
7  |  |  cHash = hash(Mem[i], cHash);
8  |  |  i++;
9  |  |  if (inter == 1) then break;
10 |  end
11 |  Mem[CSR] = cHash;
12 |  IdleLoop();
   |  /* To allow time for HWM to
   |     calculate the response.       */
13 |  cHash = Mem[CSR];
14 |  return cHash;
```

System Architecture: IDA

Controlling Attestation with HWM

HWM monitors process with FSM
Receives input from HWM (interrupts)

HWM

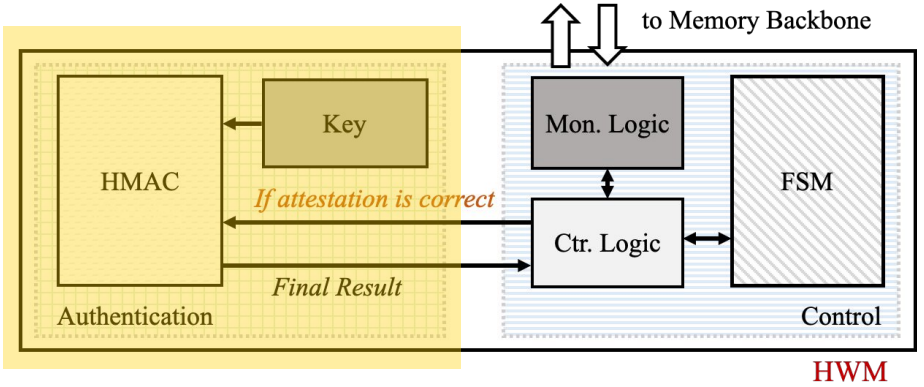
Algorithm 1: IDA: Algorithm and SW-Att function

```
1  input: Ch. = {nonce, AR*}, Repeat;
   /* AR: address range to be attested.
      */
2  while Repeat do
3  |  Resp. = Sw-Att(Ch.);
4  end
5  return Resp.;
1  Function SW-Att
2  |  input: nonce, AR, inter;
3  |  CSR = CSR_adr;
   |  /* Address for accessing HWM.    */
4  |  cHash = nonce;
5  |  i = AR_begin;
6  |  while (i < AR_end) do
7  |  |  cHash = hash(Mem[i], cHash);
8  |  |  i++;
9  |  |  if (inter == 1) then break;
10 |  end
11 |  Mem[CSR] = cHash;
12 |  IdleLoop();
   |  /* To allow time for HWM to
   |     calculate the response.       */
13 |  cHash = Mem[CSR];
14 |  return cHash;
```

System Architecture: IDA

Authentication Unit

- Validates integrity and authenticity
  - HMAC-SHA-256



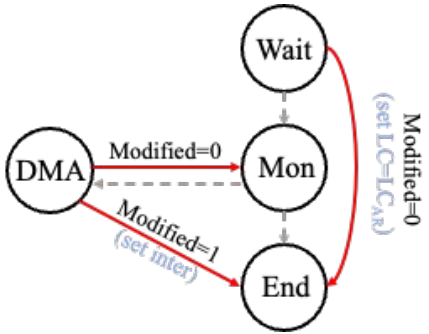
System Architecture: IDA+

TOCTOU

After last RA

Program is NOT modified

HWM sends Response (HMAC of Nonce)

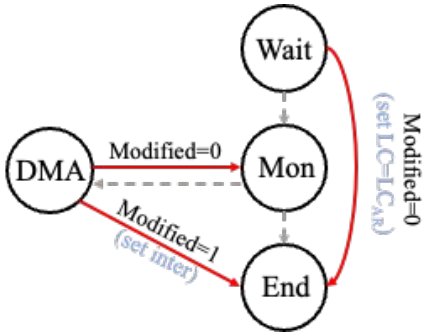


System Architecture: IDA+

TOCTOU

After last RA

- Program is NOT modified
  - HWM sends Response (HMAC of Nonce)
- O.W.
  - Attestation
  - HWM sends NULL (random)



System Architecture: IDA+

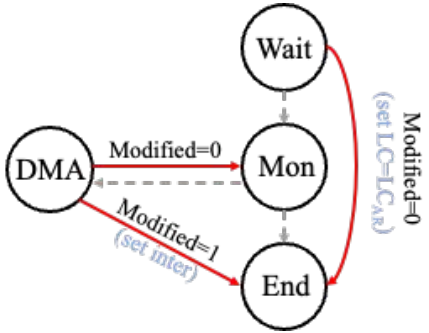
Interrupts

During interrupt handling

Monitor memory (writes)

NOT Modified

Continue

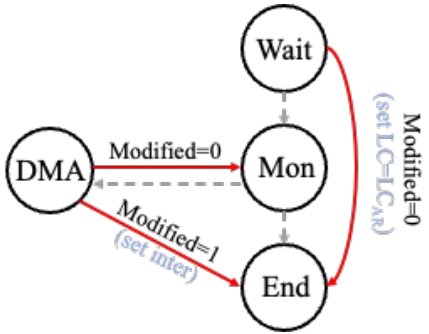


System Architecture: IDA+

Interrupts

During interrupt handling

- Monitor memory (writes)
  - NOT Modified
    - Continue
  - O.W.
    - Reset



Outline

Introduction

Motivation

Our Work

System Architecture

Implementation

Evaluation

Conclusions

Implementation

GitHub: https://github.com/ssysarch/IDA

OpenMSP430

ZYNQ FPGA

Implementation

GitHub: https://github.com/ssysarch/IDA

OpenMSP430

ZYNQ FPGA

- SW-Att: SHA3-256 from the HACL* library
- HWM: HMAC-SHA-256 implemented by Secworks

Outline

Introduction

Motivation

Our Work

System Architecture

Implementation

Evaluation — Overhead

Conclusions — Advantages

Evaluation: Overhead

3% Memory

MSP430 4 KB of ROM

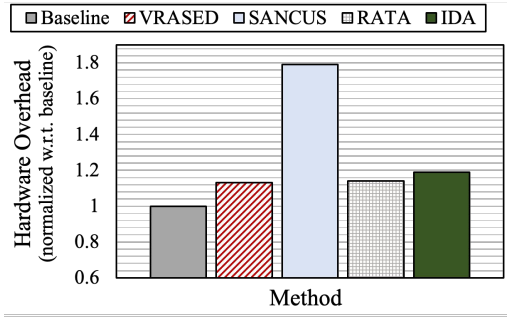
Evaluation: Overhead

19% Hardware

HWM Area:

54.4% Authentication

45.6% Controller

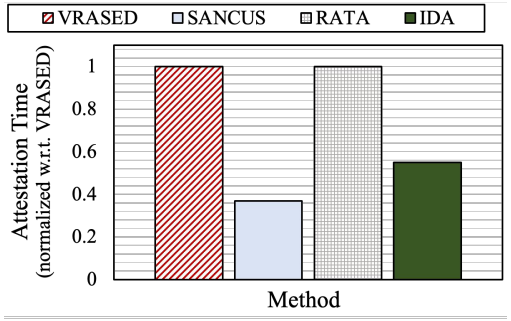


Evaluation: Overhead

≈ 2x Attestation Time

8 KB program memory (8 MHz Clock)

HMAC vs hash verification for each access



Fatemeh Arkannezhad <fatemeharkan@ucla.edu>
Secure Systems and Architectures Lab <ssysarch.ee.ucla.edu>

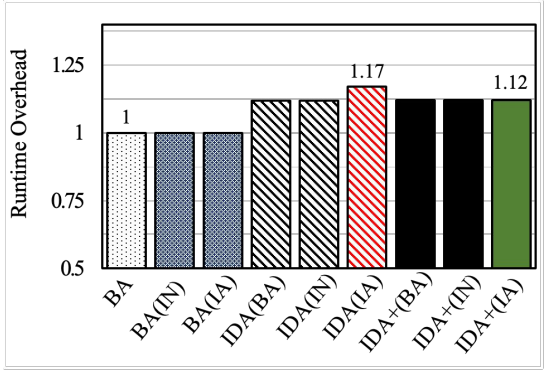
Evaluation: Overhead

12% Runtime

Benchmark: MiBench

FFT, Basicmath, dijkstra

B: baseline I: Interrupted
N: Normal A: Aggressive

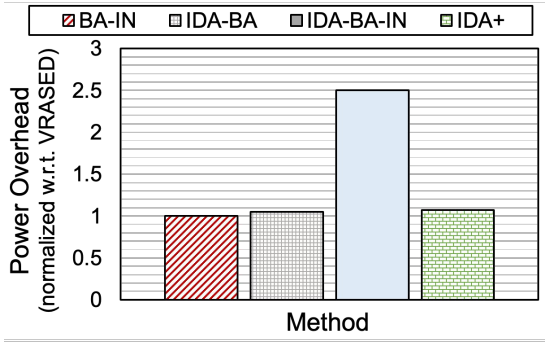


Evaluation: Overhead

2.5% Energy

Worst case:

Interrupt intervals ≈ Attestation time



Evaluation: IDA / IDA+ Advantages

✔TOCTOU

✔Minimal storage overhead

✔Immunity to vulnerabilities related to storing the key

✔Support for interrupts and DMA requests

Outline

Introduction

Motivation

Our Work

System Architecture

Implementation

Evaluation

Conclusions

Summary

RA for low-end devices

Flexible:

- No key storage
  - Interrupts
  - DMA requests

Secure:

- Hardware protection
- TOCTOU attack

Summary

RA for low-end devices

Flexible:

- No key storage
  - Interrupts
  - DMA requests

Secure:

- Hardware protection
- TOCTOU attack

IDA / IDA+

Hardware-monitored hash computation

Authenticate response (HMAC)

2x faster attestation time

19% hardware overhead

Contact

UCLA Secure Systems and Architectures Lab (SsysArch)

https://ssysarch.ee.ucla.edu/

Fatemeh: fatemeharkan@ucla.edu

Justin: jfeng10@ucla.edu

Nader: nsehat@ucla.edu

Thank you, Questions?

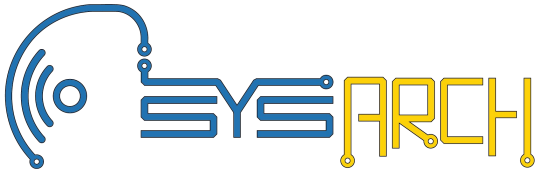
IDA / IDA+: hybrid RA for low-end devices

Flexible:

- No key storage
- Interrupts
- DMA requests

Secure:

- Hardware monitoring
- TOCTOU attack



Contact:

fatemeharkan@ucla.edu