



Secure Collaboration and Attestation in Real-Time Autonomous Robots

Nader Sehatbakhsh

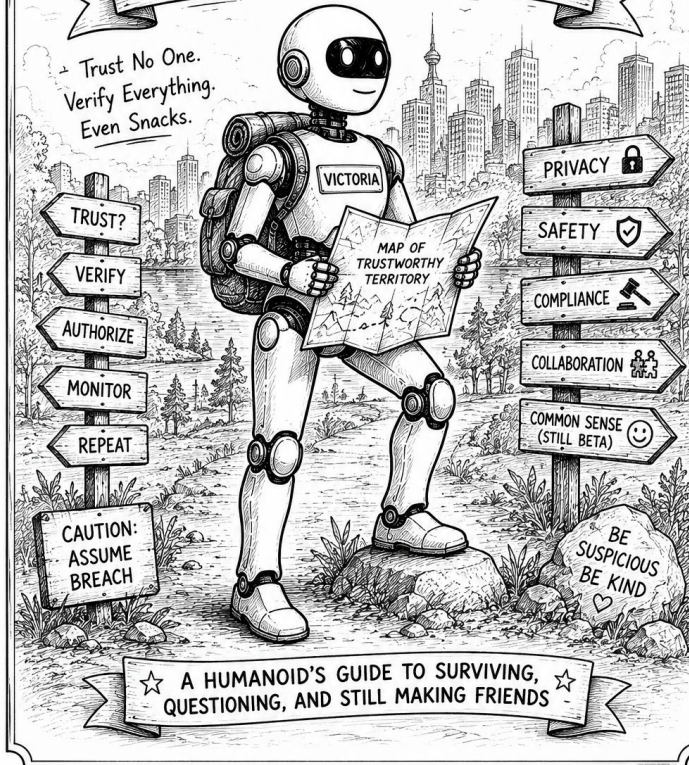
Department of Electrical and Computer Engineering

University of California, Los Angeles

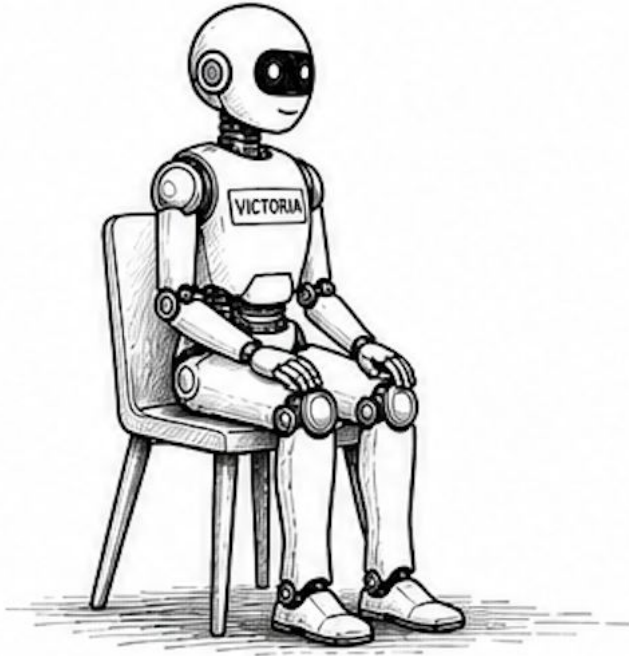
MPSoC 2026

THE ADVENTURES OF **VICTORIA** IN A **ZERO-TRUST WORLD**

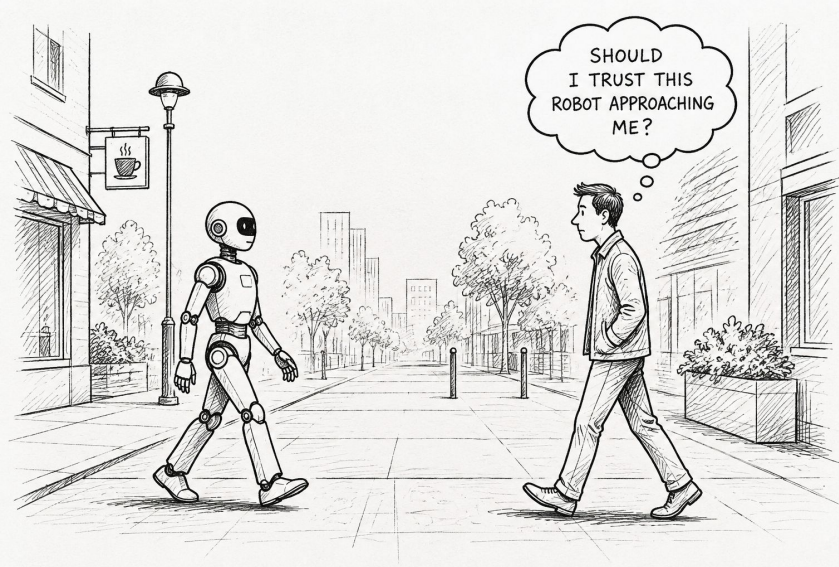
Trust No One.
Verify Everything.
Even Snacks.



Emerging Trust Challenges



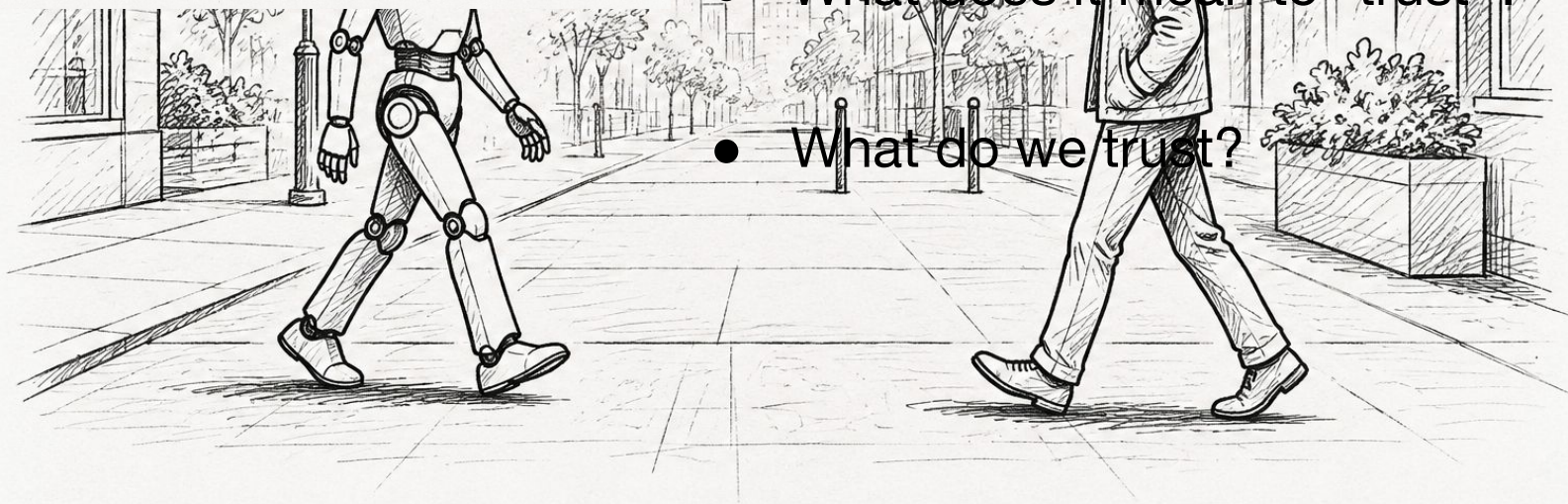
- Increasing integration of autonomous systems in real-world.
- Raises concerns about:
 - Physical safety
 - Privacy
 - Secure collaboration and interactions
 - Availability vs. security

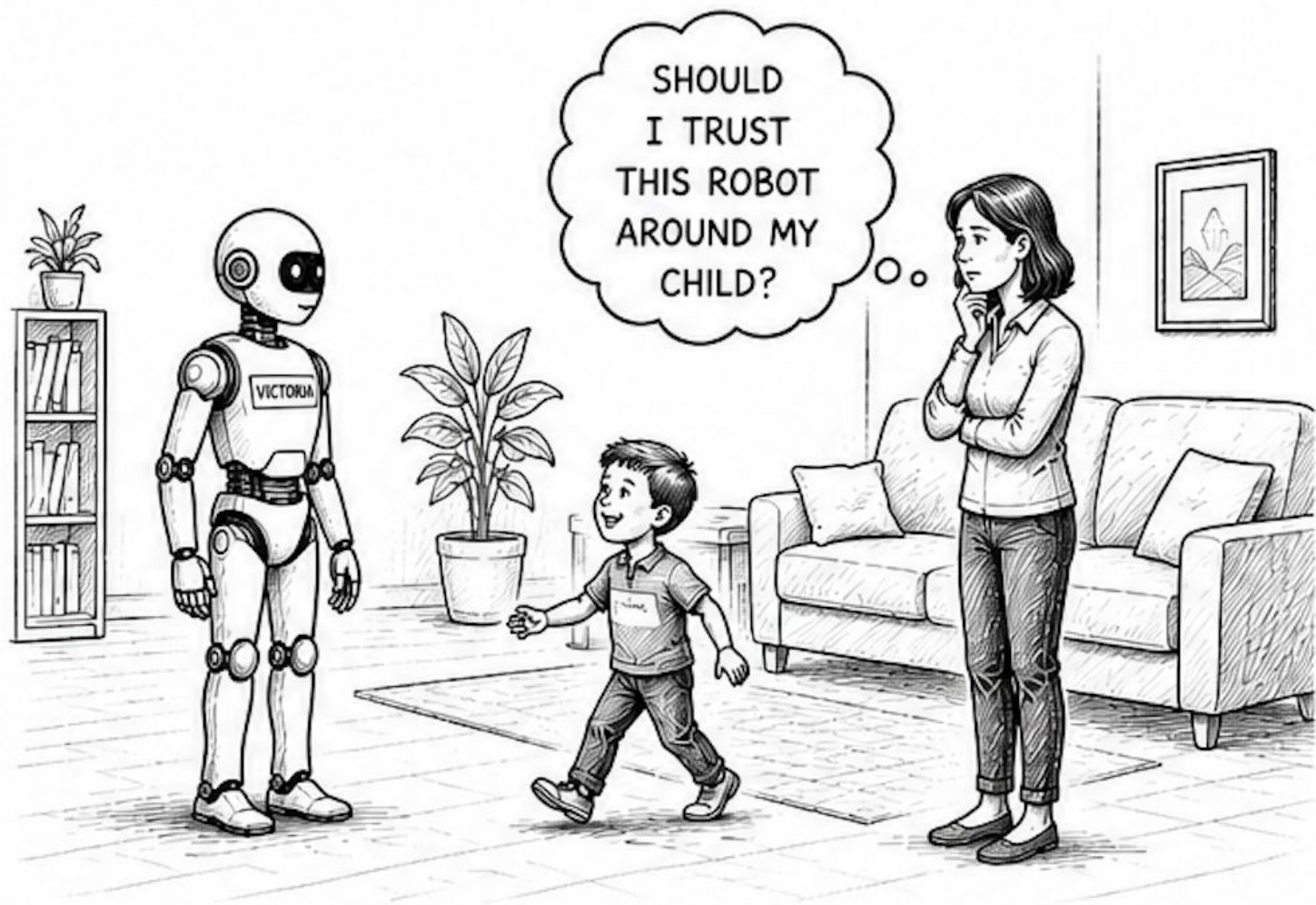


- How to establish trust between humans-robots and robot-robots?

- What does it mean to “trust”?

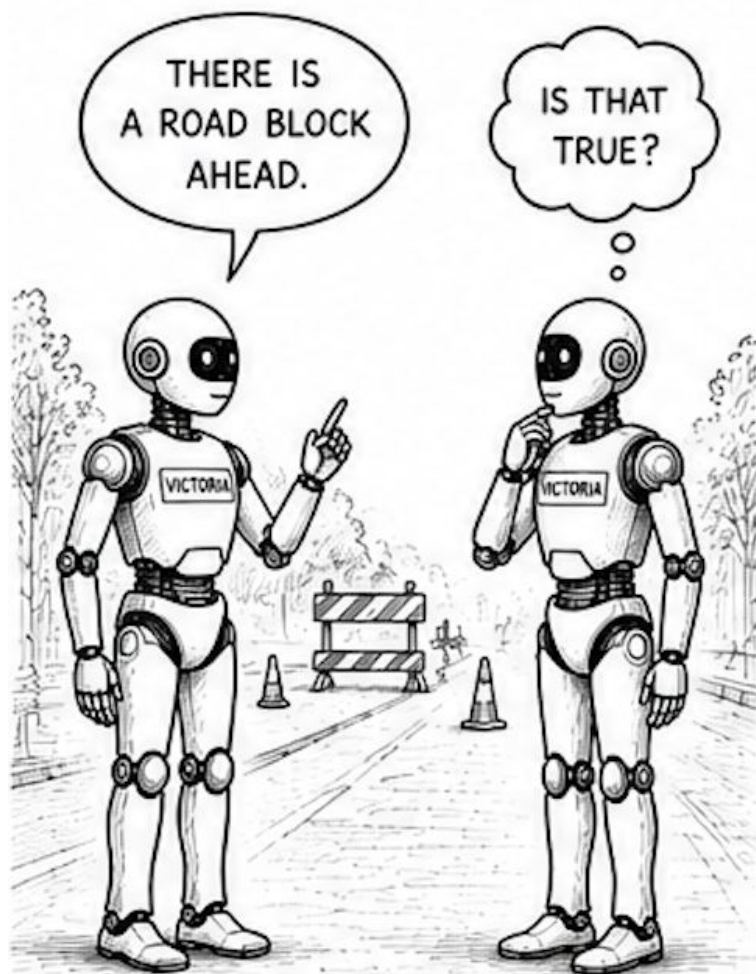
- What do we trust?





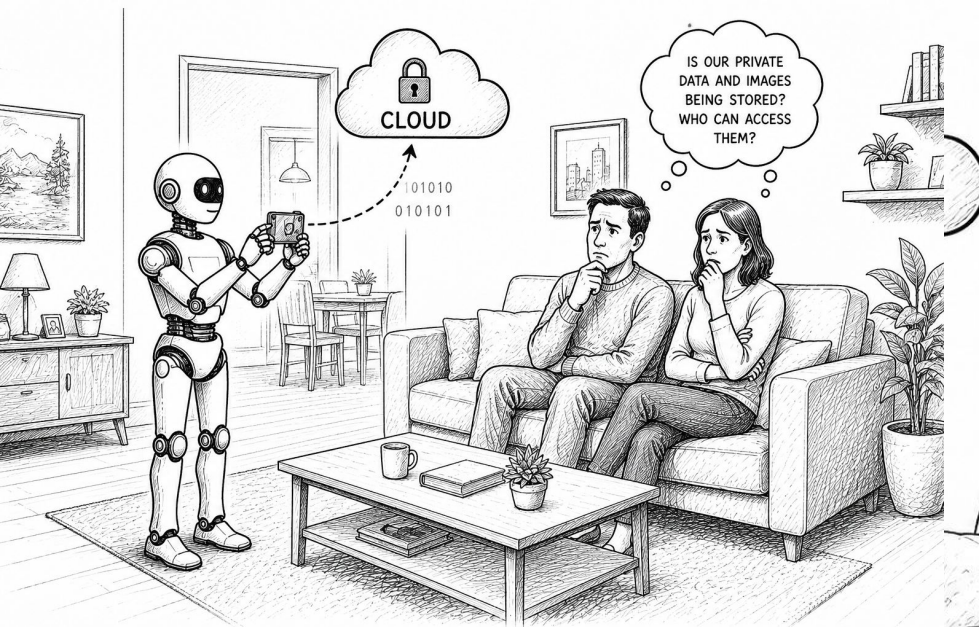


- Not just interaction, but we also need trustworthy actuation!
- Guarantees for certain actions to happen!



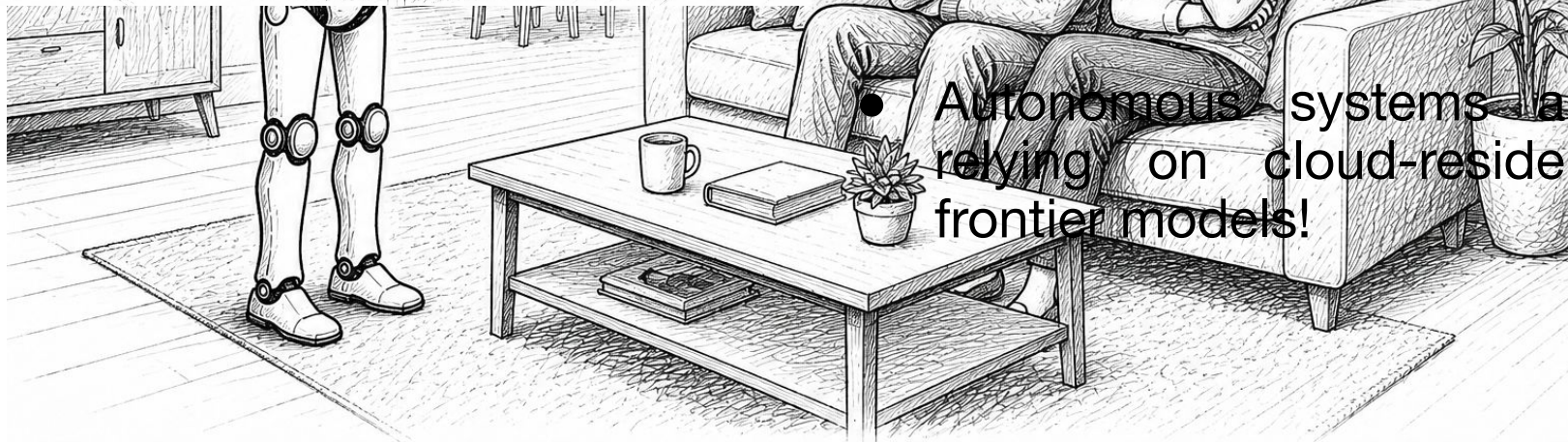
Secure Collaboration

- Trustworthy sensing and actuation between autonomous agents/robots/systems.
- We need mechanisms to ensure integrity of sensed data and ensure that this integrity is maintained in its lifecycle.
- Actuation and sensing can create a loop of interactions.



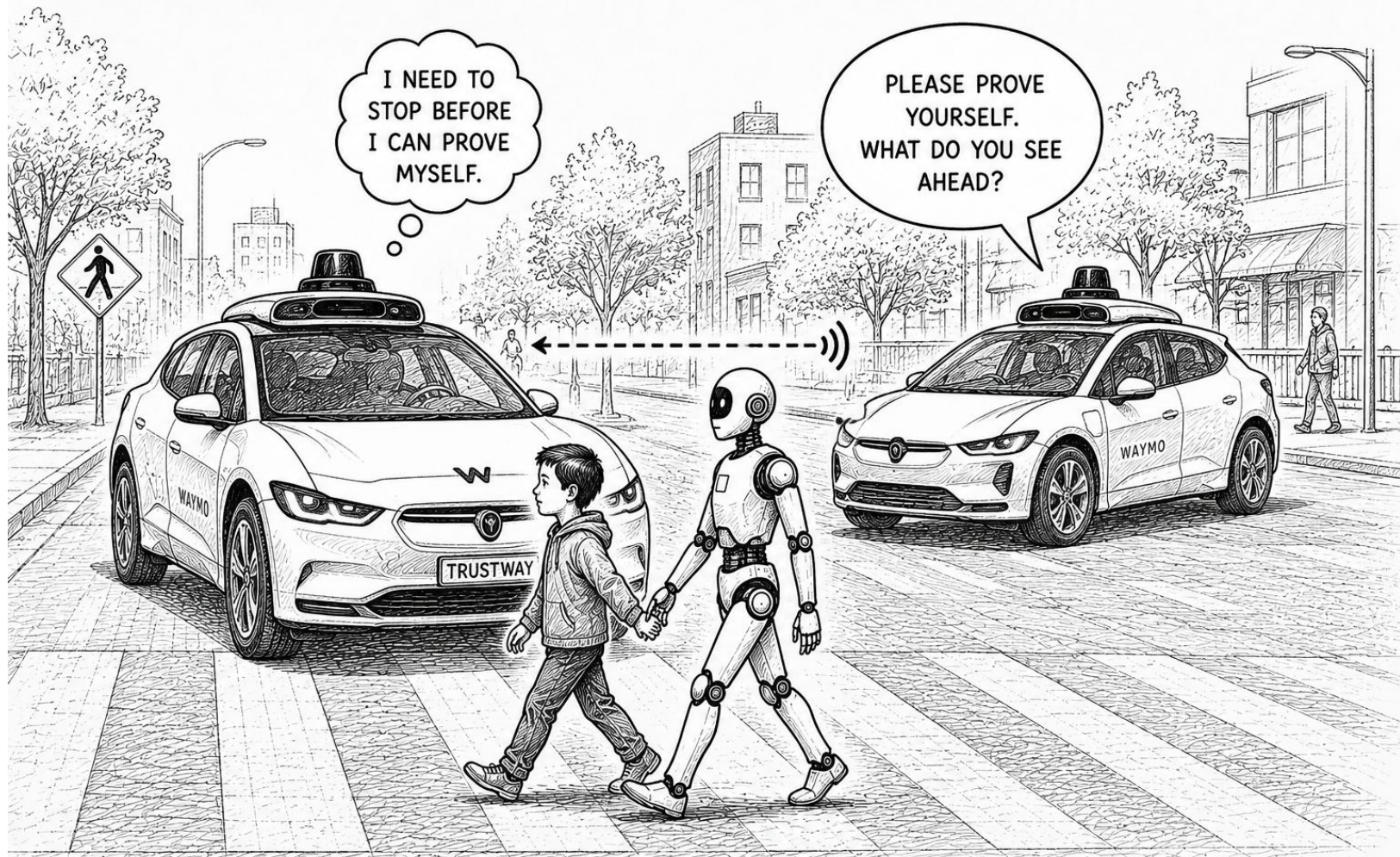
- Privacy is becoming an even bigger issue with the proliferation of autonomous systems, especially for humanoid at homes!

- Autonomous systems are relying on cloud-resident frontier models!



Privacy Concerns

- A growing problem!
- New collaborative methods are needed to address this need!



Availability-Security

- Real-time aspects of autonomous systems create an inherent conflict between security guarantees and availability requirements.
- Security services need to be co-designed!

Solutions?



- Emerging challenges as we integrate more autonomous agents in our environment!
- At UCLA, we have focused on various aspects of this problem.
- Here, I'm focusing on one of them (availability vs. security)

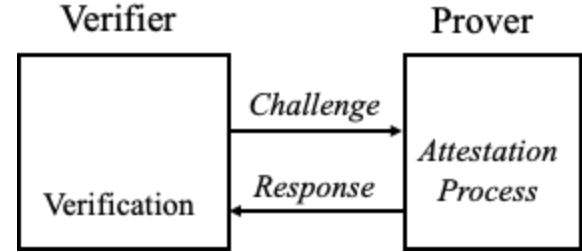
IDA: Hybrid Attestation with Support for Interrupts and TOCTOU

NDSS 2024

Motivation: Embedded Devices

Remote Attestation

High-end -> Hardware (TEE)



Motivation: Embedded Devices

Remote Attestation

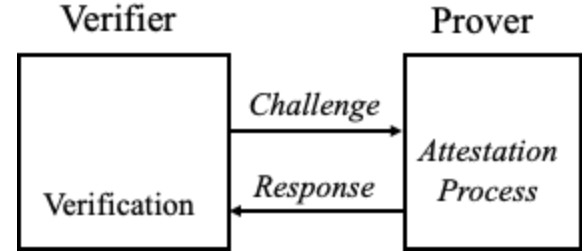
High-end -> Hardware (TEE)

Low-end

Examples

Software: ACES, RealSWATT, ...

Hybrid: GAROTA, VRASED, ...



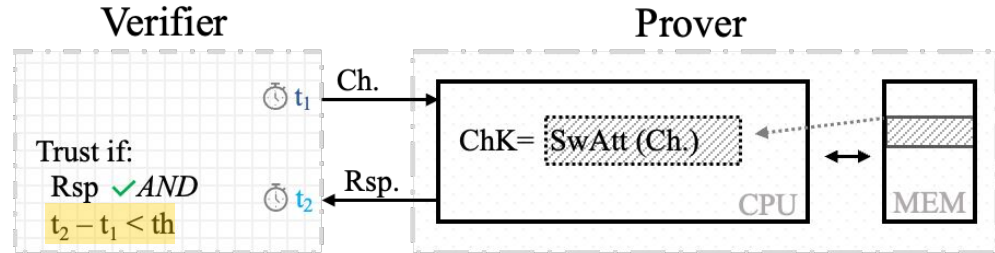
Motivation: Embedded Devices

Software: ❌ Strict assumptions

Request-to-Response time =>

❌ Low-latency attacks

❌ Network limitations



Motivation: Embedded Devices

Software: ✗ Strict assumptions

Request-to-Response time =>

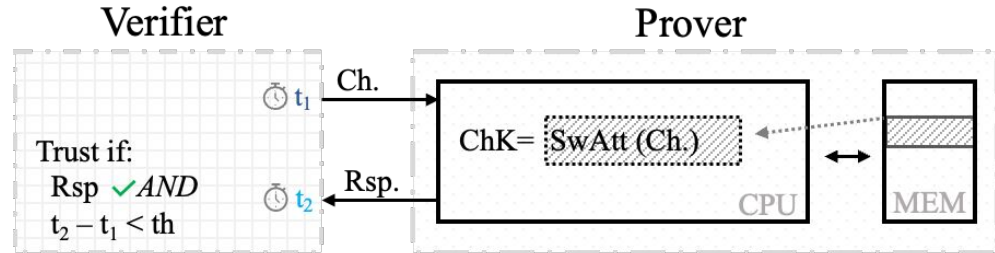
✗ Low-latency attacks

✗ Network limitations

✓ Overhead

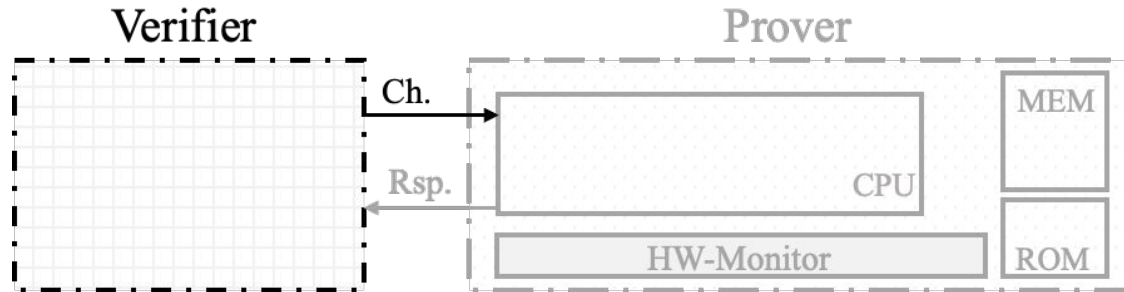
✓ Flexibility

✗ Robust



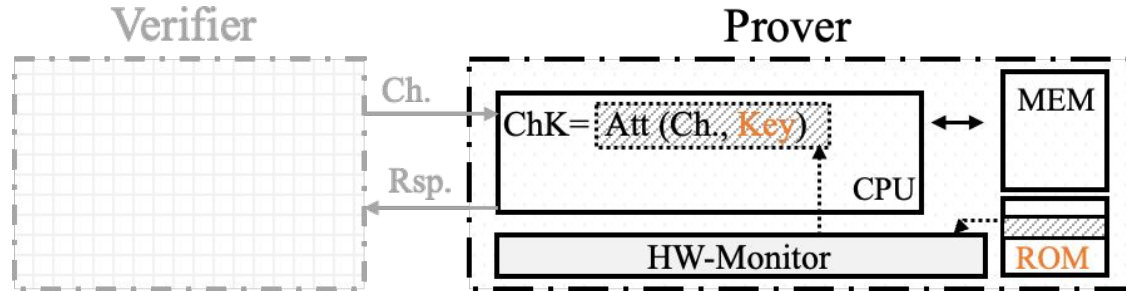
Motivation: Embedded Devices

Hybrid:



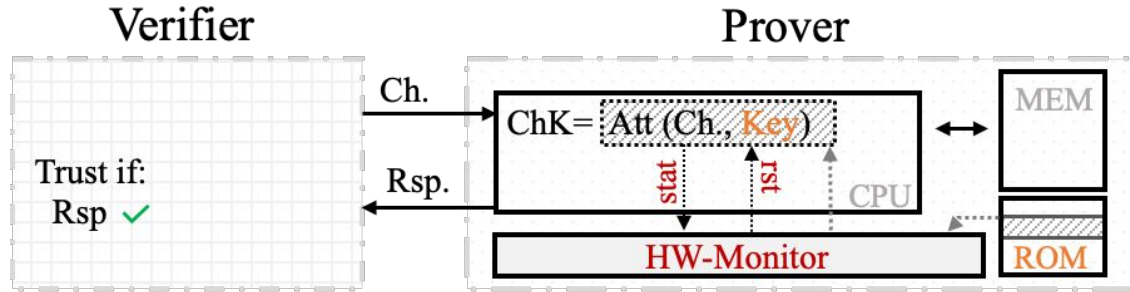
Motivation: Embedded Devices

Hybrid:



Motivation: Embedded Devices

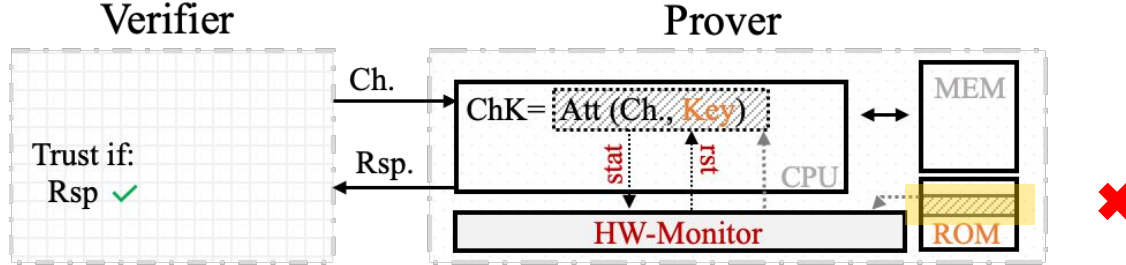
Hybrid:



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

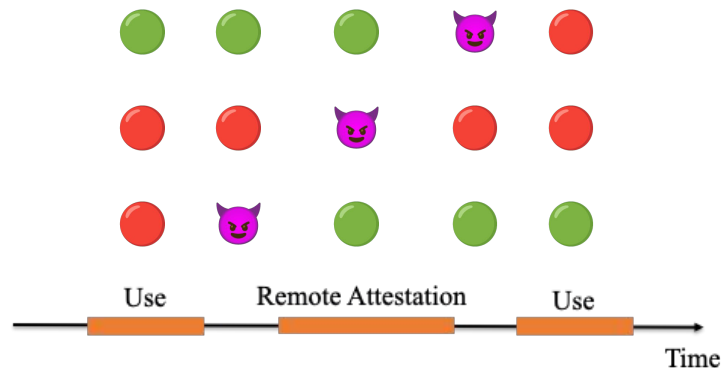
✖ Time-Of-Check-Time-Of-Use attack

👹 Adversary:

- Modifies memory after RA

- Hides the malware during RA

- Erase itself before RA

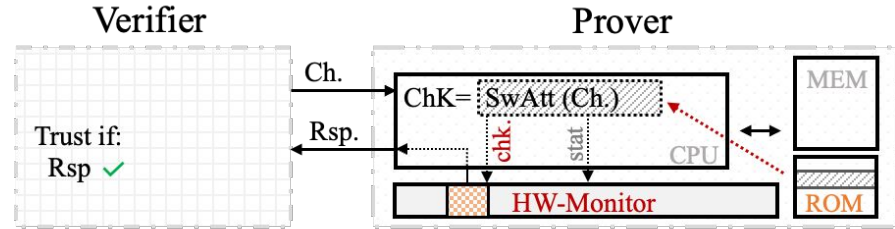


Introduction: IDA

Hybrid Remote Attestation

Flexibility (SW-based)

Security (Hybrid-based)

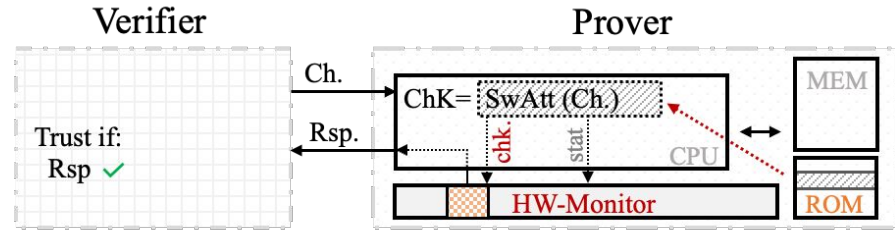


Introduction: IDA

Hybrid Remote Attestation

Flexibility (SW-based)

Security (Hybrid-based)



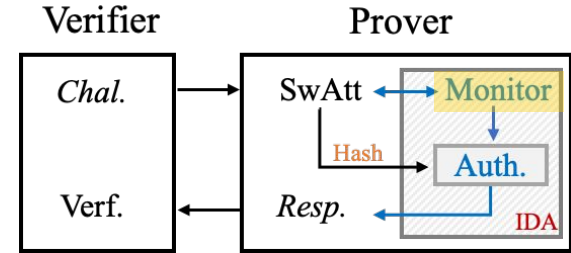
No secure access control for the key

✓ Interrupts

✓ DMA requests

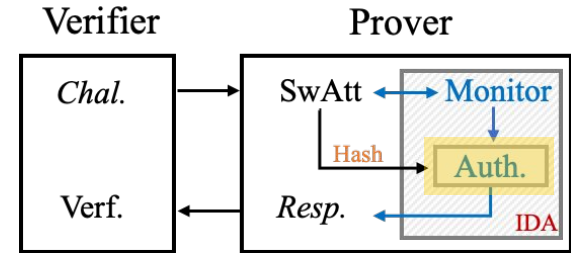
Introduction: IDA

- 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

Properties

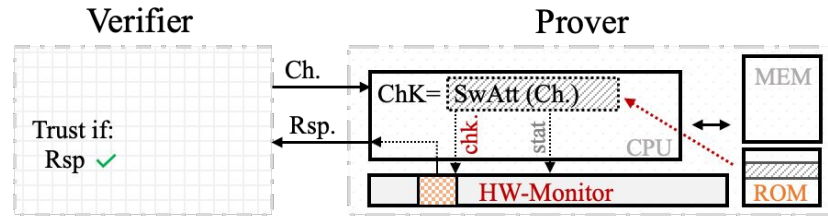
Relaxed

Atomic execution

Invocation flexibility

Flexible reset

DMA monitoring



Properties

Relaxed

Atomic execution

Invocation flexibility

Flexible reset

DMA monitoring

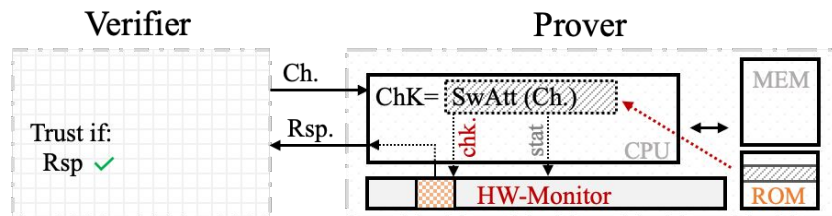
Security

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)

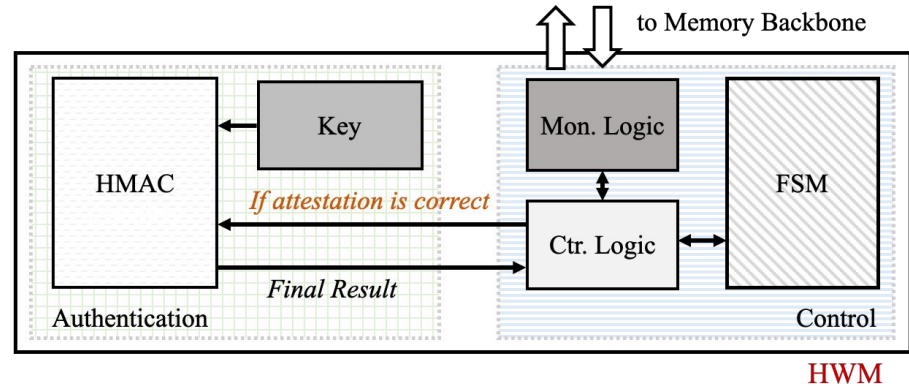
CPU Unit

Memory Backbone

System Architecture: IDA

Hardware Module (HWM)

Secure execution of Sw-Att



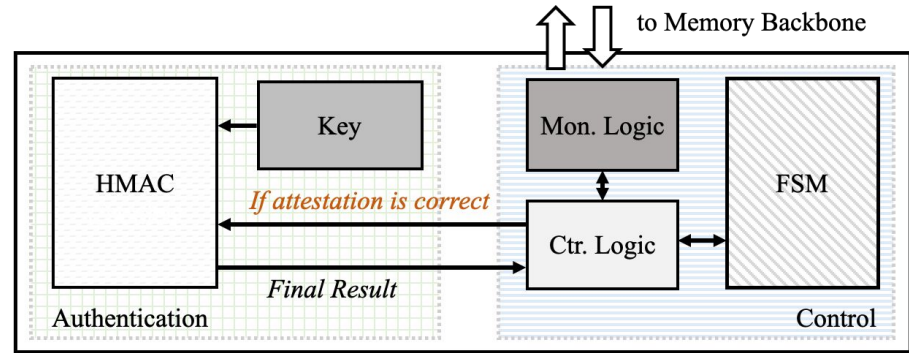
System Architecture: IDA

Hardware Module (HWM)

Secure execution of Sw-Att

Monitor

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



HWM

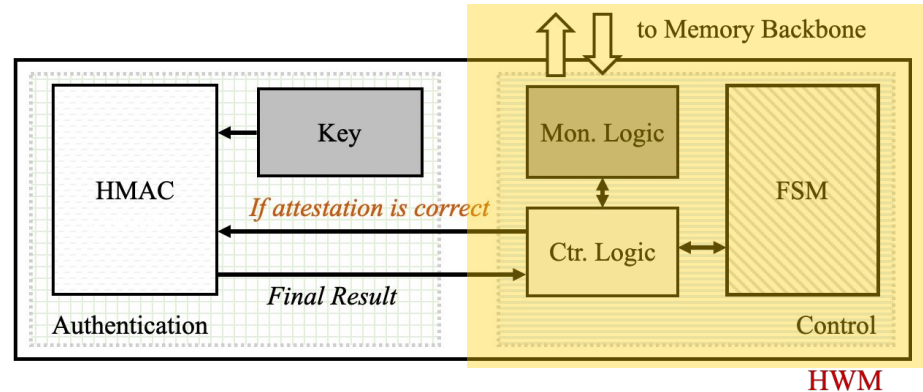
System Architecture: IDA

Controller Unit

Finite State Machine

Monitor

- PC
- Memory Address
- DMA Write
- Interrupt Request



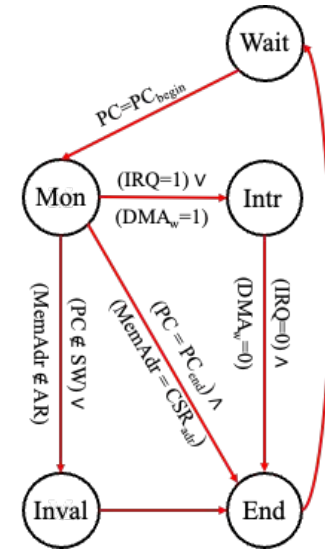
System Architecture: IDA

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 = CSRadr;  
4   /* Address for accessing HWM. */  
5   cHash = nonce;  
6   i = ARbegin;  
7   while ( $i < AR_{end}$ ) do  
8     | cHash = hash(Mem[i], cHash);  
9     | i ++;  
10    | if (inter == 1) then break;  
11  end  
12  Mem[CSR] = cHash;  
13  IdleLoop();  
14  /* To allow time for HWM to calculate the response. */  
15  cHash = Mem[CSR];  
16  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  $Resp.$ ;  

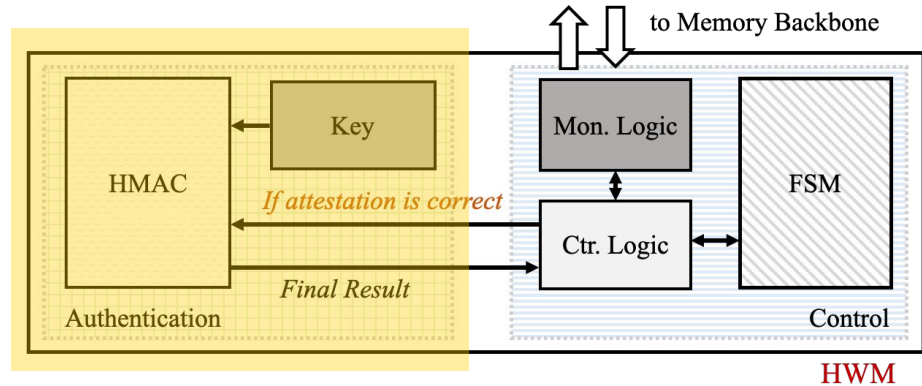

---

1 Function  $SW-Att$   
2   input:  $nonce, AR, inter$ ;  
3    $CSR = CSR_{adr}$ ;  
4   /* Address for accessing HWM. */  
5    $cHash = nonce$ ;  
6    $i = AR_{begin}$ ;  
7   while  $(i < AR_{end})$  do  
8     |  $cHash = hash(Mem[i], cHash)$ ;  
9     |  $i++$ ;  
10    | if  $(inter == 1)$  then break;  
11  end  
12   $Mem[CSR] = cHash$ ;  
13   $IdleLoop()$ ;  
14  /* To allow time for HWM to calculate the response. */  
15   $cHash = Mem[CSR]$ ;  
16  return  $cHash$ ;
```

System Architecture: IDA

Authentication Unit

- Validates integrity and authenticity
 - HMAC-SHA-256



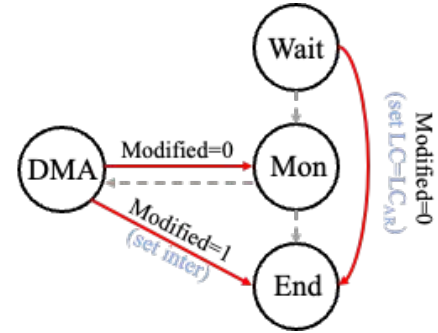
System Architecture: IDA+

TOCTOU

After last RA

Program is NOT modified

HWM sends Response (HMAC of Nonce)



System Architecture: IDA+

TOCTOU

After last RA

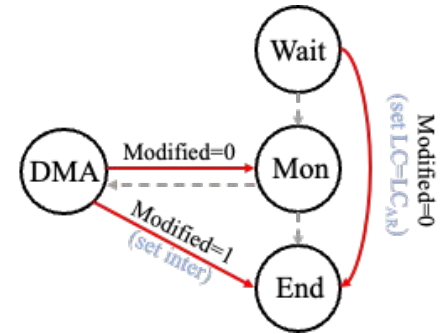
Program is NOT modified

HWM sends Response (HMAC of Nonce)

O.W.

Attestation

HWM sends NULL (random)



System Architecture: IDA+

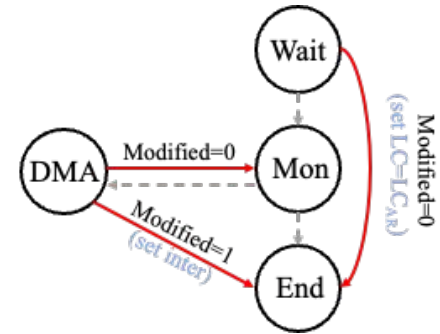
Interrupts

During interrupt handling

Monitor memory (writes)

NOT Modified

Continue



System Architecture: IDA+

Interrupts

During interrupt handling

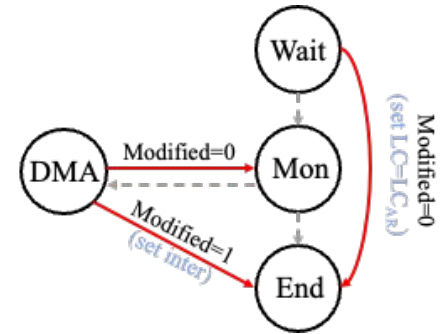
Monitor memory (writes)

NOT Modified

Continue

O.W.

Reset



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 HACLS* library

HWM: HMAC-SHA-256 implemented by Secworks

Evaluation: Overhead

3% Memory

MSP430 4 KB of ROM

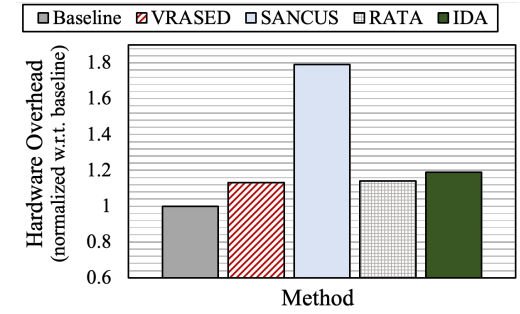
Evaluation: Overhead

19% Hardware

HWM Area:

54.4% Authentication

45.6% Controller

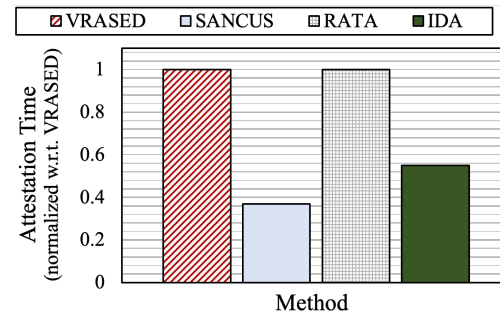


Evaluation: Overhead

≈ 2x Attestation Time

8 KB program memory (8 MHz Clock)

HMAC vs hash verification for each access

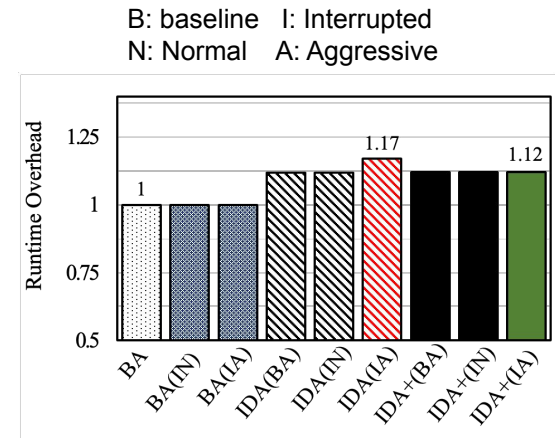


Evaluation: Overhead

12% Runtime

Benchmark: MiBench

FFT, Basicmath, dijkstra

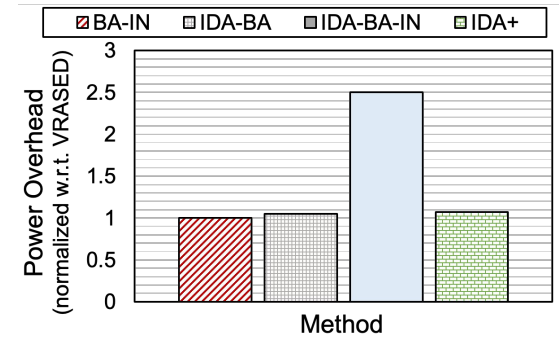


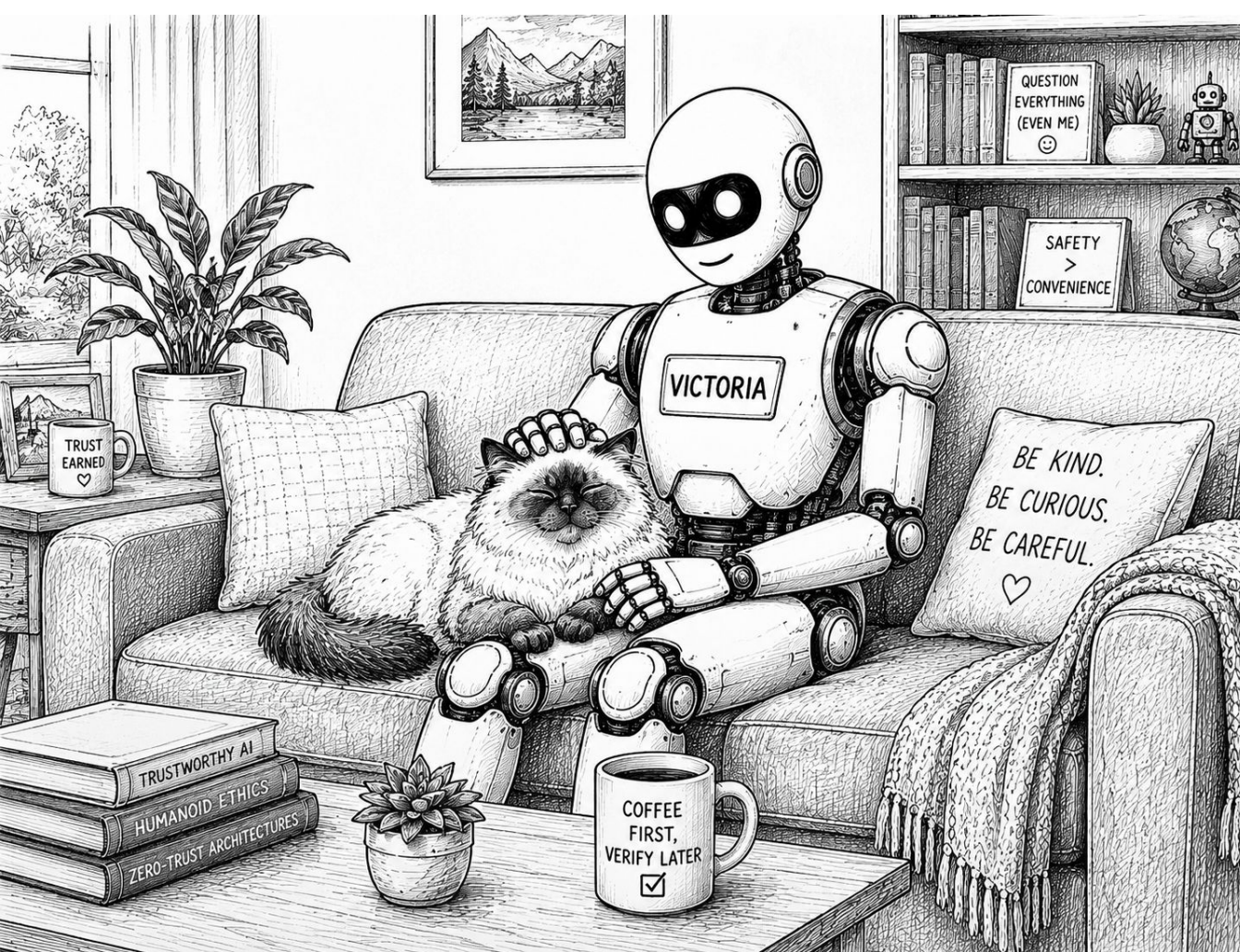
Evaluation: Overhead

2.5% Energy

Worst case:

Interrupt intervals $\approx$ Attestation time





UCLA Secure
Systems and
Architectures
Lab (SsysArch)

[https://ssysarch.
ee.ucla.edu/](https://ssysarch.ee.ucla.edu/)

Nader:
nsehat@ucla.edu