Finding timing leaks
in Java Card implementations when the clock is hidden



OpenAlt 2026

Veronika Hanulíková
hanulikova@mail.muni.cz

CAT CROCS
Centre for Research on Cryptography and Security

Outline

- Why should we care about side channels?
- Constrained secure platforms and side channels
- How to build a time profiler for a smart card
- How to find inputs for profiling

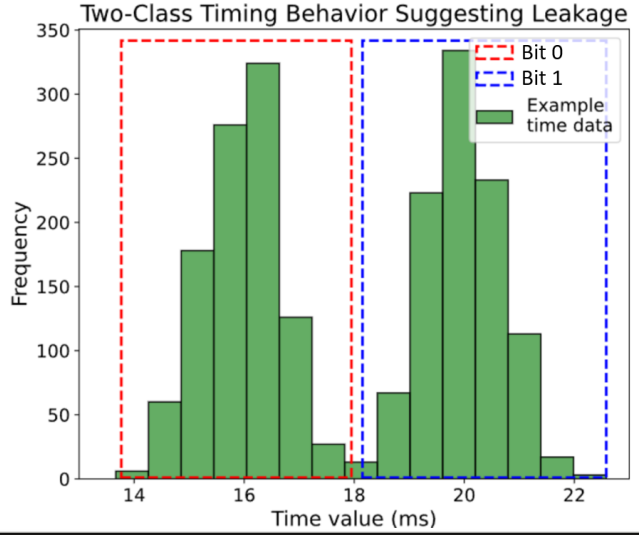
Why Should Engineers Care About Side Channels

Theoretical security algorithm guarantees

vs

Implementation on hardware

- Information leakage through:
  - Execution time
  - Power consumption
  - Electromagnetic radiation
- First timing attacks in 1996 by Kocher



Constrained Secure Platforms

Isolated environment

- Tamper-resistant
- Resource or permission limits
- Cryptographic operations executed on the platform

Our case: **Java Card platform**

- Java on a smart card
- Portability, security isolation
- Custom implementations



The Measurement Problem

in Constrained Environments & Smart Cards

- Custom implementations
  - Prone to side-channel leakage
  - Detecting leakage

    is **difficult**

Noise

Restricted runtime environment

No internal timing triggers

Variable-time API

Challenges to Solve

Precise measurement

How can we precisely measure **code performance** in Java Card applets, without internal support from the platform?

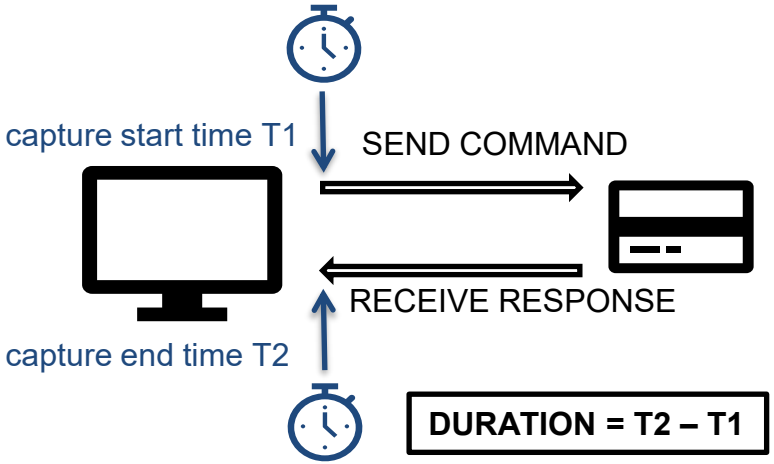
Automated input generation

How can **input data** be **automatically** generated for systematic **time-leakage** assessment of Java Card applets?

Challenge 1: **Building a Time Profiler**
for Constrained Platform

Solution 1: End-to-end latency measurement

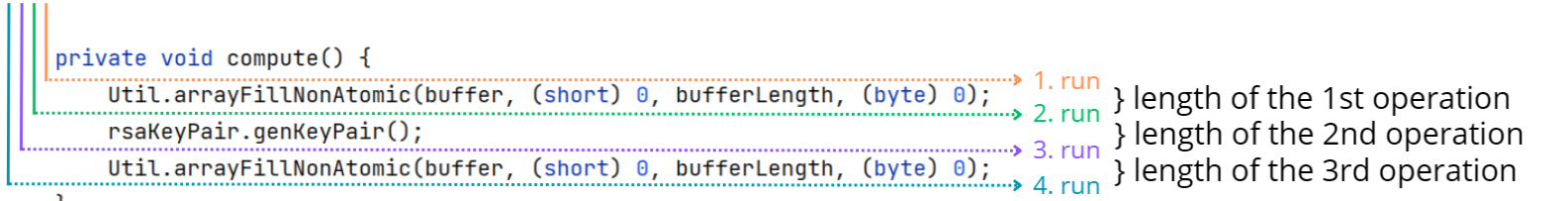
- Total round-trip duration between sending the command and receiving the response
- Suitable for production binaries
- Coarse-grained visibility only
  - Noise, runtime services
  - Masking line-level code behavior



Challenge 1: **Building a Time Profiler**
for Constrained Platform but Better

Solution 2: Exception-based profiling

- Java Card exception handling
  - Operation terminates, applet returns error code
- Automatically insert exceptions between target method code lines
- JCProfilerNext tool (github.com/crocs-muni/JCProfilerNext)



Source: Hanulíková, V. (2025). *Improving side-channel resistance of Java Card implementations.*

The Downfall of Exception Profiling

1. **Excessive method execution**
   - For each exception
2. **Variable-time operations**
   - Non-identical time between consecutive runs
   - E.g, generation of RSA keys
3. **Noise**
   - In-between method executions
   - Execution handling inside JCVM

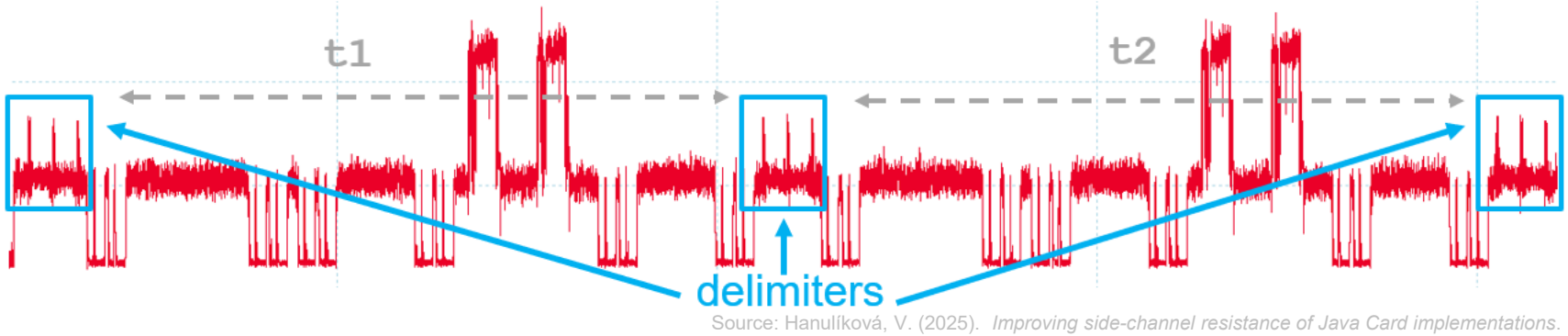
Challenge 1: **Building a Time Profiler**
for Constrained Platform but Better

Solution 3: Simple power analysis (SPA)

- Traditional SPA to extract secret bits from the power trace
- Power trace patterns → high-precision temporal timestamps

Challenge 1: **Building a Time Profiler**
for Constrained Platform but Better

Solution 3: Simple power analysis (SPA)



Source: Hanulíková, V. (2025). *Improving side-channel resistance of Java Card implementations.*

Insert delimiters between code lines → Measure power consumption trace → Compute time in between delimiters

Challenge 1: **Building a Time Profiler**
for Constrained Platform but Better

Solution 3: Simple power analysis (SPA)

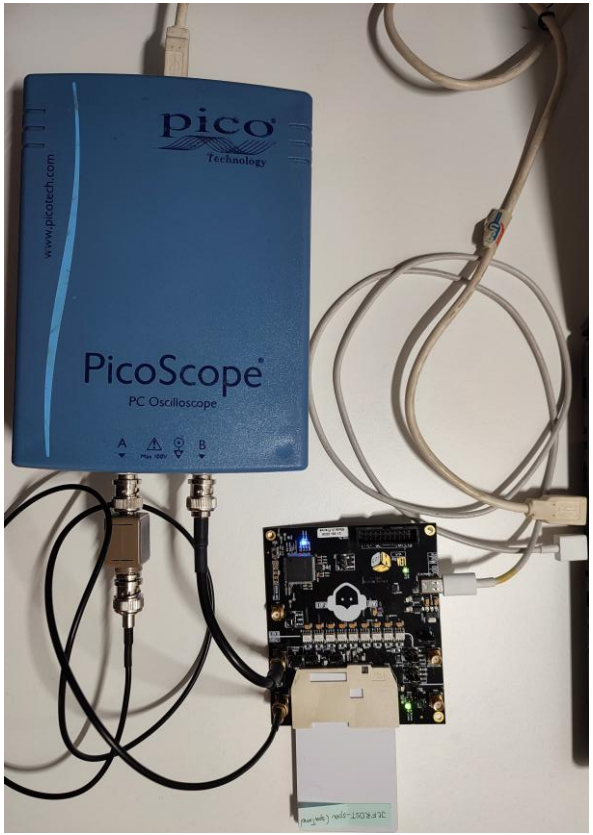
- Single execution run to acquire trace
  - No excessive method execution
  - Variable-time operations have no influence
  - Reduced noise

Challenge 1 Solved:

SPA-Based Time Profiler

Practical implementation in **JCProfilerNext**:

- https://github.com/crocs-muni/JCProfilerNext
- LEIA measurement board
- PicoScope oscilloscope
- Similarity search for delimiters in trace



Source: Hanulíková, V. (2025). *Improving side-channel resistance of Java Card implementations.*

Leakage Measurement Example

```
if (condition) {
        // body
}
```



```
if (condition) {
        // if body
} else {
        // else body
}
```

```
if<cond> LabelElse // jump to else if false
        // if-branch computation
goto LabelEnd // skip else
LabelElse:
        // else-branch computation
LabelEnd:
        // code after if-else
```

Leakage Measurement Example

```
if (condition) {
        // body
}
```



```
if (condition) {
        // if body
} else {
        // else body
}
```

false:

```
if<cond> LabelElse // jump to else if false
        // if-branch computation
goto LabelEnd // skip else
LabelElse:
        // else-branch computation
LabelEnd:
        // code after if-else
```

Leakage Measurement Example

```
if (condition) {
        // body
}
```



```
if (condition) {
        // if body
} else {
        // else body
}
```

true:

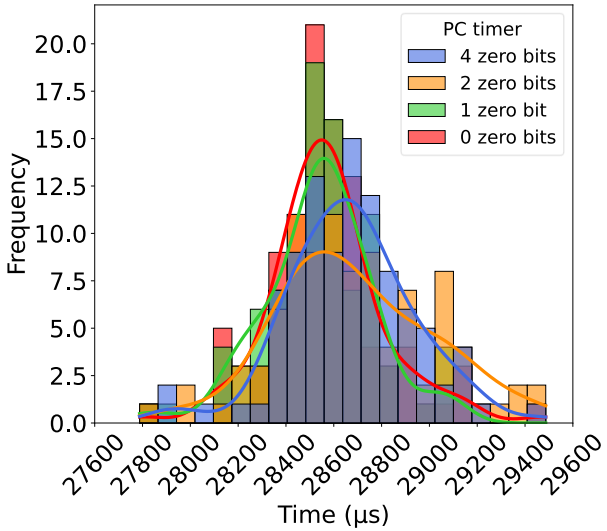


```
if<cond> LabelElse // jump to else if false
        // if-branch computation
goto LabelEnd // skip else
LabelElse:
        // else-branch computation
LabelEnd:
        // code after if-else
```

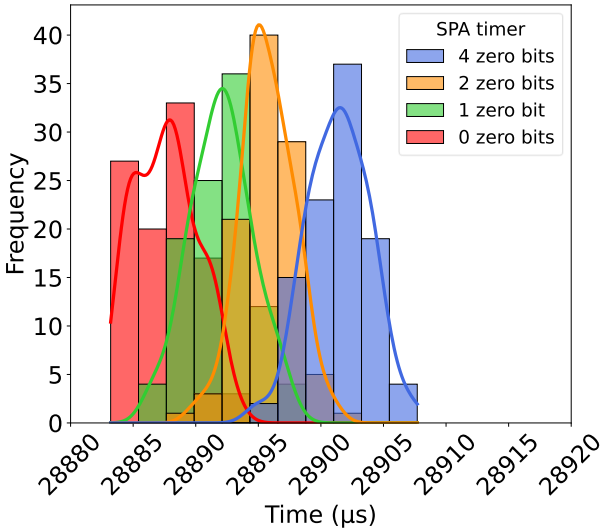
Leakage Measurement Example

```
if (bit == 0) {
        r += 1
} else {
        tmp += 1 //dummy
}
```

Exception-based profiling



SPA-based profiling



Source: Hanulíková, V. (2025). *Improving side-channel resistance of Java Card implementations.*

Challenge 2: **Discovering Problematic Inputs**

Idea 1

Exhaustive testing

- Large data space
- Computationally infeasible

Idea 2

Random testing

- Blind to structure
- Miss edge cases

Idea 3

Fuzzing

- Guided input exploration
- Path divergence

Challenge 2: **Discovering Problematic Inputs**
via Differential Fuzzing

Flow:

1. Execute the target code with two distinct inputs
2. Measure the difference in executed paths or resource costs
3. Generate inputs to maximize the difference in paths

Implementation:

- DifFuzz framework
- American Fuzzy Lop (AFL) fuzzer

Challenge 2: **Discovering Problematic Inputs**

via Differential Fuzzing for the Java Card Platform

- Fuzzing requires bytecode instrumentation
- A physical card cannot run such code
- Use the Java Card Development Kit Simulator on a host PC



Challenge 2 Solved:

End-to-End Analysis Pipeline

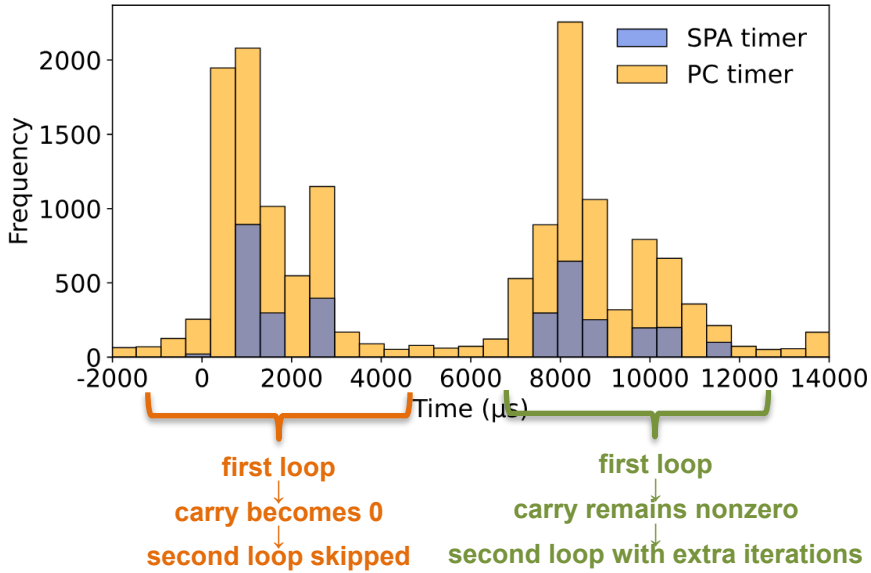
1. **Differential fuzzing with a simulator**
   - Efficient exploration without repeated card access
2. **SPA profiling on a physical card**
   - Validates timing differences on real hardware
3. **Statistical evaluation**
   - Hypothesis testing

Testing Open-Source Applets & Beyond

JCMathLib Library (https://github.com/OpenCryptoProject/JCMathLib)

- Mathematical library for big numbers and elliptic curve points

```
private byte add(BigNatInternal other, short shift) {
    short i = (short) (other.size - 1 + other.offset);
    short j = (short) (size - 1 - shift + offset);
    for (; i ⩾ other.offset && j ⩾ offset; i--, j--) {
        // Add the corresponding digits,
         / store result digit and update carry.
    }
    for (; acc > 0 && j ⩾ offset; --j) {
        // Propagate remaining carry
    }
    // Output carry bit if present
}
```



Source: Hanulíková, V. (2025). *Improving side-channel resistance of Java Card implementations.*

Conclusions

1. Timing side channels remain a real risk in Java Card applets
2. SPA-based profiling provides high precision
3. Differential fuzzing enables automatic input discovery

CAT

Co-funded by the European Union

The project funded under Grant Agreement No. 101225878 is supported by the European Cybersecurity Competence Centre.

ECCC EUROPEAN CYBERSECURITY COMPETENCE CENTRE

Questions & Discussion

hanulikova@mail.muni.cz

JCProfilerNext

github.com/crocs-muni/JCProfilerNext

SCRUTINY toolkit

github.com/crocs-muni/scrutiny

Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Cybersecurity Competence Centre. Neither the European Union nor the European Cybersecurity Competence Centre can be held responsible for them.