Implementing Selective Disclosure Protocols on Java Cards. A first experience report.

Hendrik Tews Bart Jacobs

Radboud Universiteit Nijmegen
The Netherlands

WISTP, September 3, 2009

- Hendrik left Nijmegen in June 2009 (and is on *Elterngeld* now).
- Wojciech Mostowski and Pim Vullers will continue the work.
- Sponsored by the NLnet foundation through the OV-chipkaart project.

Implementing Selective Disclosure Protocols on Java Cards. A first experience report.

Hendrik Tews Bart Jacobs

Radboud Universiteit Nijmegen
The Netherlands

WISTP, September 3, 2009

- Hendrik left Nijmegen in June 2009 (and is on *Elterngeld* now).
- Wojciech Mostowski and Pim Vullers will continue the work.
- Sponsored by the NLnet foundation through the OV-chipkaart project.

Outline

Introduction

Performance critical part: Multi-powers on Java Card

OV-Chip 2.0 Applet: Structure and Performance

Conclusion

Identity-based Authentication

For example with the Mifare Classic chip card

- used in
  - the Dutch OV-chipkaart
  - London's Oyster card
  - Hong Kong's Octopus card
  - access control at Radboud Universiteit
  - ...
- fixed anti-collision UID to facilitate tracing people on the street
- fixed application level identity to facilitate tracing customers

Identity-based Authentication

For example with the Mifare Classic chip card

- used in
  - the Dutch OV-chipkaart
  - London's Oyster card
  - Hong Kong's Octopus card
  - access control at Radboud Universiteit
  - ...
- fixed anti-collision UID to facilitate tracing people on the street
- fixed application level identity to facilitate tracing customers

The OV-chip 2.0 project

Design and (partially) implement a privacy friendly alternative for the Dutch OV-chipkaart

- single, nationwide chip card for all public transport in the Netherlands
- built-in privacy: nobody can generate traces of customers or even identify customers
- implementation on Java Card with a free license

Until early 2009

- focus on **selective disclosure** and **blinded issuing** by Stefan Brands

Now and near future

- develop new protocols based on elliptic curve pairing

Privacy friendly authentication

Protocol for Selective Disclosure

- customer possesses attributes
  (password, private key, age, permission to board a train, . . .)
- proves knowledge/possession of his attributes
- **without** disclosing some (all) attributes

Protocol for Blinded Issuing

Generate signatures such that the signing authority

- **does not** know what is signed
  (not requiring the disclosure of attributes when obtaining a signature)
- **does not** learn the resulting signature
  (permitting the customer to stay anonymous after getting a new signature)

Stefan Brands. *Rethinking Public Key Infrastructures and Digital Certificates: Building in Privacy*. MIT Press, 2000. See www.credentica.com.

Privacy friendly authentication

Protocol for Selective Disclosure

- customer possesses attributes
  (password, private key, age, permission to board a train, ...)
- proves knowledge/possession of his attributes
- **without** disclosing some (all) attributes

Protocol for Blinded Issuing

Generate signatures such that the signing authority

- **does not** know what is signed
  (not requiring the disclosure of attributes when obtaining a signature)
- **does not** learn the resulting signature
  (permitting the customer to stay anonymous after getting a new signature)

Stefan Brands. *Rethinking Public Key Infrastructures and Digital Certificates: Building in Privacy*. MIT Press, 2000. See www.credentica.com.

Privacy friendly authentication

Protocol for Selective Disclosure

- customer possesses attributes
  (password, private key, age, permission to board a train, ...)
- proves knowledge/possession of his attributes
- **without** disclosing some (all) attributes

Protocol for Blinded Issuing

Generate signatures such that the signing authority

- **does not** know what is signed
  (not requiring the disclosure of attributes when obtaining a signature)
- **does not** learn the resulting signature
  (permitting the customer to stay anonymous after getting a new signature)

Stefan Brands. *Rethinking Public Key Infrastructures and Digital Certificates: Building in Privacy*. MIT Press, 2000. See www.credentica.com.

Outline

Introduction

Performance critical part: Multi-powers on Java Card

OV-Chip 2.0 Applet: Structure and Performance

Conclusion

Protocol Example

(here for proving knowledge of two attributes only)

System setup (glossing over side conditions)

$\mathbf{n} = \mathbf{pq}$ RSA modulus (about 1280 bits)

$\mathbf{g_1, g_2} < \mathbf{n}$ public bases

$\mathbf{v}$ public RSA exponent (about 160 bits)

Card setup

$\mathbf{a_1, a_2} < \mathbf{v}$ attributes of the card

$\mathbf{b} < \mathbf{n}$ blinding

$\mathbf{A} = (\mathbf{b^v\, g_1^{a_1}\, g_2^{a_2}}) \mod \mathbf{n}$ blinded attribute expression

Protocol Example (cont.)

Card proves knowledge of a_1 and a_2 to a **G**ate

Card Commitment

$$\mathrm{C} \longrightarrow \mathrm{G} \;:\; A, w \quad \text{where } w = \left(\beta^v \, g_1^{\alpha_1} \, g_2^{\alpha_2}\right) \bmod n \text{ for random } \beta, \alpha_i$$

Gate Challenge

$$\mathrm{C} \longleftarrow \mathrm{G} \;:\; \gamma < v \quad \text{random}$$

Card Response

$$\mathrm{C} \longrightarrow \mathrm{G} \;:\; r_1, r_2, s \quad \text{where} \begin{cases} r_i = (\gamma a_i + \alpha_i) \bmod v \\ q_i = (\gamma a_i + \alpha_i) \div v \\ s = \left(\beta \, b^\gamma \, g_1^{q_1} \, g_2^{q_2}\right) \bmod n \end{cases}$$

Acceptance Check

Gate accepts the prove if $\quad s^v \, g_1^{r_1} \, g_2^{r_2} = A^\gamma \, w \pmod{n}$

Stefan Brands. *Rethinking Public Key Infrastructures and Digital Certificates: Building in Privacy*. MIT Press, 2000. See www.credentica.com.

Performance critical part: Multi-Powers on the card

Modular Multi-Powers

- $(g_1^{a_1}\, g_2^{a_2}\, \cdots\, g_n^{a_n}) \mod n$
- bases of 1300–2000 bits, exponents of 160–200 bits

Performance measurements for

- 4 attributes + blinding
- short term security: 1300 bit bases/modulus, 160 bit exponents
- SUN Java JRE 1.6
- Intel Core Duo 1.66 GHz

SUN BigInteger	0.0395 seconds	single modular powers
our Bignat	0.0264 seconds	simultaneous squaring
Java Card		estimation for pure Java Card
Crypto Coprocessor		potentially

Performance critical part: Multi-Powers on the card

Modular Multi-Powers

- $(g_1^{a_1}\, g_2^{a_2} \cdots g_n^{a_n}) \mod n$
- bases of 1300–2000 bits, exponents of 160–200 bits

Performance measurements for

- 4 attributes + blinding
- short term security: 1300 bit bases/modulus, 160 bit exponents
- SUN Java JRE 1.6
- Intel Core Duo 1.66 GHz

SUN BigInteger	0.0395 seconds	single modular powers
our Bignat	0.0264 seconds	simultaneous squaring
Java Card		estimation for pure Java Card
Crypto Coprocessor		potentially

Performance critical part: Multi-Powers on the card

Modular Multi-Powers

- $(g_1^{a_1} \, g_2^{a_2} \cdots g_n^{a_n}) \mod n$
- bases of 1300–2000 bits, exponents of 160–200 bits

Performance measurements for

- 4 attributes + blinding
- short term security: 1300 bit bases/modulus, 160 bit exponents
- SUN Java JRE 1.6
- Intel Core Duo 1.66 GHz

SUN BigInteger	0.0395 seconds	single modular powers
our Bignat	0.0264 seconds	simultaneous squaring
Java Card	2 hours 25 min	estimation for pure Java Card
Crypto Coprocessor		potentially

Performance critical part: Multi-Powers on the card

Modular Multi-Powers

- $(g_1^{a_1}\, g_2^{a_2} \cdots g_n^{a_n}) \mod n$
- bases of 1300–2000 bits, exponents of 160–200 bits

Performance measurements for

- 4 attributes + blinding
- short term security: 1300 bit bases/modulus, 160 bit exponents
- SUN Java JRE 1.6
- Intel Core Duo 1.66 GHz

SUN BigInteger	0.0395 seconds	single modular powers
our Bignat	0.0264 seconds	simultaneous squaring
Java Card	2 hours 25 min	estimation for pure Java Card
Crypto Coprocessor	0.1 seconds	potentially

Java Card potential

Cryptographic Coprocessor

- does modular multiplications for big integers in hardware
- used for high-level cryptographic operations in Java Card (RSA, DSA, . . .)
- accessible via `javacardx.framework.math.BigNumber` in Java Card 2.2.2
- estimated 0.3 *milliseconds* for a modular big integer multiplication

However . . .

Java Card Limitations

BigNumber class in Java Card 2.2.2

- only addition, subtraction, multiplication
- no division, no modular multiplication, no modular exponentation
- found only 2 cards on this planet that implement Java Card 2.2.2: Athena IDProtect and a new NXP card
- `BigNumber` is optional and not implemented on these cards

Java Card 2.2.1

- no `BigNumber` in Java Card 2.2.1
- trick a high-level crypto operation of the Java Card API into an arithmetic operation on big integers
- for security these crypto operation contain random padding, which cannot be controlled from the interface
- **only exception:** `ALG_RSA_NOPAD` **cipher computes cipher text** $g^a \mod n$ **for plain text** g **and key** n, a

Java Card Limitations

BigNumber class in Java Card 2.2.2

- only addition, subtraction, multiplication
- no division, no modular multiplication, no modular exponentation
- found only 2 cards on this planet that implement Java Card 2.2.2: Athena IDProtect and a new NXP card
- `BigNumber` is optional and not implemented on these cards

Java Card 2.2.1

- no `BigNumber` in Java Card 2.2.1
- trick a high-level crypto operation of the Java Card API into an arithmetic operation on big integers
- for security these crypto operation contain random padding, which cannot be controlled from the interface
- **only exception:** `ALG_RSA_NOPAD` **cipher computes cipher text** $g^a \mod n$ **for plain text** g **and key** n, a

Java Card Limitations

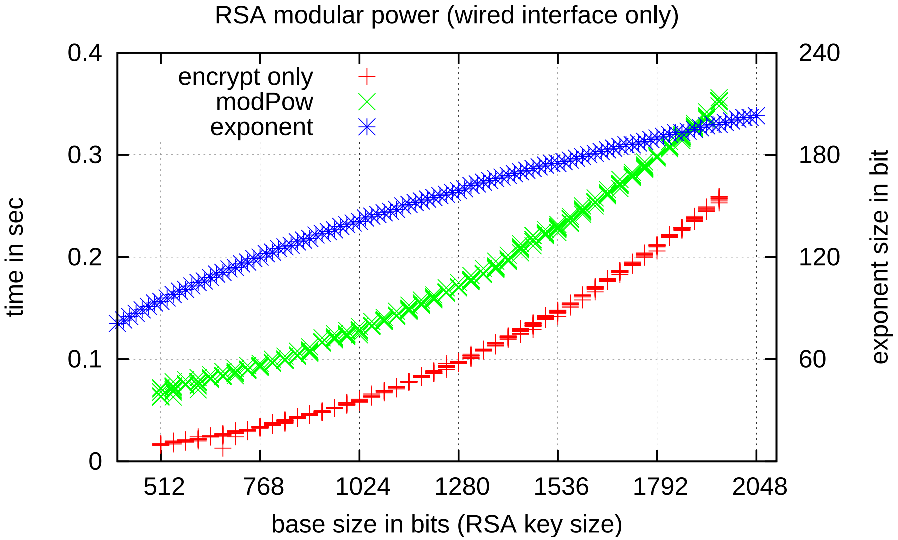
BigNumber class in Java Card 2.2.2

- only addition, subtraction, multiplication
- no division, no modular multiplication, no modular exponentation
- found only 2 cards on this planet that implement Java Card 2.2.2: Athena IDProtect and a new NXP card
- `BigNumber` is optional and not implemented on these cards

Java Card 2.2.1

- no `BigNumber` in Java Card 2.2.1
- trick a high-level crypto operation of the Java Card API into an arithmetic operation on big integers
- for security these crypto operation contain random padding, which cannot be controlled from the interface
- **only exception: `ALG_RSA_NOPAD` cipher computes cipher text $g^a \mod n$ for plain text g and key n, a**

Modular exponentiation on the crypto coprocessor



(for many more charts go to ovchip.cs.ru.nl/OV-chip_2.0_progress)

Modular Products?

$(a\,b) \mod n$

- no access to the cryptographic coprocessor for multiplication on currently available Java Cards
- Montgomery multiplication in Java takes 25 seconds for 1280 bit numbers

$(a+b)^2 = a^2 + 2ab + b^2$

- i.e., $ab = \frac{(a+b)^2 - a^2 - b^2}{2}$
- can be turned into a modular multiplication for odd moduli
- requires 2 subtractions, 1–4 additions, 1 right shift
- called *squaring multiplication*

Modular Products?

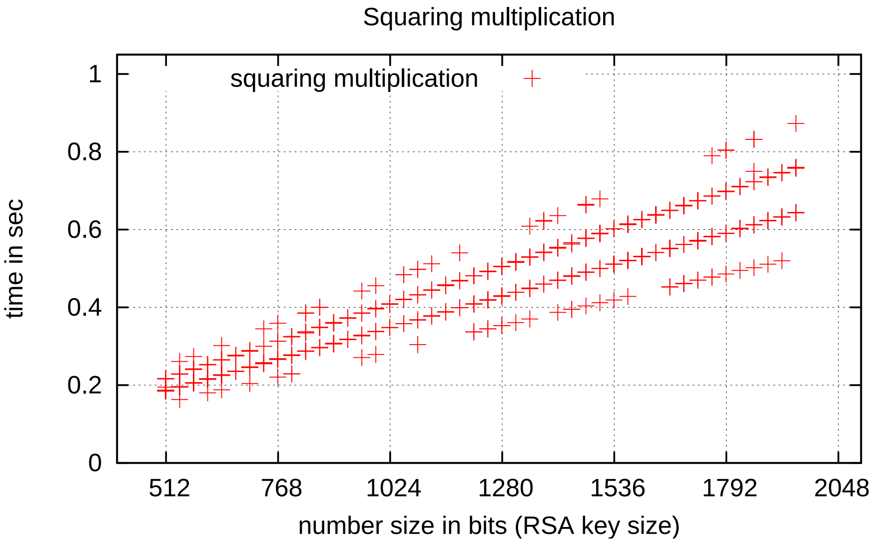
$(a\,b) \mod n$

- no access to the cryptographic coprocessor for multiplication on currently available Java Cards
- Montgomery multiplication in Java takes 25 seconds for 1280 bit numbers

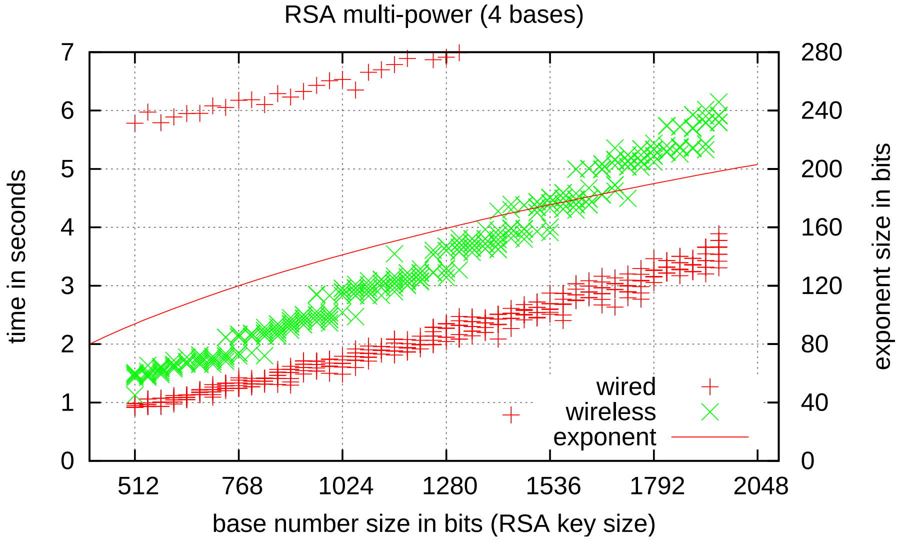
$(a+b)^2 = a^2 + 2ab + b^2$

- i.e., $ab = \frac{(a+b)^2 - a^2 - b^2}{2}$
- can be turned into a modular multiplication for odd moduli
- requires 2 subtractions, 1–4 additions, 1 right shift
- called *squaring multiplication*

Performance of Squaring Multiplication



Multi-Powers using the cryptographic coprocessor



Outline

Introduction

Performance critical part: Multi-powers on Java Card

OV-Chip 2.0 Applet: Structure and Performance

Conclusion

Implementation

Features implemented

- key setup, choose number of attributes, selection of bases (g_i)
- applet initialisation
  - attribute selection (a_i)
  - download key material, bases, attributes to the card
  - first blinding, signature generation
- proof protocol
  - signature check
  - card proves knowledge of all its attributes
- resign protocol
  - arbitrary attribute update
  - new blinding, new signature

Implementation (cont.)

Features missing

- selective disclosure of some attributes
- proving relations (balance $>$ €20)
- ...

2 Applets

Coprocessor-enabled applet: Uses the crypto coprocessor as far as possible

Pure Java Applet: Only JVM, without crypto coprocessor, very very slow

One Host driver: can talk to both applets

Implementation (cont.)

Features missing

- selective disclosure of some attributes
- proving relations (balance $>$ €20)
- ...

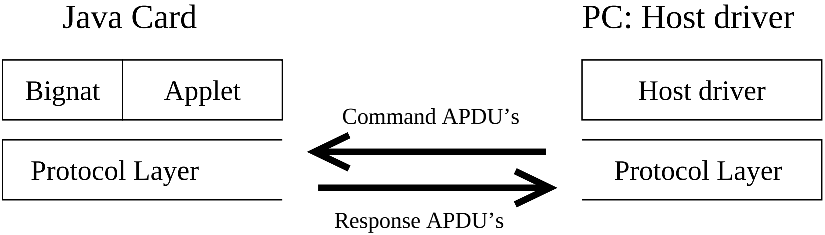
2 Applets

Coprocessor-enabled applet: Uses the crypto coprocessor as far as possible

Pure Java Applet: Only JVM, without crypto coprocessor, very very slow

One Host driver: can talk to both applets

Application structure

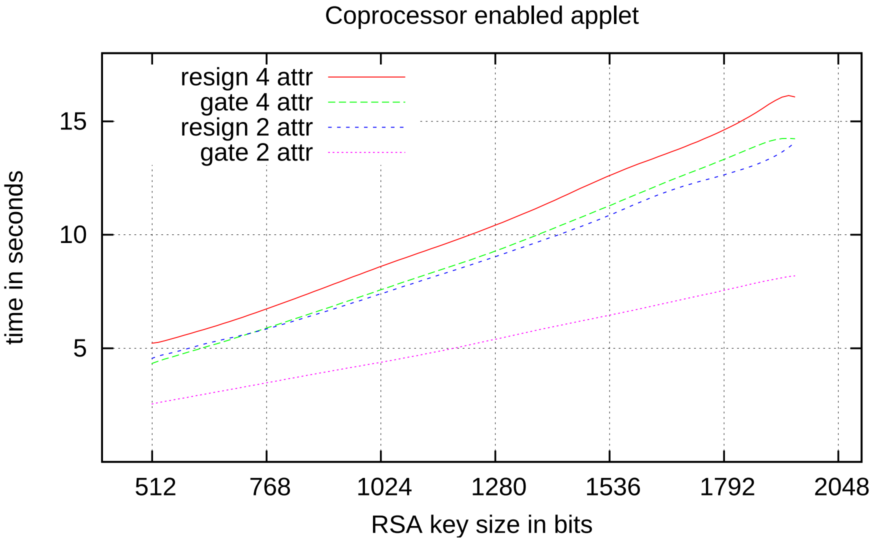


Bignat newly written big-integer library for Java Card
with convenience methods for the crypto coprocessor
uses byte/short normally, but can be compiled using int/long

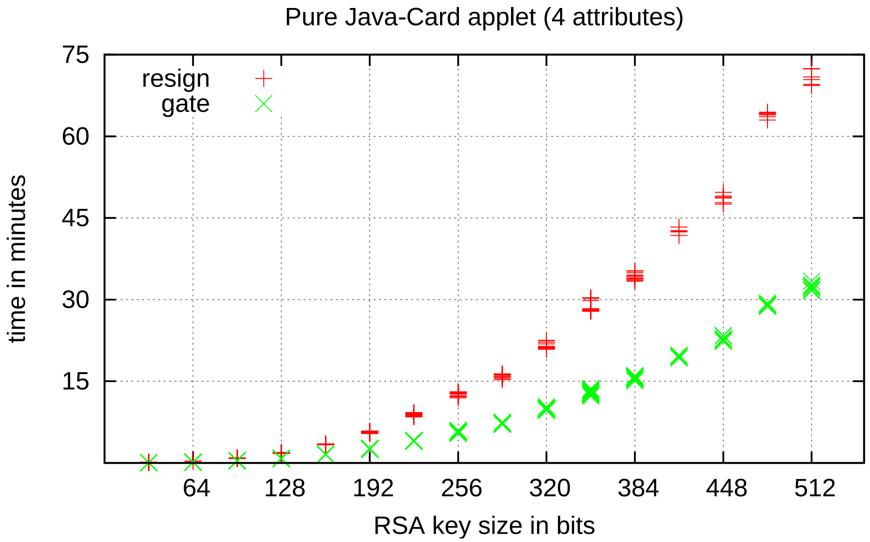
Protocol Layer Custom remote method library
supporting arbitrary many arguments and results up to 32 KByte,
also supporting easy applet debugging on the PC

Host driver relies on `java.math.BigInteger`

Coprocessor-enabled applet



Pure Java Card Applet



Outline

Introduction

Performance critical part: Multi-powers on Java Card

OV-Chip 2.0 Applet: Structure and Performance

Conclusion

Conclusion

Java Card API does not facilitate new cryptographic protocols

- necessary big-integer operations not supported in the API
- no card seems to implement `javacardx.framework.math.BigNumber`
- most cards *do* implement the needed operations in a native library
- API limitations force big-integer operations on the Java Card VM **with terrible performance results**

Achievements

Brands protocols: implementation on real Java Card

- 5–10 seconds for RSA keys with short term security
- not ready for public transport
- ready for special usage, e.g., over the internet
- <1 second possible with optimal crypto coprocessor access

Bignat library for mutable big-integer operations on Java Card

Protocol Layer Custom remote method library (up to 32KB arguments/results)

Download **(almost) all sources at http://www.sos.cs.ru.nl/ovchip/**

Conclusion

Java Card API does not facilitate new cryptographic protocols

- necessary big-integer operations not supported in the API
- no card seems to implement `javacardx.framework.math.BigNumber`
- most cards *do* implement the needed operations in a native library
- API limitations force big-integer operations on the Java Card VM **with terrible performance results**

Achievements

Brands protocols: implementation on real Java Card

- 5–10 seconds for RSA keys with short term security
- not ready for public transport
- ready for special usage, e.g., over the internet
- <1 second possible with optimal crypto coprocessor access

Bignat library for mutable big-integer operations on Java Card

Protocol Layer Custom remote method library (up to 32KB arguments/results)

Download **(almost) all sources at http://www.sos.cs.ru.nl/ovchip/**

Conclusion

Java Card API does not facilitate new cryptographic protocols

- necessary big-integer operations not supported in the API
- no card seems to implement `javacardx.framework.math.BigNumber`
- most cards *do* implement the needed operations in a native library
- API limitations force big-integer operations on the Java Card VM **with terrible performance results**

Achievements

Brands protocols: implementation on real Java Card

- 5–10 seconds for RSA keys with short term security
- not ready for public transport
- ready for special usage, e.g., over the internet
- <1 second possible with optimal crypto coprocessor access

Bignat library for mutable big-integer operations on Java Card

Protocol Layer Custom remote method library (up to 32KB arguments/results)

Download (almost) all sources at http://www.sos.cs.ru.nl/ovchip/

Conclusion

Java Card API does not facilitate new cryptographic protocols

- necessary big-integer operations not supported in the API
- no card seems to implement `javacardx.framework.math.BigNumber`
- most cards *do* implement the needed operations in a native library
- API limitations force big-integer operations on the Java Card VM **with terrible performance results**

Achievements

Brands protocols: implementation on real Java Card

- 5–10 seconds for RSA keys with short term security
- not ready for public transport
- ready for special usage, e.g., over the internet
- <1 second possible with optimal crypto coprocessor access

Bignat library for mutable big-integer operations on Java Card
Protocol Layer Custom remote method library (up to 32KB arguments/results)

Download (almost) all sources at http://www.sos.cs.ru.nl/ovchip/

Conclusion (cont.)

General Java Card

- Java Card is not compatible with Java
  System.arraycopy versus javacard.framework.Util.arrayCopyNonAtomic or javacard.framework.Util.arrayCopy
- Java is not really suited for programming Java cards
  idealised Java: all platforms are identical/no conditional compilation
  reality: Java and Java Card differ a lot!

Needed

- additional classes in the Java Card API with
  - modular exponentiation, multiplication, addition/subtraction
  - division, modulus
  - modular inverse
  - elliptic curve point addition, scalar point multiplication
- Cards implementing `javacardx.framework.math.BigNumber`
- Cards supporting `int`
- conditional compilation for code running on both JVM and JCVM

Conclusion (cont.)

General Java Card

- Java Card is not compatible with Java
  System.arraycopy versus javacard.framework.Util.arrayCopyNonAtomic or javacard.framework.Util.arrayCopy
- Java is not really suited for programming Java cards
  idealised Java: all platforms are identical/no conditional compilation
  reality: Java and Java Card differ a lot!

Needed

- additional classes in the Java Card API with
  - modular exponentiation, multiplication, addition/subtraction
  - division, modulus
  - modular inverse
  - elliptic curve point addition, scalar point multiplication
- Cards implementing `javacardx.framework.math.BigNumber`
- Cards supporting `int`
- conditional compilation for code running on both JVM and JCVM

Conclusion (cont.)

General Java Card

- Java Card is not compatible with Java
  System.arraycopy versus javacard.framework.Util.arrayCopyNonAtomic or javacard.framework.Util.arrayCopy
- Java is not really suited for programming Java cards
  idealised Java: all platforms are identical/no conditional compilation
  reality: Java and Java Card differ a lot!

Needed

- additional classes in the Java Card API with
  - modular exponentiation, multiplication, addition/subtraction
  - division, modulus
  - modular inverse
  - elliptic curve point addition, scalar point multiplication
- Cards implementing `javacardx.framework.math.BigNumber`
- Cards supporting `int`
- conditional compilation for code running on both JVM and JCVM