ProSpeCT: Provably Secure Speculation for the Constant-Time Policy

August 11th 2023

32nd USENIX Security Symposium

Lesly-Ann Daniel
KU Leuven

Marton Bognar
KU Leuven

Job Noorman
KU Leuven

Sébastien Bardin
CEA List

Tamara Rezk
INRIA

Frank Piessens
KU Leuven

I want to protect my secrets...

```
char array[len]
char mysecret
if (idx < len)
    x = array[idx]
    leak(x)
```

Easy: Constant-Time Programming!

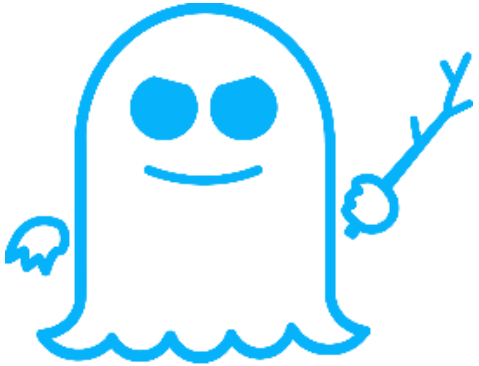
`mysecret` $\not\rightarrow$ `leak()`

De facto standard for crypto

… even against Spectre attacks

```
char array[len]
char mysecret
if (idx < len)
    x = array[idx]
    leak(x)
```

Predict branch taken



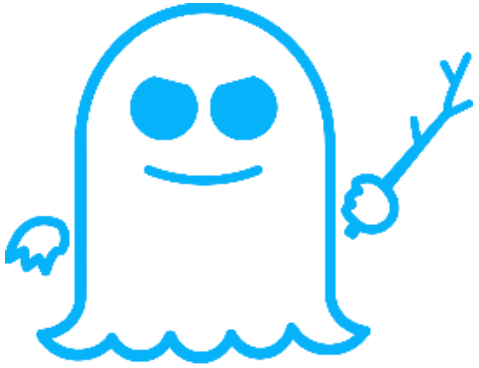
… even against Spectre attacks

```
char array[len]
char mysecret
if (idx < len)
    x = array[idx]
    leak(x)
```

Predict branch taken

`x = mysecret`

Leaks `mysecret` to microarchitecture!



How can I protect my code?

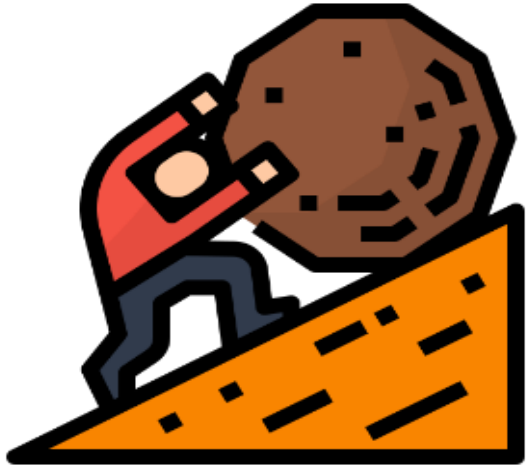
Constant-Time Foundations for the New Spectre Era

Sunjay Cauligi[†] Craig Disselkoen[†] Klaus v. Gleissenthall[†]
Dean Tullsen[†] Deian Stefan[†] Tamara Rezk[⋆] Gilles Barthe[♠♣]

[†]UC San Diego, USA [⋆]INRIA Sophia Antipolis, France
[♠]MPI for Security and Privacy, Germany [♣]IMDEA Software Institute, Spain

Speculative constant-time

- Hard to reason about
- New speculation mechanisms?



We need Secure Speculation for Constant-Time!

Developers should not care about speculations

Hardware shall not speculatively leak secrets

But still be efficient and enables speculation

Hardware defense:
Secure speculation for constant-time!

Hardware Secrecy Tracking

Software side

- Label secrets
- Constant-time program

Hardware side

- Track security labels
- Secrets do not speculatively flow to insecure instructions

ConTExT: A Generic Approach for Mitigating Spectre

Michael Schwarz[1], Moritz Lipp[1], Claudio Canella[1], Robert Schilling[1,2], Florian Kargl[1], Daniel Gruss[1]
[1]Graz University of Technology [2]Know Center GmbH

SpectreGuard: An Efficient Data-centric Defense Mechanism against Spectre Attacks

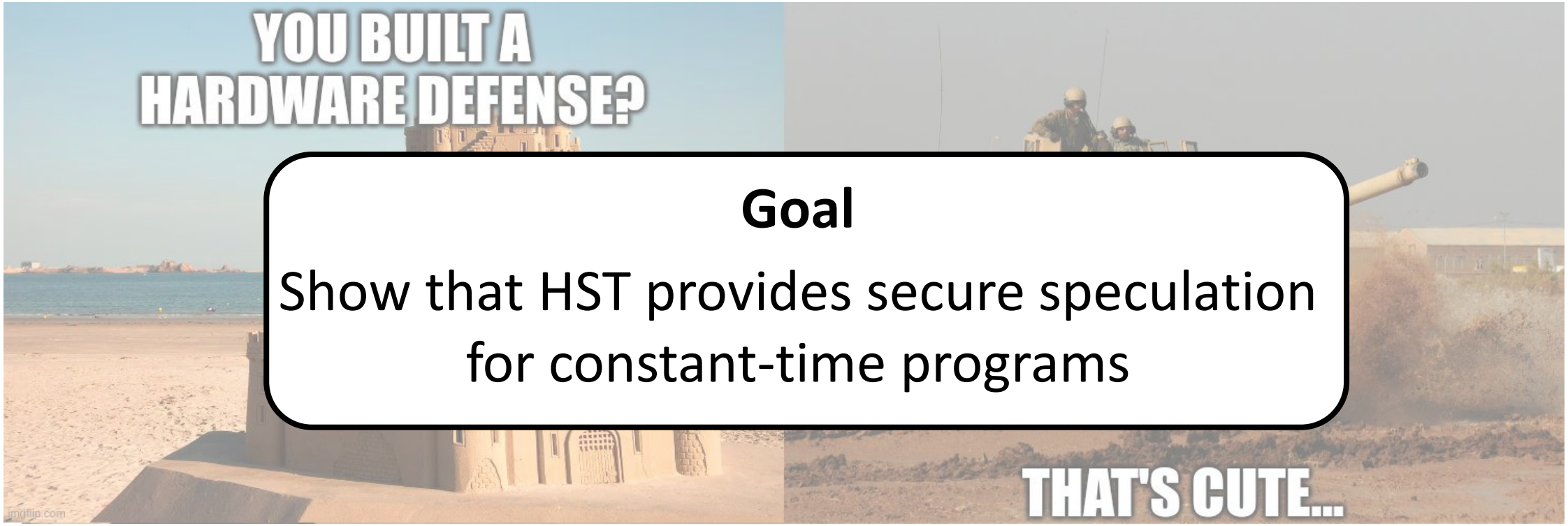
Jacob Fustos University of Kansas
Farzad Farshchi University of Kansas
Heechul Yun University of Kansas

Speculative Privacy Tracking (SPT): Leaking Information From Speculative Execution Without Compromising Privacy

Rutvik Choudhary UIUC, USA
Jiyong Yu UIUC, USA
Christopher W. Fletcher UIUC, USA
Adam Morrison Tel Aviv University, Israel

How do I know that my defense works?

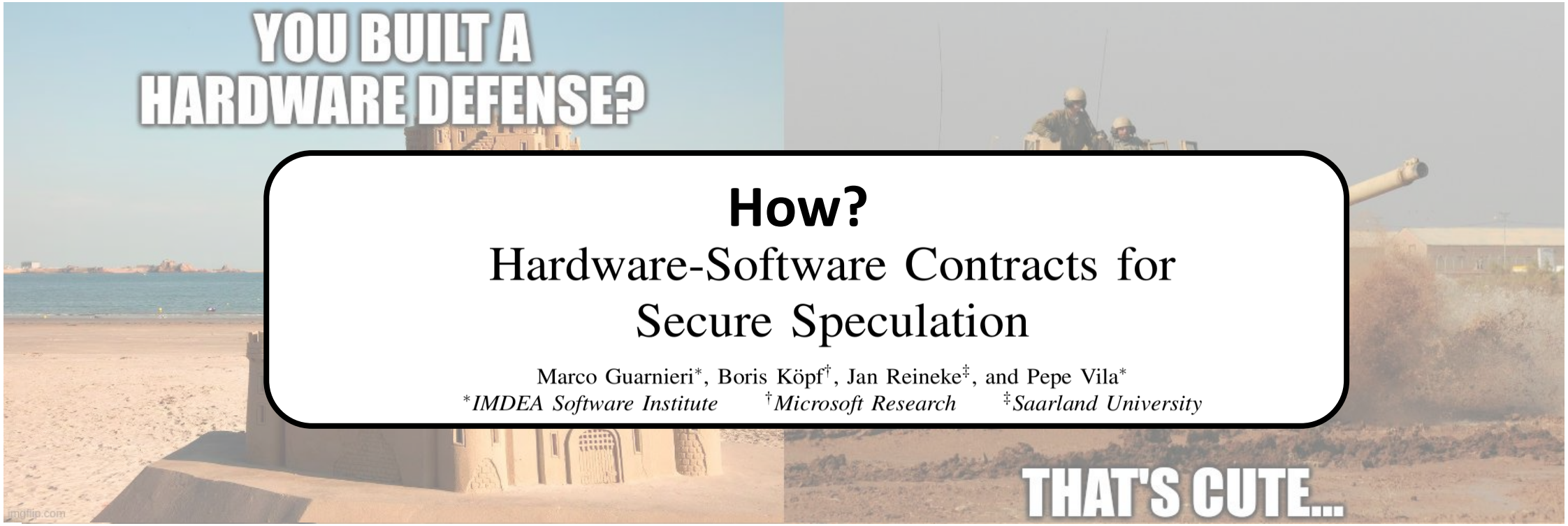
How do I know that my defense works?



Goal

Show that HST provides secure speculation for constant-time programs

How do I know that my defense works?



How?

Hardware-Software Contracts for Secure Speculation

Marco Guarnieri*, Boris Köpf†, Jan Reineke‡, and Pepe Vila*
*IMDEA Software Institute †Microsoft Research ‡Saarland University

Challenges

- Adapt HW/SW contract framework to account for:
  - All existing speculation mechanisms (Spectre, LVI)
  - Futuristic speculation mechanisms (value prediction)
  - Declassification
- Evaluation: hardware costs?

Our contributions

- ProSpeCT: Formal processor model with HST
  - Proof: constant-time programs do not leak secrets
  - Allows for declassification
  - Generic: all Spectre variants / LVI

- [In paper] First to consider (Load) Value Speculation
  - Novel insight: sometimes need to rollback *correct* speculations for security

- Implementation in a RISC-V microarchitecture
  - First synthesizable implementation
  - Evaluation: hardware cost, performance, annotations

Illustration with Spectre-v1

```
char array[len]
char mysecret
if (idx < len)
    x = array[idx]
    leak(x)
```

Consider `idx = len`

Illustration with Spectre-v1

```
char array[len]
secret char mysecret
if (idx < len)
    x = array[idx]
    leak(x)
```

Developer marks secrets

Consider `idx = len`

Illustration with Spectre-v1

```
char array[len]
secret char mysecret
if (idx < len)
    x = array[idx]
    leak(x)
```

Developer marks secrets

Speculative execution

Consider `idx = len`

Illustration with Spectre-v1

```
char array[len]
secret char mysecret
if (idx < len)
    x = array[idx]
    leak(x)
```

Developer marks secrets

Speculative execution

`x = mysecret:secret`

Consider `idx = len`

Illustration with Spectre-v1

```
char array[len]
secret char mysecret
if (idx < len)
    x = array[idx]
    leak(x)
```

Developer marks secrets

Speculative execution

`x = mysecret:secret`

Speculative execution + **secret**
=
`x` *not* forwarded to `leak`

Consider `idx = len`

ProSpeCT: Generic formal processor model for HST

Semantics of out-of-order speculative processor with HST

All public values are leaked & predictions can depend on any public value

Declassify = write secrets to public memory

Security proof

Constant-time programs do not leak secrets

Implementation

Prototype RISC-V implementation

On top of Proteus modular RISC-V processor

- Branch target prediction
- Conservative approach
- Open source
  https://github.com/proteus-core/prospect

Limited Hardware Cost

- LUTs: +17%
- Registers: +6%
- Critical path: +2%

Evaluation

4 primitives (HACL*)

- Annotate secret
- Ensure no secrets spilled
- Stack public in 3/4 cases
- ≤1h / primitive

Performance overhead (benchmark from [1])

Speculation/Crypto	**25/75**	**50/50**	**75/25**	**90/10**
Precise (Key)	**0%**	**0%**	**0%**	**0%**
Conservative (All)	10%	25%	36%	45%

No overhead in software for constant-time code when secrets are precisely annotated

[1] Jacob Fustos, Farzad Farshchi, and Heechul Yun. “SpectreGuard: An Efficient Data-Centric Defense Mechanism against Spectre Attacks”. In: DAC. 2019

Conclusion

Software informs hardware about secrets

Strong security guarantees

End-to-end security for constant-time programs

Low overhead

No software overhead for constant-time code

github.com/proteus-core/prospect

Icons made by **Freepik**, **Vectors Market, monkik** from www.flaticon.com

Backup

Illustration with LVI

LVI. Inject values at faulting loads

```
char A[16]   // public memory
char secret // secret memory
x = load idx
y = load A[x]
leak(y)
```

No defense

Attacker injects `x = 16`

`y = secret`

`secret` is transiently leaked!

Akin to Load Value Prediction

Illustration with LVI

```
char A[16]  // public memory
char secret // secret memory
x = load idx
y = load A[x]
leak(y)
```

ProSpeCT

Developer annotates secret memory

Akin to Load Value Prediction

Illustration with LVI

```
char A[16]  // public memory
char secret // secret memory
x = load idx
y = load A[x]
leak(y)
```

ProSpeCT

Developer annotates secret memory

Attacker injects `x = 16`

`y = secret:H`

Akin to Load Value Prediction

Illustration with LVI

```
char A[16]  // public memory
char secret // secret memory
x = load idx
y = load A[x]
leak(y)
```

ProSpeCT

Developer annotates secret memory

Attacker injects `x = 16`

`y = secret:H`

`secret` is not forwarded to `leak`

Akin to Load Value Prediction

Design Choices

Software side	Hardware side
• Label secret memory • Constant-time program • Secret written to public memory is declassified	• Track security labels • Secrets do not speculatively flow to insecure instructions • Predictions do not leak secrets

Code without secret free speculation
Constant-time programs only block mispredictions

Load Prediction: Rollback correct executions?

```
char secret // secret memory
x = load secret
y = x + 4
```

Execution 1: `secret=0`

Execution 2: `secret=1`

Predict load value to 0

```
x = 0 (?); y = 4
```

```
x = 0 (?); y = 4
```

Load Prediction: Rollback correct executions?

```
char secret // secret memory
x = load secret
y = x + 4
```

Execution 1: `secret=0`

Execution 2: `secret=1`

Predict load value to 0

```
x = 0 (?); y = 4
```

```
x = 0 (?); y = 4
```

Resolve

```
x = 0; y = 4
```

```
x = 1
```

Commit if secret = 0 vs Rollback if secret 0

Implicit resolution-based channel

Load Prediction: Rollback correct executions?

```
char secret // secret memory
x = load secret
y = x + 4
```

Execution 1: `secret=0`

Execution 2: `secret=1`

Predict load value to 0

```
x = 0 (?); y = 4
```

```
x = 0 (?); y = 4
```

Resolve

```
x = 0:H
```

```
x = 1:H
```

Solution: Always rollback when actual value is secret

Future Work?

Formal model

- Cryptographic security down to the hardware?

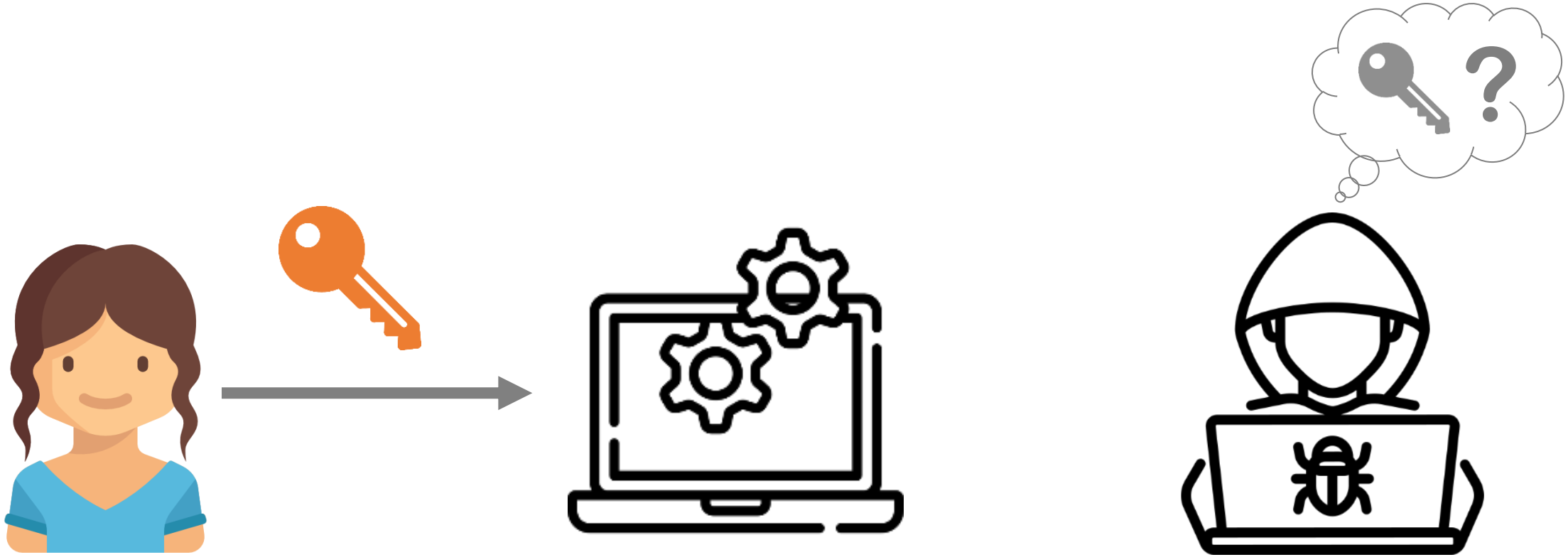
Compiler-support

- Separate secret from public memory
- Ensure no unintentional declassification

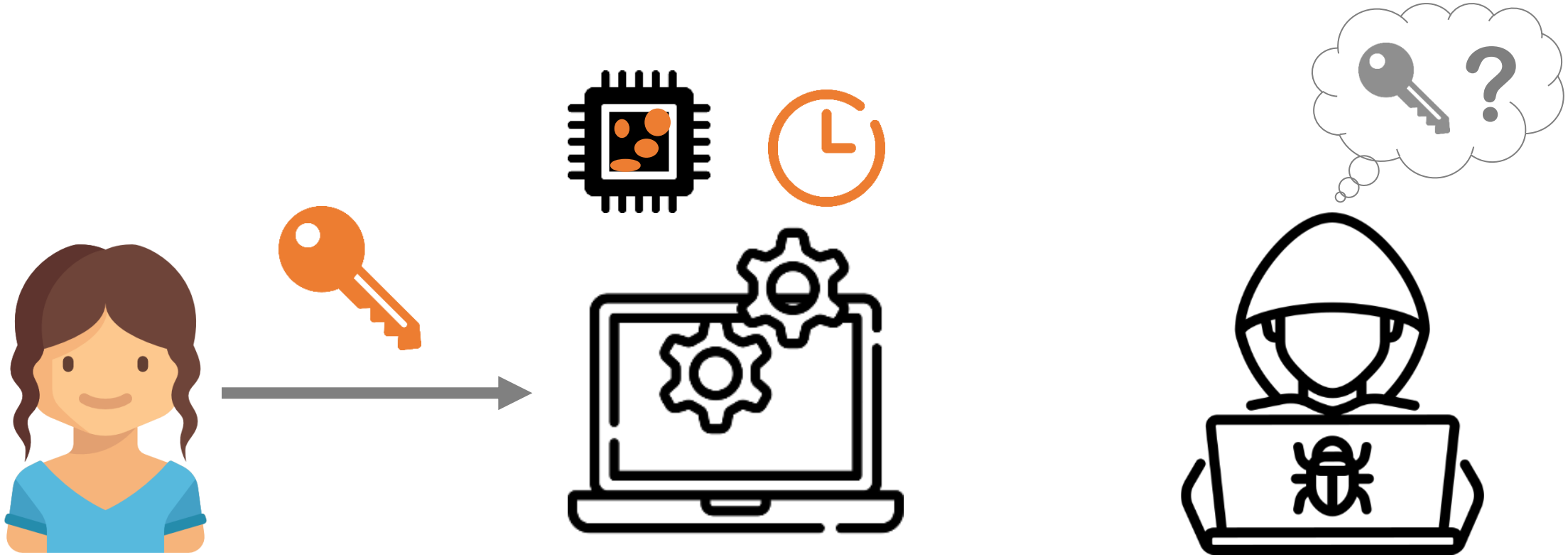
Validate RISC-V implementation

- Contract-based CPU testing (e.g., Revizor, Scam-V)?
- Hardware-fuzzing / Model checking / Formal methods?

I want to protect my secrets



Computations have physical side effects



Computations have physical side effects