A Tool for Static Detection of Timing Channels in Java

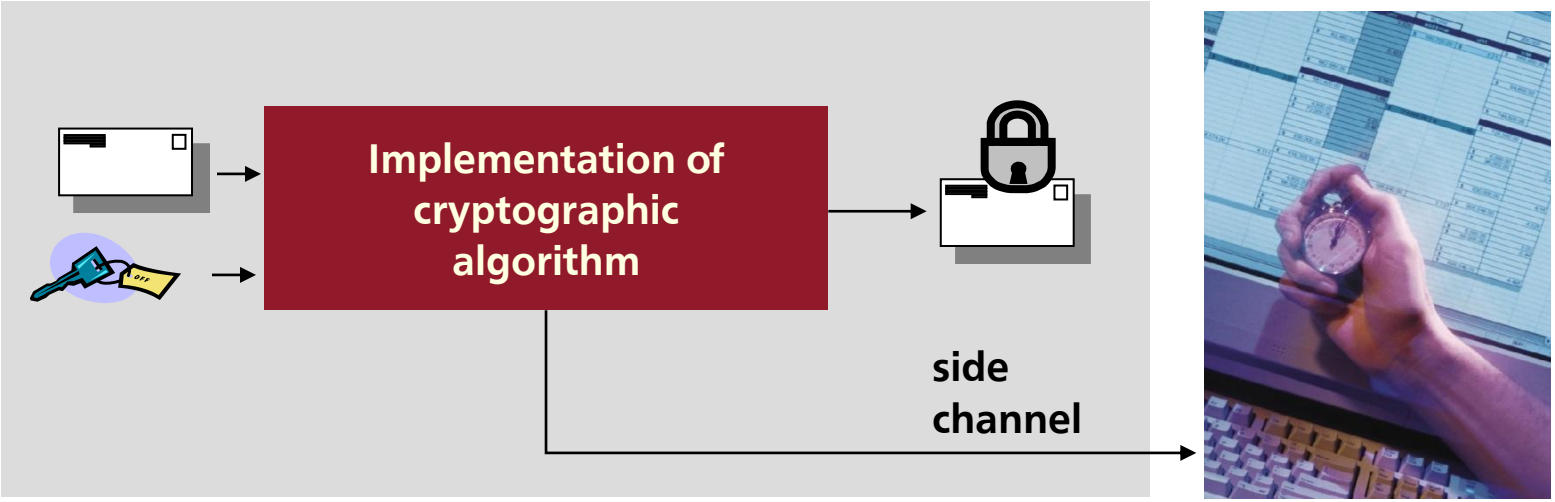
Alexander Lux, Artem Starostin, et. al.

Modeling and Analysis of Information Systems (MAIS)
Computer Science, TU Darmstadt

COSADE 2011, February 25th, Darmstadt

CASED

Timing Channels



Well-founded, automatic identification of timing channels

- in Java-programs
- by static program analysis

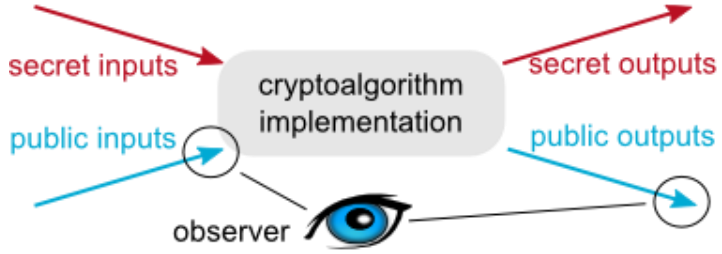
Application on Cryptographic Library

Source: Alexander Lux, TU Darmstadt

Security: Information Flow

No insecure information flow:

attacker learns nothing about secret information by his observations



How can we be sure that a given piece of software is trustworthy enough to access confidential data?

Also in the presence of timing channels?

Source: Alexander Lux, TU Darmstadt

Security Type Systems

Data type systems: check that data is used as intended

- check correct transfer of data
- check whether operations are applied to correct type

Security Type Systems:

- check correct transfer of data

```
output = secretkey
```

- check that no implicit flow can occur

```
keypart = secretkey[i];
if (keypart == 0)
  { output := 0; }
else
  { output := 1; }
```

Source: Alexander Lux, TU Darmstadt

Causes of Timing Channels

In the literature: branching on confidential information

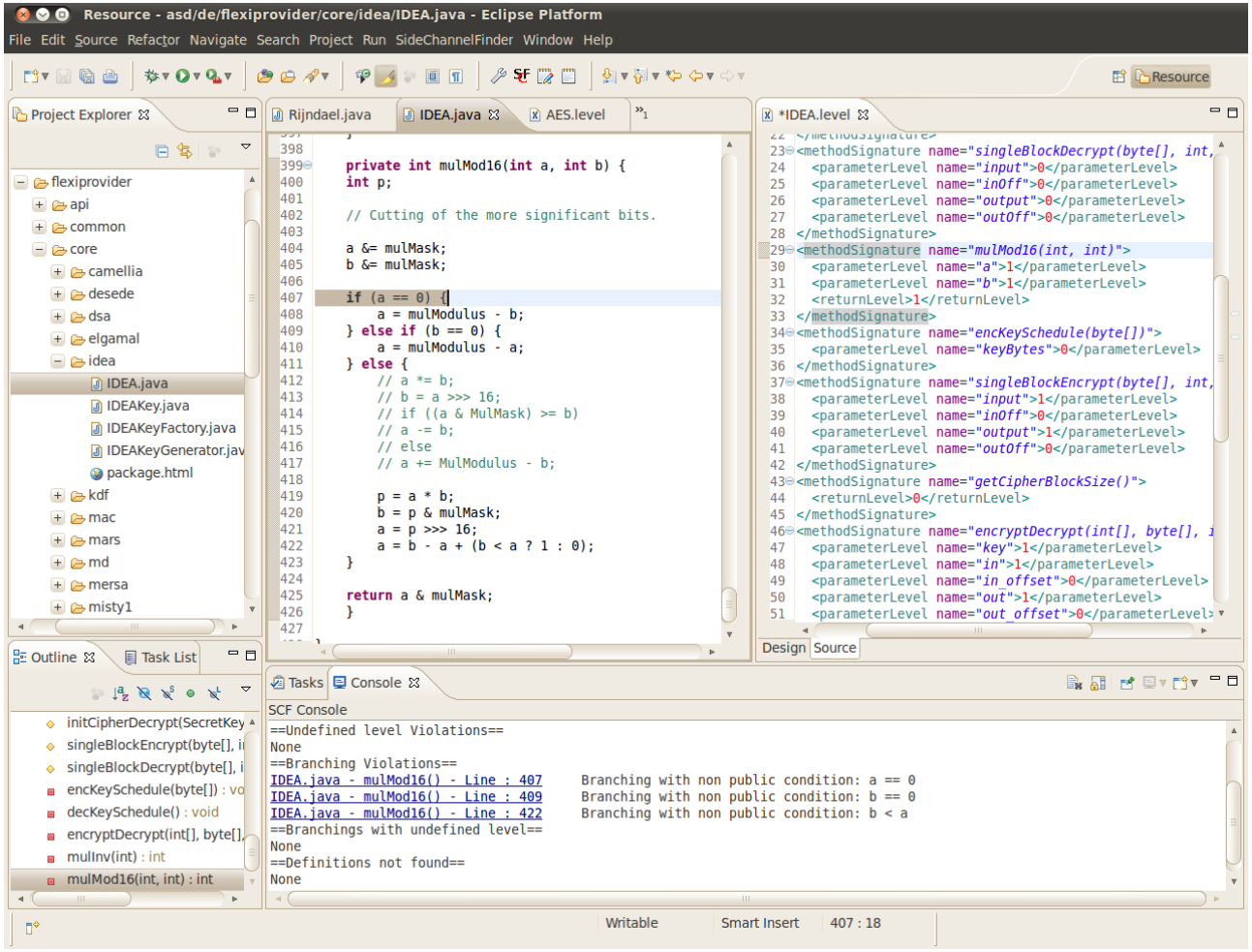
- extra reduction step in Montgomery multiplication
- data-dependend choice of multiplication algorithm (Karatsuba)
- extra reduction step in Blakley‘s modular multiplication
- special treatment of value 0 in IDEA
- loop-conditions in McEliece

(Disclamer: not considering microarchitectural features)

Information flow control in principle can detect such vulnerabilities.

Source: Alexander Lux, TU Darmstadt

Tool: Side Channel Finder



Source: Alexander Lux, TU Darmstadt

Language-Coverage

Propagation of information

- assignment to local variable: `v=exp`
- assignment to fields: `v.f=exp`
- parameter passing: `v.m(exp)`
  - (by value and by reference)
  - for all possible call targets (inheritence and polymorphism)
- value return: `return exp`

Control Flow

- conditional branching:

```
if(v==0) then { do something }
 else { do something different}
```

- similarly: for- and while-loops
- polymorphism (where called class is confidential): `v.m(exp)`

Source: Alexander Lux, TU Darmstadt

Policy

Purpose

- specify secrets
- provide guidance

Security Levels

- high (1) for secret, low (0) for publi
- level for each field, method parameter, return value

Automation

- automatic generation of default policy
- no levels for local variables necessary

Source: Alexander Lux, TU Darmstadt

Type System

$$P, c, MS, fs \vdash exp : ls \rightarrow ls', l$$

$$P, c, MS, fs, l_{\text{ret}} \vdash stm : ls \rightarrow ls'$$

$$P, c, MS, fs \vdash md : ms$$

$$[\text{ExpStmt}]\ \frac{\vdash exp : ls \rightarrow ls', l}{\vdash exp; : ls \rightarrow ls'}$$

$$[\text{ReturnStmt1}]\ \frac{\vdash exp : ls \rightarrow ls', l}{\vdash \text{return } exp; : ls \rightarrow ls'}\ l \leq l_{\text{ret}} \qquad [\text{ReturnStmt2}]\ \frac{}{\vdash \text{return;} : ls \rightarrow ls}$$

$$[\text{IfStmt1}]\ \frac{\vdash exp : ls \rightarrow ls', low \quad \vdash stm_1 : ls' \rightarrow ls'' \quad \vdash stm_2 : ls' \rightarrow ls''}{\vdash \text{if } (exp)\ \{stm_1\}\ \text{else}\ \{stm_2\} : ls \rightarrow ls''}$$

$$[\text{IfStmt2}]\ \frac{\vdash exp : ls \rightarrow ls', low \quad \vdash stm : ls' \rightarrow ls'}{\vdash \text{if } (exp)\ \{stm\} : ls \rightarrow ls'}$$

$$[\text{WhileStmt}]\ \frac{\vdash exp : ls \rightarrow ls, low \quad \vdash stm : ls \rightarrow ls}{\vdash \text{while } (exp)\ \{stm\} : ls \rightarrow ls}$$

Source: Alexander Lux, TU Darmstadt

Exemplary Analysis

Target:

- block algorithm IDEA of FlexiProvider 1.6p9
- method to be analyzed:

```
singleBlockEncrypt(byte[] input, int inOff,
        byte[] output, int outOff)
```

- secret key in field `encr`

Result of analysis:

```
Violations:

de.flexiprovider.core.idea.mulMod16(int, int)[(1,1)] in line 407;

Branching with non public condition a==0
```

Source: Alexander Lux, TU Darmstadt

Experimential Evaluation of Found Timing Channel

Implemented step of attack from Kelsey et al. 2000

No.	key bits pos. 70–85	total avg. time	max. avg. time of cluster	16 bits (A) with max time	est. key bits $(2^{16}+1)-A$)	key bits match
1	0xea23	2960 ns	3002 ns	0x15de	0xea23	✓
2	0x50a2	2934 ns	2991 ns	0xaf5f	0x50a2	✓
3	0x0855	2893 ns	2917 ns	0x8894	0x776d	×
4	0x87d3	2888 ns	2929 ns	0x782e	0x87d3	✓
5	0x6030	2907 ns	2992 ns	0x9fd1	0x6030	✓
6	0xb46a	2900 ns	2943 ns	0x4b97	0xb46a	✓
7	0xc08c	2923 ns	2965 ns	0x3f75	0xc08c	✓
8	0xe1fc	2883 ns	2978 ns	0x1e05	0xe1fc	✓
9	0x93ef	2908 ns	2958 ns	0x6c12	0x93ef	✓
10	0x6786	2944 ns	2966 ns	0x987b	0x6786	✓

Source: Alexander Lux, TU Darmstadt

Conclusion

Summery

- tool for static detection of timing channels in Java programs
- covers non-trivial subset of Java (objects, loops, methods,…)
- analyzed several algorithms
- found actual timing channel

Future Work

- automatic transformation
- increase coverage of language features
- analyze further algorithms

Source: Alexander Lux, TU Darmstadt