

Formal verification of a JavaCard Virtual Machine for Common Criteria Certification

Previously presented at FM'21 and ERTS'22

Nikolai KOSMATOV

Joint work with Adel DJOUDI, Martin HANA

“Principles of Contract Languages”

Dagstuhl seminar 22451, November 10, 2022




Introduction

General approach

Proof issues and solutions

Results and conclusion


Context: three fields of expertise



ORACLE[®]
Java Card

Standard
Specification

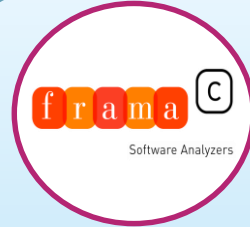
Java Card Virtual Machine
JCVM



COMMON CRITERIA

Security Assurance
Requirements

Evaluation Assurance Level
EAL6-EAL7



frama^C
Software Analyzers

Formal
Verification

WP: Deductive verification
C code



THALES
Building a future we can all trust

- **C** implementation of the **Standard Specification** of the **JCVM**
- **Formal Security Properties** meet **Security Assurance Requirements**
- **Formal verification** of global formal security properties using **Frama-C/WP**

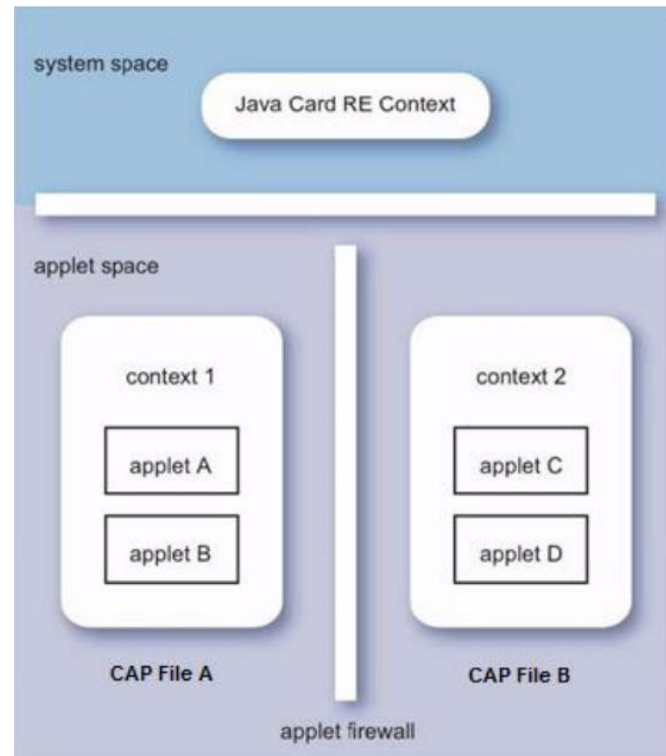


- Execute Java Card applications' bytecode with **basic operations**
- Bytecodes are read iteratively inside the main **dispatch loop**

- 3 main memory areas: Java **stack**, data **heap** and **code** area
- 3 types of **heap** memory: **persistent**, **transient** **reset/deselect**

- A **unique context** assigned to each Java Card binary (CAP file)
- **Object owner context** is stored inside the object header

- The **Firewall guarantees isolation** of heap data between different contexts
- Java Card Runtime Environment (**JCRE**) context is a **privileged context** devoted to system operations
- **Well-defined exceptions:** global arrays, shareable interfaces,...



EAL6-EAL7: Formal verification of Security Properties



Security Aspect

#.Firewall: “The Firewall shall ensure controlled sharing of class instances, and **isolation of their data and code between packages** (that is, controlled execution contexts) as well as between packages and the JCRE context...”

[Java Card System – Open Configuration Protection Profile – V3.1]

Security properties (simplified examples)

- **integrity_header**: allocated objects' headers cannot be **modified** during a VM run.
- **integrity_data**: allocated objects' data can be **modified** only by the owner.
- **confidentiality_data**: allocated objects' data can be **read** only by the owner.

Evaluation Assurance Levels

EAL1	EAL2	EAL3	EAL4	EAL5	EAL6	EAL7
------	------	------	------	------	------	------

Formal verification

Formal verification of security properties

Overview

■ Introduction

■ General approach

■ Proof issues and solutions

■ Results and conclusion

OPEN

THALES

```
/*@
requires P;
assigns L;
ensures E;
*/
<type> function (<type> arg1,<type> arg2, ...) {
    ...

    /*@
    loop invariant I;
    loop assigns L;
    loop variant m;
    */
    while (c) {
        ...
    }

    ...
}
```

Formal Specification Structure

Basic level

STEP1: Write ACSL annotations
(Formal Specification)

STEP2: Frama-C/WP computes proof goals
(Based on Hoare logic)

STEP3: Discharge proof goals with
(QED, Alt-Ergo via Why3, ...)

Advanced level features

Ghost code

Predicates, Lemmas

Proof scripts

■ Integrity_data and Confidentiality_data **cannot be specified (easily)** with WP as global invariants

■ We use metaproperties:

name

targets all function(s)

application context:
whenever a location is read

```
meta \prop, \name(meta_persi_objects_confident), \targets(\ALL), \context(\reading),  
( \forall integer i; 0 <= i < gNumObjs && !gIsTrans[i] &&  
  ObjHeader[gHeadStart[i] + 0] != JCC ==>  
  \separated(\read, PersiData+(gDataStart[i]..gDataEnd[i])) ); */
```

The read location must be separated from the data of any persistent object if the current context is not its owner.

- **MetAcsI** translates metaproperties into **assertions/checks** at each relevant program point.
- If all **assertions/checks** are proved, the metaproperty is proved.
- Thanks to the translation of metaproperties into **checks** that do not overload proof contexts, the metaproperty-based approach scales very well, despite a great number of generated annotations.

Example: Integrity Metaproperty Verified with MetAcsI

Resulting code after generating assertions with MetAcsI and proof with Frama-C/WP:

```
/*@ meta "A_unchanged_unless";
*/
/*@ requires A == B;
    ensures
        (C ≥ 0 ∧ A == C ∧ B == C) ∨
        (C < 0 ∧ A == \old(A) ∧ B == \old(B));
    assigns A, B;
*/
void foo(void)
{
    if (C ≥ 0) {
        /*@ check A_unchanged_unless: _1: meta: C < 0 → \separated(&A, &A);
            A = C;
        /*@ check A_unchanged_unless: _2: meta: C < 0 → \separated(&B, &A);
            B = C;
        }
    }
    return;
}
```

MetAcsI

Initial C code:

```
test5.c
1 int A, B, C;
2 /*@
3   meta \prop, \name(A_unchanged_unless),
4     \targets(\ALL), \context(\writing),
5     C < 0 ==> \separated(\written, &A);
6 */
7 /*@
8   requires A==B;
9   assigns A,B;
10  ensures C>=0 && A==C && B==C ||
11    C<0 && A==\old(A) && B==\old(B); */
12 void foo(){
13     if ( C ≥ 0 ){
14         A = C;
15         B = C;
16     }
17 }
18
```

■ Introduction

■ General approach

■ Proof issues and solutions

■ Results and conclusion

Some Issues (I) and Solutions (S)

Companion ghost model

- I: **Automatic proof fails** on low-level code (bit-fields)
- S: Linking bits to ghost integer variables brings the **prover back into its comfort zone**



Proof scripts for complex predicates

- I: **Automatic proof fails** to use the right predicates
- S: **Guide the first proof steps** by unfolding relevant predicates or instantiating values



Carefully chosen lemmas

- I: **Automatic proof fails** repeatedly in similar cases
- S: Lemmas help to **re-use the same reasoning**



■ Introduction

■ General approach

■ Proof issues and solutions

■ Results and conclusion

Specification effort for EAL6

JCVM C code		ACSL Annotations			
		User provided annotations		MetAcsl	RTE
# Functions	# Loc C	# Loc Ghost	# Loc ACSL	# Loc ACSL	# Loc ACSL
381	7,014	162	35,480	396,603	2,290

Large code

A few yet necessary

12,432 before preprocessing macros that
gather redundant annotations
Still a considerable effort

Automatically generated from 36
metaproperties only

- **User-provided annotations:** predicates, lemmas, function contracts, loop contracts and other assertions
- **MetAcsl:** automatically generated annotations according to user-defined metaproperties
- **RTE:** automatically generated annotations in order to prevent undefined behaviors

■ Successful industrial application of deductive verification

- World-first proof of real-life JavaCard VM code
- EAL7 certificate issued by ANSSI
- Careful combination of: ghost code, lemmas, proof scripts, ...
- High level of automation (99% of goals proved automatically)
- MetAcsI is crucial for specification of security properties
- Efficient tool support from Frama-C developers was essential

Ongoing and future work directions

- Introduce proof into a continuous integration process
- Custom and more flexible proof strategies to save manual script effort
- Scaling to large programs having parts with and without low-level operations, or where some of the maintained properties are irrelevant
 - Collaborative memory models
 - More abstract levels of reasoning
- Participate in collaborative (e.g. EU) projects to develop / apply innovative verification techniques to Thales products

References

- Adel Djoudi, Martin Hana and Nikolai Kosmatov.
“Formal verification of a JavaCard virtual machine with Frama-C”. **FM 2021**, Springer.
- Adel Djoudi, Martin Hana, Nikolai Kosmatov, Milan Kříženecký, Franck Ohayon, Patricia Mouy, Arnaud Fontaine and David Féliot.
“A Bottom-Up Formal Verification Approach for Common Criteria Certification: Application to JavaCard Virtual Machine”. **ERTS 2022, Best paper award**.
- Virgile Robles, Nikolai Kosmatov, Virgile Prevosto, Louis Rilling, and Pascale Le Gall.
“MetAcsl: Specification and Verification of High-Level Properties.” **TACAS 2019**, Springer.
- Virgile Robles, Nikolai Kosmatov, Virgile Prevosto, Louis Rilling, and Pascale Le Gall.
“Tame your annotations with MetAcsl: Specifying, Testing and Proving High-Level Properties”. **TAP 2019**, Springer.
- Virgile Robles, Nikolai Kosmatov, Virgile Prevosto, Louis Rilling, and Pascale Le Gall.
“Methodology for Specification and Verification of High-Level Properties with MetAcsl”. **FormaliSE 2021**, IEEE.