From Hardware Fingerprint to Access Token: Enhancing the Authentication on IoT Devices

Yue Xiao*, Yi He, Xiaoli Zhang, Qian Wang, Renjie Xie, Kun Sun, Ke Xu, Qi Li

IoT devices need reliable authentication

Embedded devices are an important part of our daily lives.

Car Key

Hardware Wallet

Smart Homes

They are associated with
- Daily Travel
- Personal Property
- Home Life
- …

IoT devices need reliable authentication

Embedded devices are an important part of our daily lives.

Car Key

Hardware Wallet

Smart Homes

They are associated with

- Daily Travel
- Personal Property
- Home Life
- …

Token-based authentication solutions suffer from token compromise.

Flipper Zero

Car Key Clone[1, 2] : The attacker uses Flipper Zero to copy the key fob, then unlocks the victim's car.

Risks brought by token compromise

- Property Loss
- Privacy Disclosure
- Tax Fraud
- …

[1] Hackers can clone tesla key fobs in seconds. https://www.esat.kuleuve n.be/cosic/news/fast-furious-and-insecure-passive-keyless-entry-and-st art-in-modern-supercars/.

[2] Flipper Zero Car Key Signal - Unlock Car Key FOB Hack. https://www.youtube.com/watch?v=HwdoHMVKTpU

IoT devices need reliable authentication

Embedded devices are an important part of our daily lives.

Car Key

Hardware Wallet

Smart Homes

They are associated with
- Daily Travel
- Personal Property
- Home Life
- …

Token-based authentication solutions suffer from token compromise.

Flipper Zero

Car Key Clone[1, 2]: The attacker uses Flipper Zero to copy the key fob, then unlocks the victim's car.

Risks brought by token compromise
- Property Loss
- Privacy Disclosure
- Tax Fraud
- …

Need unclonable authentication factors!

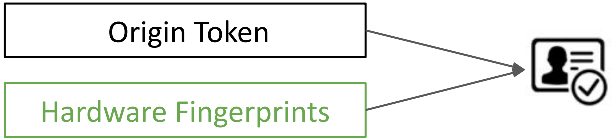
[1] Hackers can clone tesla key fobs in seconds. https://www.esat.kuleuve n.be/cosic/news/fast-furious-and-insecure-passive-keyless-entry-and-st art-in-modern-supercars/.
[2] Flipper Zero Car Key Signal - Unlock Car Key FOB Hack. https://www.youtube.com/watch?v=HwdoHMVKTpU

A Solution: Hardware-based Authentication

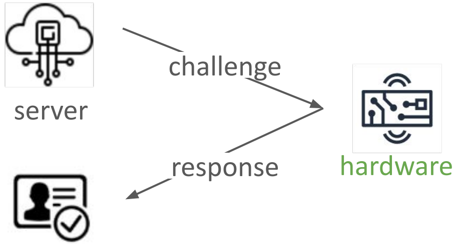
Unclonable?

Bind authentication to hardware fingerprints.

Two ways to use:



As new identifier



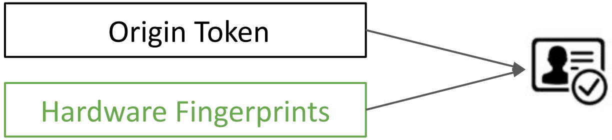
Challenge response protocol

A Solution: Hardware-based Authentication

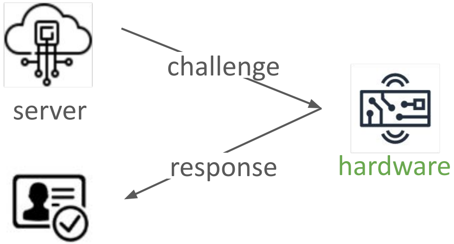
Unclonable?

Bind authentication to hardware fingerprints.

Two ways to use:



As new identifier



Challenge response protocol

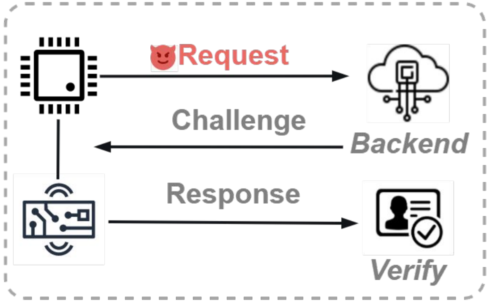
Limitations

- Require extra hardware that may not be supported on MCUs.
- Difficult to prevent man-in-the-middle (MiTM) adversaries.

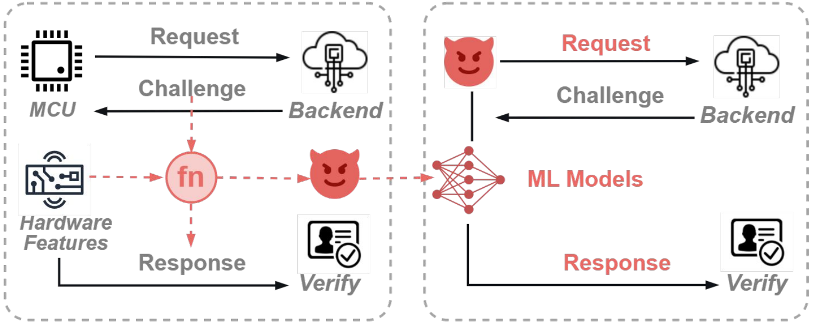
Existing Limitation: Man-in-the-middle Adversaries

IoT devices are resource constraint to adopt a secure implementation of TLS[2], and even do not encrypt messages[3].

Insecure Communication Channel



Reuse Attack

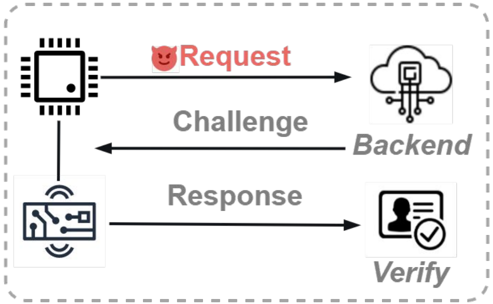


Mimic Attack

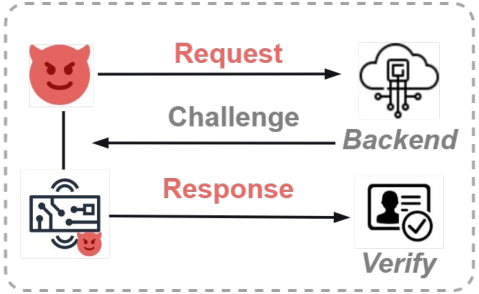
[2] Tls/pki challenges and certificate pinning techniques for iot and m2m secure communications. Daniel Díaz-Sánchez et al. IEEE Communications Surveys Tutorials, 2019.
[3] Breakmi: Reversing, exploiting and fixing xiaomi fitness tracking ecosystem. Marco Casagrande et al. IACR 2022.

Threat Model and Assumption

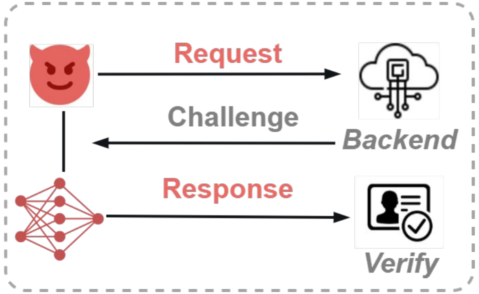
Attackers: attempt to impersonate legitimate devices.



Tampering Attack



Hardware Mimic Attack



Software Mimic Attack

Assumptions

- Devices are not compromised locally or remotely.

- A secure environment to collect hardware fingerprints (once).

Our Solution: Unique Hardware-based Access Token

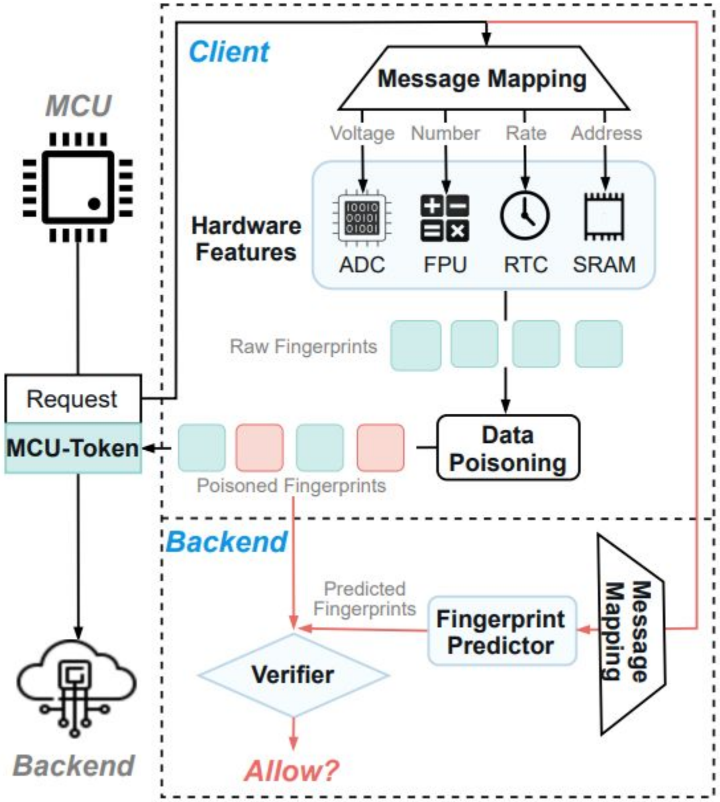
Key idea: Bind each request to a unique hardware-based access token.

Step-1: Collect hardware fingerprints (secure env)

Step-2: Generate token for the request

- 2-1: Map the request into hardware tasks
- 2-2: Obtain raw fingerprints via hardware
- 2-3: Generate token (poisoned fingerprints)
- 2-4: Send request with token to the backend

Step-3: Verify fingerprints on the backend



A Running Example

Request

```
/api/1/vehicles/{id}
{
	"op": DOOR_UNLOCK (0x0),



}
```

A Running Example

Request

/api/1/vehicles/{id}

{

"op": DOOR_UNLOCK (0x0),

"nonce": 1700902800

"token": [2631, 42822]

}

Message Mapping

Hash(0x0,1700902800)

=10 | 10101 | 0011

$task_{id}$ $task_{args}$

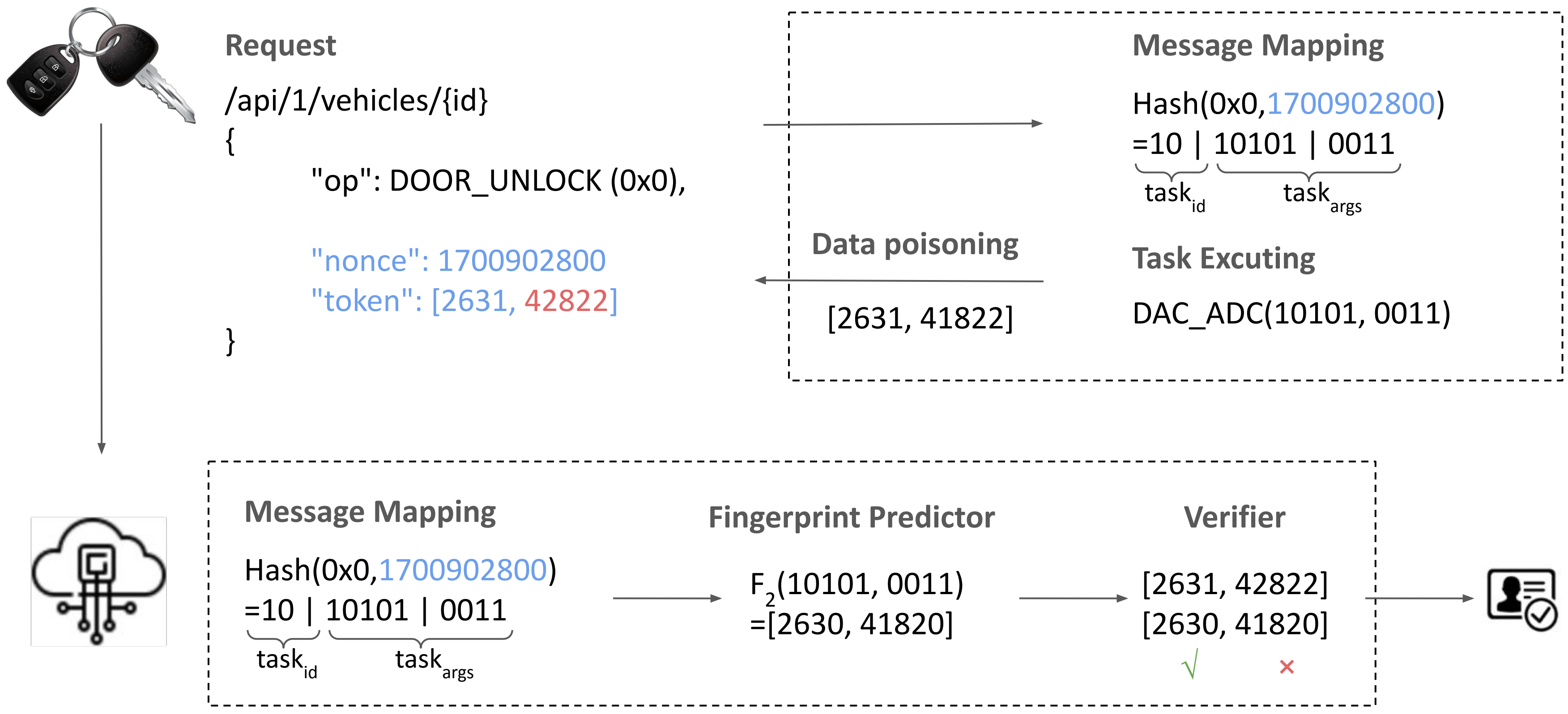
Task Excuting

DAC_ADC(10101, 0011)

Data poisoning

[2631, 41822]

A Running Example

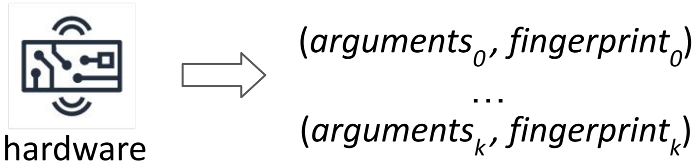


How to select and use hardware features?

Represent a hardware module as *(arguments, fingerprint)* pairs.

Select Feature: Check existing works and examine all potential features in datasheets.

Use Feature: Design execution tasks with arguments for each hardware module.



How to select and use hardware features?

Represent a hardware module as *(arguments, fingerprint)* pairs.

Select Feature: Check existing works and examine all potential features in datasheets.

Use Feature: Design execution tasks with arguments for each hardware module.

arguments →  → fingerprint

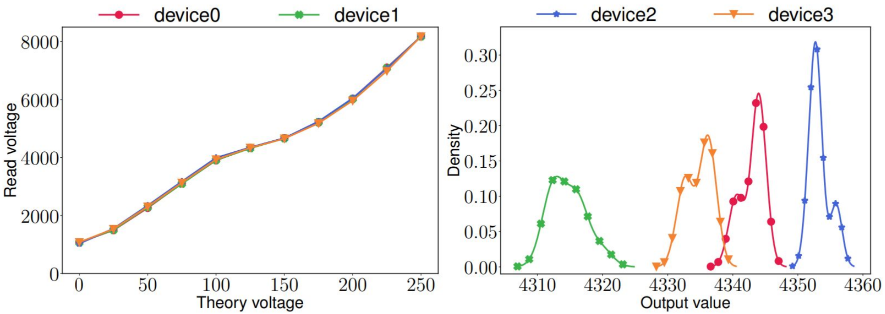
hardware

 ⇨ $(arguments_0, fingerprint_0)$ … $(arguments_k, fingerprint_k)$

hardware

Prevent Hardware Mimic Attacks

Hardware features are unique among devices. With the same arguments, the fingerprints are different.



How to ensure the uniqueness of each token?

Message Mapping: Bind task arguments to the request via hash function.

Request

$(op, payload_0, payload_1, \ldots)$ → $Hash(op, payload_0, nonce) = h_1$, $Hash(op, payload_1, nonce) = h_2$ → $Hash(h_1, h_2, \ldots) = digest\ (tasks)$

How to ensure the uniqueness of each token?

Message Mapping: Bind task arguments to the request via hash function.

Request

(op, $payload_0$, $payload_1$, …) → Hash(op,$payload_0$,nonce) = h_1 ; Hash(op,$payload_1$,nonce) = h_2 → Hash(h_1,h_2,...) = digest (tasks)

Uniqueness → Add nonce: The same operations have different digests.

Resist tampering → Hash function: Changes to the message will change the tasks.

How to prevent the software mimic attack?

Collect (*arguments, fingerprint*) pairs and learn the relationship.

How to prevent the software mimic attack?

Collect (*arguments, fingerprint*) pairs and learn the relationship.

Make tasks more complex ⇒

- Need to explore hardware further
- Simple relations should be discarded
- More powerful attackers can still learn

How to prevent the software mimic attack?

Collect (*arguments, fingerprint*) pairs and learn the relationship.

Make tasks more complex ⇨

- Need to explore hardware further
- Simple relations should be discarded
- More powerful attackers can still learn

Disrupt the learning process ⇨

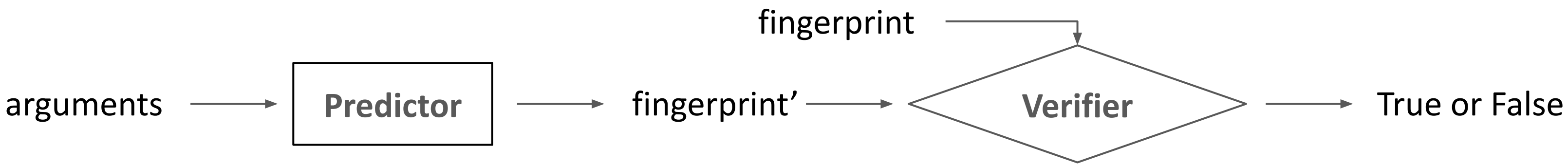
- Hardware independent
- All relationships can be used
- Fault data will fail the learning

Implementation: Select a portion of the fingerprints (e.g., 5 out of 10) and poison them as,

$$fp_{poisoned} = fp_{raw} * (noise + 1) + C$$

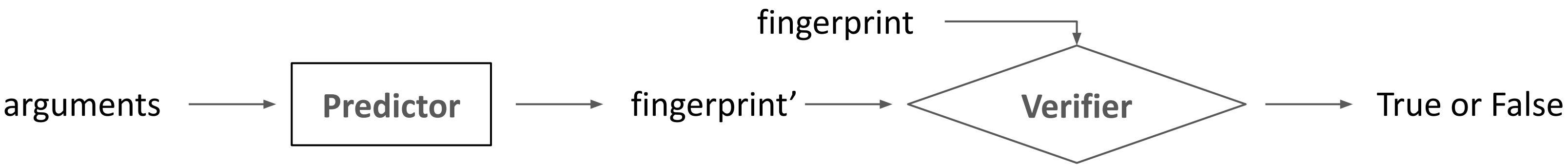
How to verify token at the backend?

Learn from hardware and compare fingerprints.



How to verify token at the backend?

Learn from hardware and compare fingerprints.



Set up: Collect enough (*argments, fingerprint*) pairs for training. (secure env)

Authenticate: Count the number of fingerprints verified.

total_num: total number of pairs

used_num: number of non-poisoned pairs

accept_num: threshold

The backend does not know if a pair is poisoned, but just counts the verified number.

MCU-Token Implementation and Evaluation Setup

Source code:

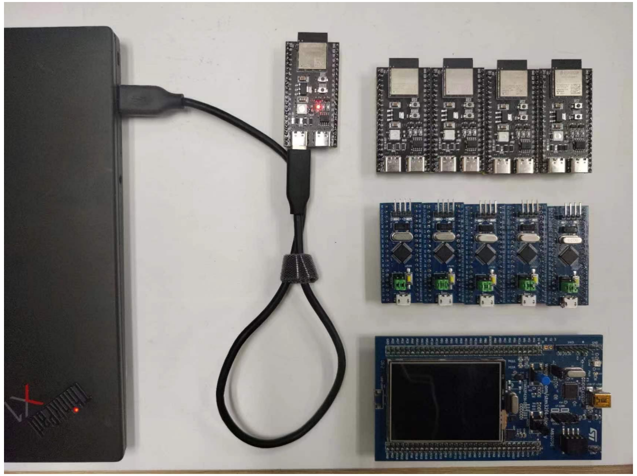
https://github.com/IoTAccessControl/MCU-Token

Selected hardware features

Modules	Features Description
DAC/ADC	Voltage features.
FPU	Float point arithmetic features.
PWM	Voltage and frequency features.
RTC	Frequency features and phase features.
SRAM	Storage medium features.

Hardware devices

Model-brand	Microcontroller	Frequency	# of devices
ESP32S2	Xtensa LX7	240MHz	30
STM32F103	Cortex M4	72MHz	20
STM32F429	Cortex M4	180MHz	10



Usability of Different Hardware Features

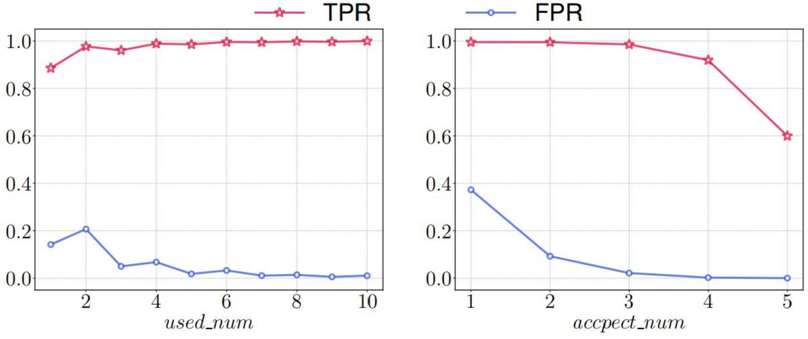
Evaluation on different hardware features

	ESP32S2		STM32F429		STM32F103	
	TPR	FPR	TPR	FPR	TPR	FPR
DAC_ADC	83.74	8.58	82.73	16.83	96.25	37.90
FPU	76.59	38.90	83.50	29.94	76.65	36.63
PWM	84.83	17.54	84.90	37.67	80.00	35.57
RTCFre	91.76	1.96	89.88	7.49	99.19	1.96
RTCPha	77.04	58.38	73.88	58.10	74.56	36.88
SRAM	94.27	0.01	98.69	0.05	96.89	0.03
Ensemble	96.63	9.44	97.06	14.10	97.94	14.31
Ensemble*	98.47	1.06	97.67	6.89	98.68	1.64

* The results of excluding useless features, i.e., FPU and RTCPhra for ESP32S2, PWM and RTCPhra for STM32F249, DAC/ADC, FPU and PWM for STM32F103.

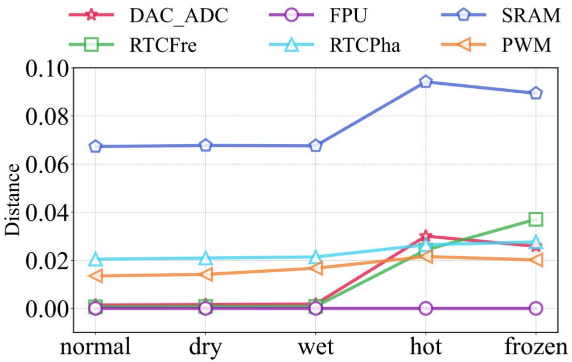
TPR: The rate at which a device is correctly verified
FPR: The rate at which a device is identified as another device

Various parameter settings



(a) Different $usedNum$ (b) Different $acceptNum$

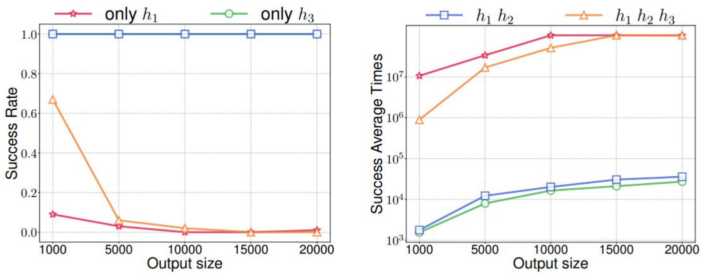
Environment settings



Security Against Various Attacks

Success Rate: The rate at which attackers successfully fool the backend.

Tampering Attack



(a) Tampering attack success rate
(b) Attack success average number

Tampering Attack: Change the request, but keep the tasks the same as before.

(a) Success rate < 0.1%
(b) Retry times for a successful attack > 10^7

Hardware Mimic Attack

	ESP32S2	STM32F103	STM32F429
ESP32S2	0.0188	0.0000	0.0000
STM32F103	0.0001	0.0606	0.0078
STM32F429	0.0000	0.0000	0.1058

Use the device in the row to mimic the device in the column.
Success rate: < 10% (average < 1%)

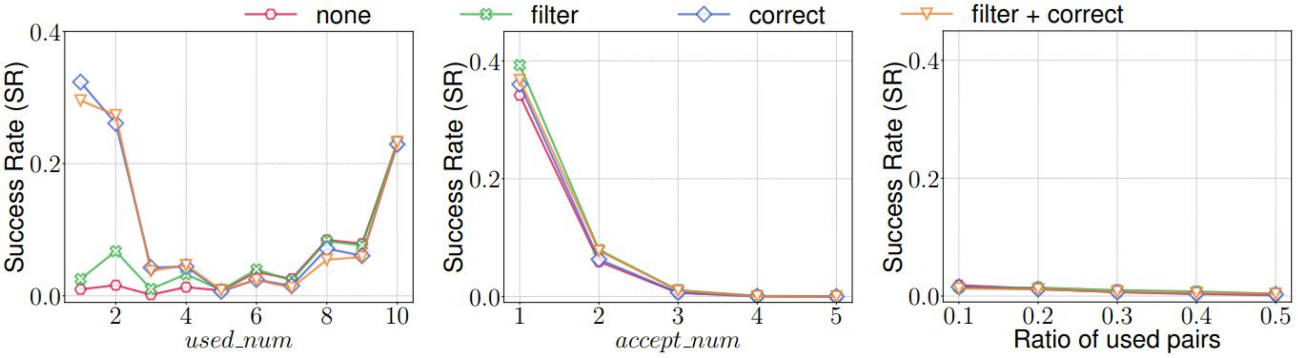
Identify the poisoned pairs

	DAC/ADC	RTCFre	SRAM	PWM
Unsupervised learning	0.5201	0.5042	0.4993	0.5354
Supervised learning	0.5142	0.5220	0.5409	0.5293
Incremental learning	0.5120	0.5005	0.5032	0.4889
Extra-device	0.9682	0.5745	0.4959	0.8991

Near random guessing via software methods

Security Against Various Attacks

Software Mimic Attack



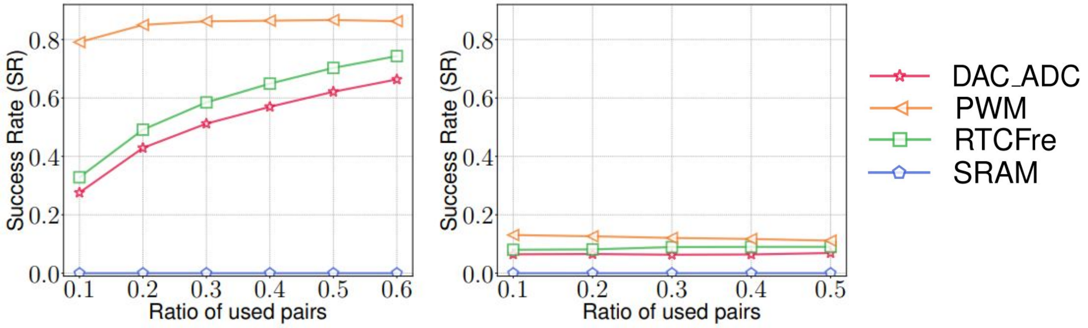
(a) Different *usedNum* (b) Different *acceptNum* (c) Different used ratio

(a) Used_num: the percentage of normal pairs.

(b) Accept_num: the difficulty of passing authentication

(c) Ratio: the ratio of normal pairs obtained by attackers

Results when authenticating with only one feature



(a) Authenticating without protection (b) Authenticating with protection

The poisoned pairs decrease the success rate of attackers.

Poisoned pairs prevent attackers from learning the relationships.

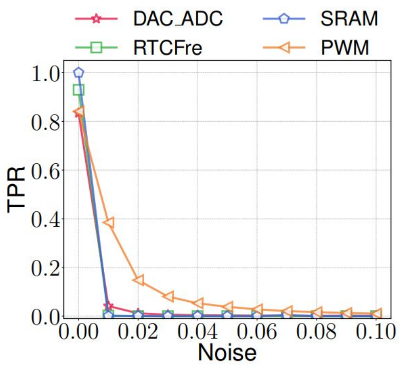
Other Evaluations

Do poisoned pairs affect normal authentication?

We use poisoned pairs for authentication.

(Right Figure) Poisoned pairs are rejected by the backend.

Normal pairs ensure that normal authentication passes.



What about the overhead of power and time?

Baseline: AES-128 encryption

	Encrypt	Voltage	FPU	Clock	Storage
ESP32S2	0.23W 2ms	0.22W 23ms	0.22W 97ms	0.19W 10ms	0.17W 10ms
STM32F429	0.74W 2ms	0.79W 39ms	0.76W 8ms	0.79W 47ms	0.71W 1ms
STM32F103	0.15W 5ms	0.16W 114ms	0.16W 17ms	0.15W 8ms	0.15W 1ms

We test the power and time to encrypt and get fingerprints.

The power consumption is low.

Time is acceptable (31ms in average).

Conclusion

- We perform a systematic study on hardware features for fingerprinting the commercial-off-the-shell MCUs.
- We introduce MCU-Token, a hardware fingerprint based authentication mechanism that resists various attacks.
- We prototype MCU-Token and demonstrate its usability and performance by evaluating it on 60 IoT devices of three types.

Thanks for listening
Q&A

Paper

Code