Selective disclosure protocols on Java Card Eine Fallstudie für das niederländische Nahverkehrssystem

Hendrik Tews

EZAG 19. März 2010

Outline


Alléén
ov chipkaart
GVB


OV-Chipkaart


Ontwikkeling Reisgebied OV-chipkaart
Reisgebied per 11 februari 2010
Trein NS
Trein overige
OV-chipkaart enige vervoerbewijs
Volgt in 2e kwartaal 2010
Volgt later
www.ov-chipkaart.nl

Technologie und Probleme

- $\approx$ 15 Verkehrsunternehmen beteiligt
- $>$ 600 Millionen € Kosten bisher
- Identitätsbasiert
- basiert auf Mifare-Classic und Mifare-Ultralight
  - genauso wie Oyster (London), Octopus (Hong Kong), Emeal vom Studentenwerk Dresden, Zugangskarten für diverse Universitäten/Ministerien/Militärbasen
  - konstante Anti-Kollisions ID's
  - proprietäre Verschlüsselung 2008 gebrochen
  - Karten können geklont werden
  - Saldo kann erhöht werden
  - System ist trotzdem sicher!
- anonyme Karten nur gegen Aufpreis (und ohne Ermäßigungen)
- zentrale Datenbank mit allen Transaktionen

OV chip 2.0

- Design und Implementierung einer Nahverkehrskarte mit Vertraulichkeit und Integrität
- durch die nInet-Stiftung gesponsort
- Implementierung auf Java Card

Outline

Kenntnis eines Geheimnisses beweisen, ohne es zu verraten

- dem Geldautomaten beweisen, dass man die PIN kennt, *ohne sie einzutippen*
- im Whiskyladen beweisen, dass man über 18 ist, *ohne Geburtsdatum oder Alter zu verraten*
- beweisen, dass man eine gültige Fahrkarte hat, *ohne deren Seriennummer zu verraten um anonym zu reisen*

Schnorrs Identifikations protokoll

Parameter Primzahl p (1024 bit), Primzahl q (160 Bit) sodass $q \mid p-1$
α Generator modulo p, $g = \alpha^{(p-1)/q} \pmod p$ hat Ordnung q

Schlüssel Beweiser Brita wählt privaten Schlüssel $x < q$
Britas öffentlicher Schlüssel $y = g^x \pmod p$

Protokoll

Beweiser Brita		Prüfer Peter
$r < q$, zufällig, $t = g^r$	$\xrightarrow{t}$	
	$\xleftarrow{c}$	challenge $c < q$, zufällig
berechnet $s = r + cx$	$\xrightarrow{s}$	Beweis akzeptiert, falls $g^s \stackrel{?}{=} t\,y^c$

Test $g^s = g^r \cdot (g^x)^c = t \cdot y^c$

Outline

Java Card Überblick

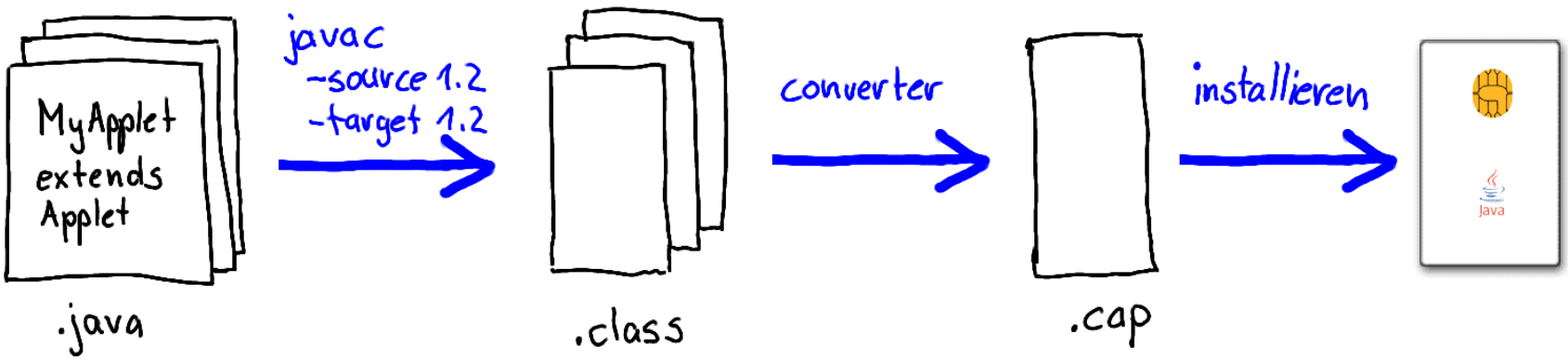
- 30 MHz CPU
- 160 KB ROM (64 K frei), 72 KB EEPROM, 4 K RAM ($\approx$ 2 K frei)
- Java byte code sadd dauert 15–45 μs
- speziell gesicherte virtuelle Machine für Java Card
- Transaktionen, atomare Speicherzugriffe
- spezielle Hardwarebeschleunigung für SHA1, DSA, DES, RSA, Elliptic-curve Diffie-Hellmann
- Java Card $\subset$ Java, trotzdem nicht kompatibel

unterstüzt	nicht unterstüzt
boolean, byte, short	long, double, float
32-bit-int optional	enums, assertions
eindimensionale arrays	mehrdimensionale arrays
packages, classes	reflection
interfaces, exceptions	threads, serialization, cloning
GC optional, nicht automatisch	

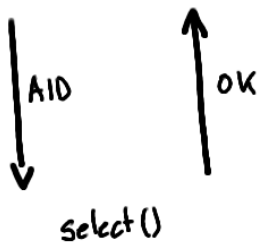
Java Card Überblick II

- Versionen
  - **2.2.1** von 2003
    - 10–20 €
    - max 256 Byte Daten per APDU
  - **2.2.2** von 2006
    - 32KByte APDU's
    - optional javacardx.framework.math.BigNumber gibt Zugriff auf Crypto-Coprozessor
    - Karten praktisch nicht erhältlich; bekannte Exemplare implementieren BigNumber nicht
  - **3.0** Spec im April 2008 veröffentlicht
    - erste Produktionsmuster
    - Web-server auf der Karte
- Java Card RMI
  - Remote Method Invocation for Java cards
  - serialisierte Argumente/Resultate müssen in eine APDU passen
  - praktisch kaum benutzbar

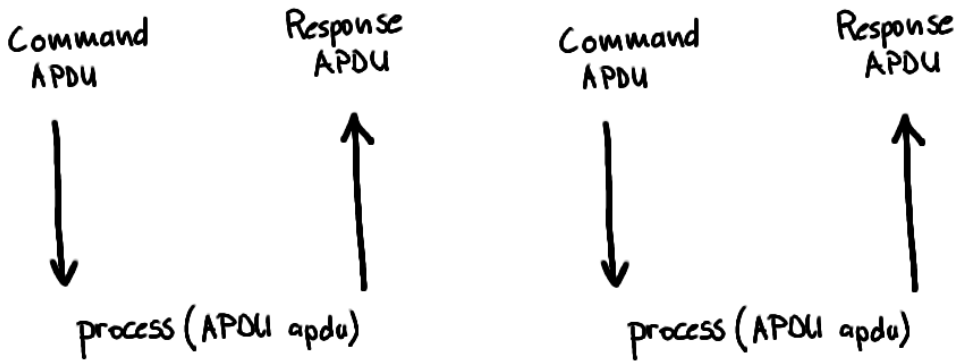
Applet Lebenslauf



Applet wählen



Kommunikation



Probleme für nicht-triviale Applets

1. Debugging, gemeinsamer Code für Applet und Host
2. potenzieren von Zahlen mit 1000 – 2000 Bit
3. Argumente/Resultate passen nicht in eine APDU
   - bis zu 3800 Byte (15 APDU's)
   - Größe variiert je nach Konfiguration
     - 4 Attribute, 1952 Bit RSA: 753 Byte (3 APDU's)
     - 2 Attribute, 512 Bit RSA: 213 Byte (1 APDU)

Gemeinsamer Code für Applet und Host

- z.B. für Konfiguration, serialisierbare Datentypen, RMI Signaturen
- Debugging: Applet- *und* Hostcode laufen zusammen auf *einer* JVM
- Inkompatibilitäten zwischen Java Card und Java
  - `package ds_ov2_srsa` vs. `package ds.ov2.bignat`
  - `import java.security.MessageDigest` vs. `javacard.security.MessageDigest`
  - `arrayCopyNonAtomic` oder `arrayCopy` vs. `arraycopy`
- keine bedingte Kompilation in Java
  (es gibt ja Reflexion — außer in Java Card)
- copy & paste ist Standardlösung

- Meine Lösung
  - cpp
  - `sed -e 's|^# |//# |'`
  - emacs defadvice für next-error

Gemeinsamer Code für Applet und Host

- z.B. für Konfiguration, serialisierbare Datentypen, RMI Signaturen
- Debugging: Applet- *und* Hostcode laufen zusammen auf *einer* JVM
- Inkompatibilitäten zwischen Java Card und Java
  - `package ds_ov2_srsa` vs. `package ds.ov2.bignat`
  - `import java.security.MessageDigest` vs. `javacard.security.MessageDigest`
  - `arrayCopyNonAtomic` oder `arrayCopy` vs. `arraycopy`
- keine bedingte Kompilation in Java
  (es gibt ja Reflexion — außer in Java Card)
- copy & paste ist Standardlösung

- Meine Lösung
  - cpp
  - `sed -e 's|^# |//# |'`
  - emacs defadvice für next-error

Bignat Programmbibliothek

- keine Big-Integer Bibliothek für Java Card verfügbar
- Portierung auf Java Card ist sehr aufwendig und wenig sinnvoll (immutable, GC)

Bignat

- ziemlich anwendungsspezifisch (positive Zahlen fester Größe)
- mutable, allokationsfrei
- spezielle Methoden zum Rechnen auf dem Crypto Coprocessor
- Basisdatentypen sind byte/short *oder* int/long

Protocoll Layer: RPC für Java Card bis 32KB

- Programmbibliothek für
  - (De-)Serialisierung von Argumenten und Resultaten
  - Aufteilung in mehrere APDU's
- idl Compiler (perl, 1500 LoC) erzeugt aus

```
protocol resign
    step sig_hash: d.host_alpha, d.remainders -> d.sig_remainder
        call card.make_sig_hash()
```

  - RPC Beschreibung als Java Objekt
  - Stub-Code inklusive Typkonvertierungen

```
public BigInteger sig_hash_call(CardChannel _cc, BigInteger _a_1,
                                Host_vector _a_2)
```

  - Resultattypdeklarationen für Methoden mit mehreren Resultaten
  - Testframe Stub-Code führt Methoden direkt aus

Outline

Overview

Protocol for Selective Disclosure

- customer possesses attributes
  (private key, kind of ticket, expiry date, balance, . . .)
- 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.

Overview

Protocol for Selective Disclosure

- customer possesses attributes
  (private key, kind of ticket, expiry date, balance, . . .)
- 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.

Overview

Protocol for Selective Disclosure

- customer possesses attributes
  (private key, kind of ticket, expiry date, balance, . . .)
- 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.

Protocol Example

(here for proving knowledge of two attributes only)

System setup (glossing over side conditions)

$n = pq$ RSA modulus (about 1280 bits)

$g_1, g_2 < n$ public bases

v public RSA exponent (about 160 bits)

Card setup

$a_1, a_2 < v$ attributes of the card

$b < n$ blinding

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

Protocol Example (cont.)

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

Card Commitment

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

Gate Challenge

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

Card Response

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

Acceptance Check

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

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

Protocol Example (cont. II)

Shortcomings of the partial disclosure protocol

- A is a pseudonym
  need to change β now and then
- card could invent attributes on the fly
  need to sign the blinded attribute expression A

Resign Protocol

- update attributes values (e.g., adjusting the balance)
- card chooses new blinding β
- card obtains a new signature on the new A

Protocol Example (cont. II)

Shortcomings of the partial disclosure protocol

- A is a pseudonym
  need to change β now and then
- card could invent attributes on the fly
  need to sign the blinded attribute expression A

Resign Protocol

- update attributes values (e.g., adjusting the balance)
- card chooses new blinding β
- card obtains a new signature on the new A

Outline

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

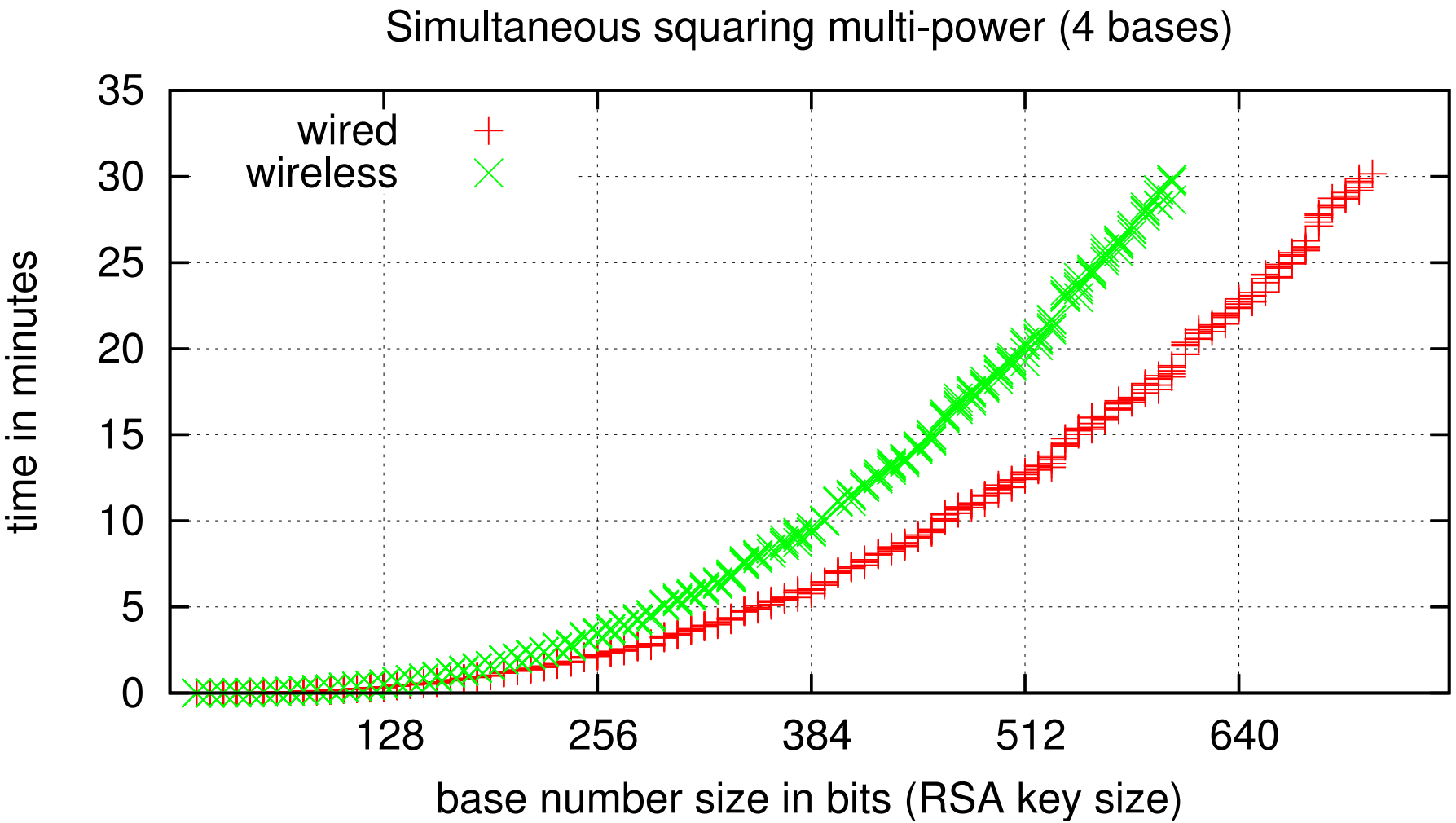
- $(g_1^{a_1} g_2^{a_2} \cdots g_n^{a_n}) \mod n$
- bases of 1300–2000 bits, exponents of 160–200 bits

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

Pure Java Card Multi-Powers



(for much more measurements go to `ovchip.cs.ru.nl/OV-chip_2.0_progress`)

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

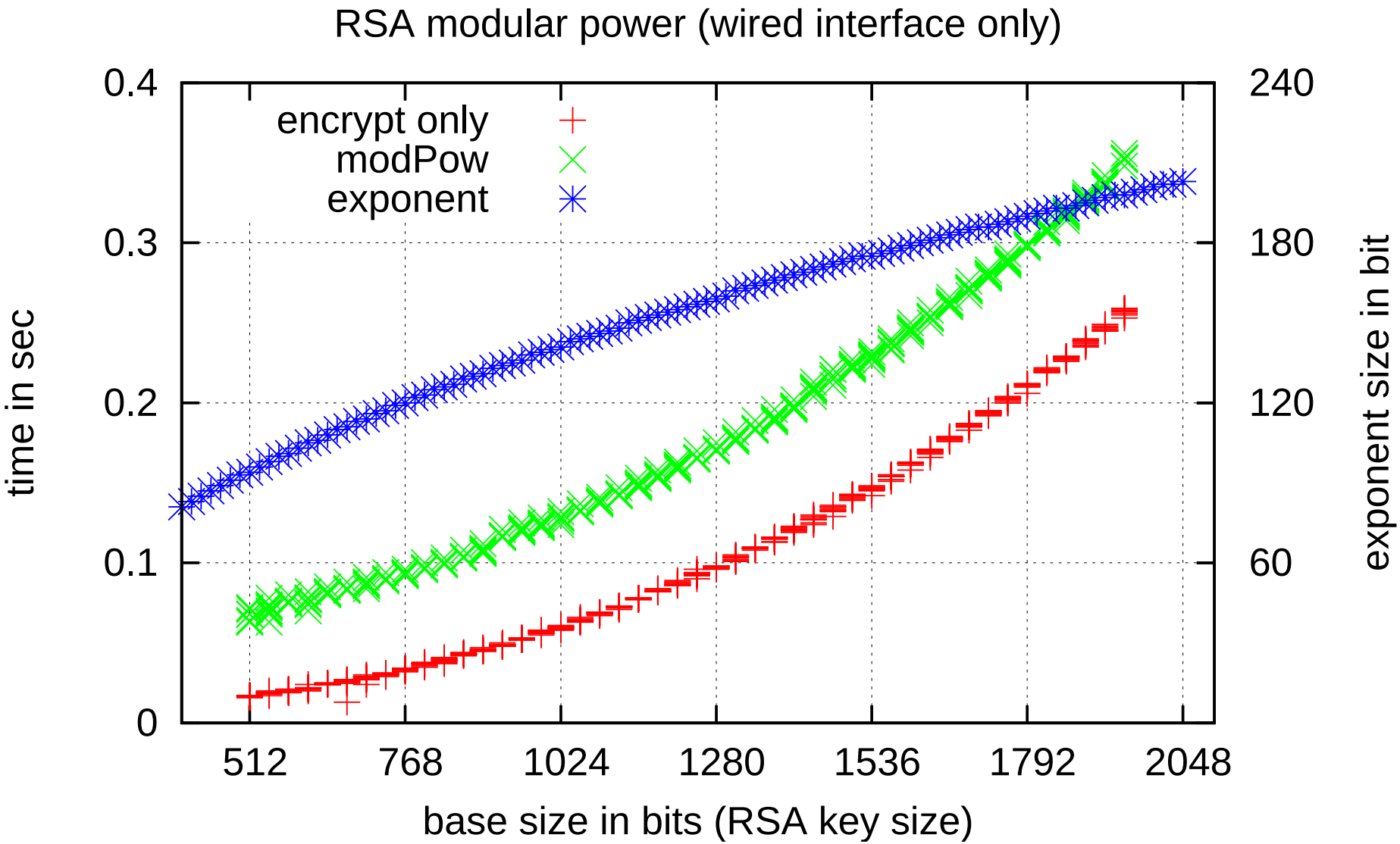
BigNumber class in Java Card 2.2.2

- 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



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

$(a+b)^2 - (a-b)^2 = 4ab$

- can be turned into a modular multiplication for odd moduli
- requires 2 squares, 2 subtractions, 1–3 additions, 1 right shift, 1 comparison
- called *squaring multiplication*

Modular Products?

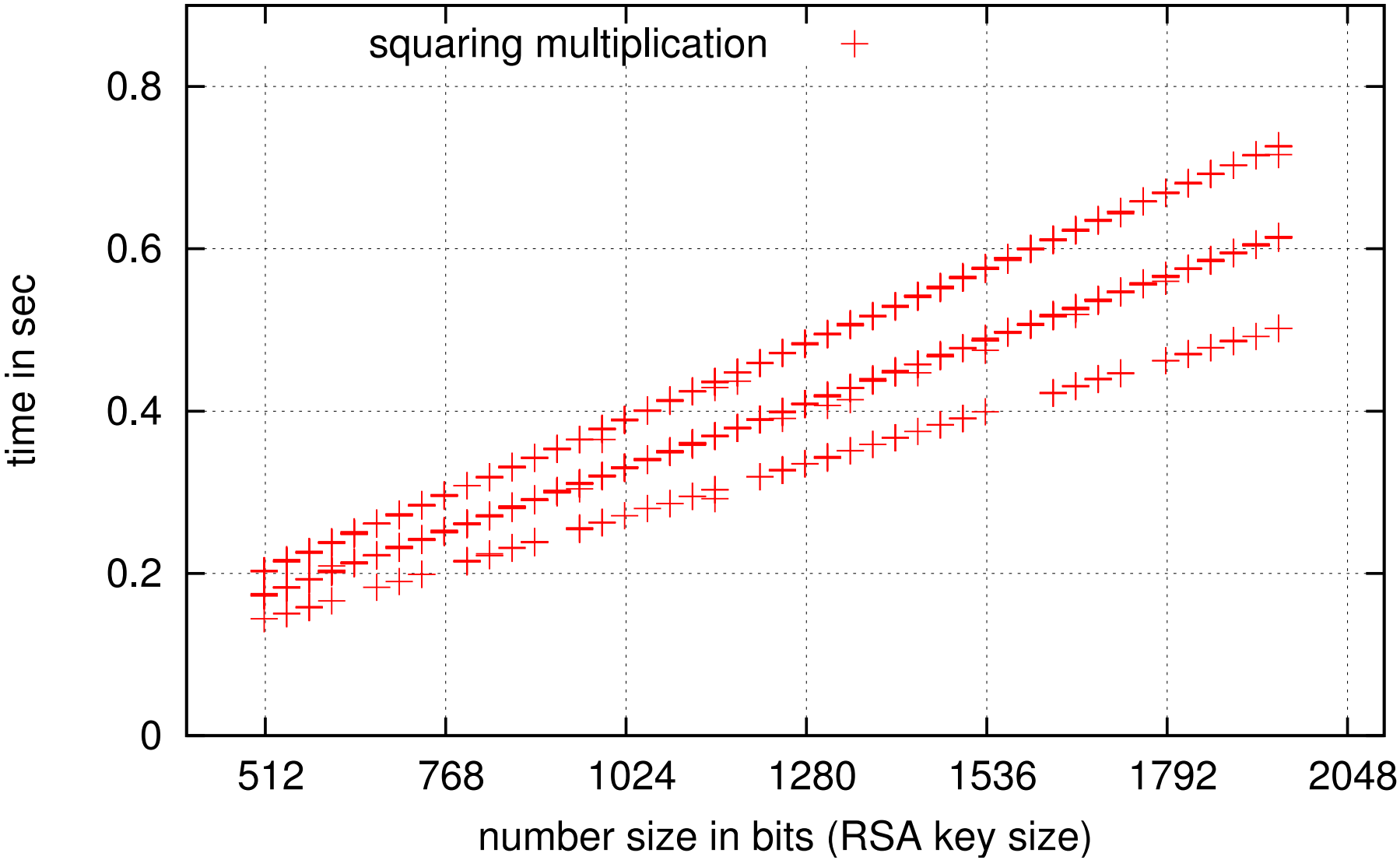
$(a\,b) \mod n$

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

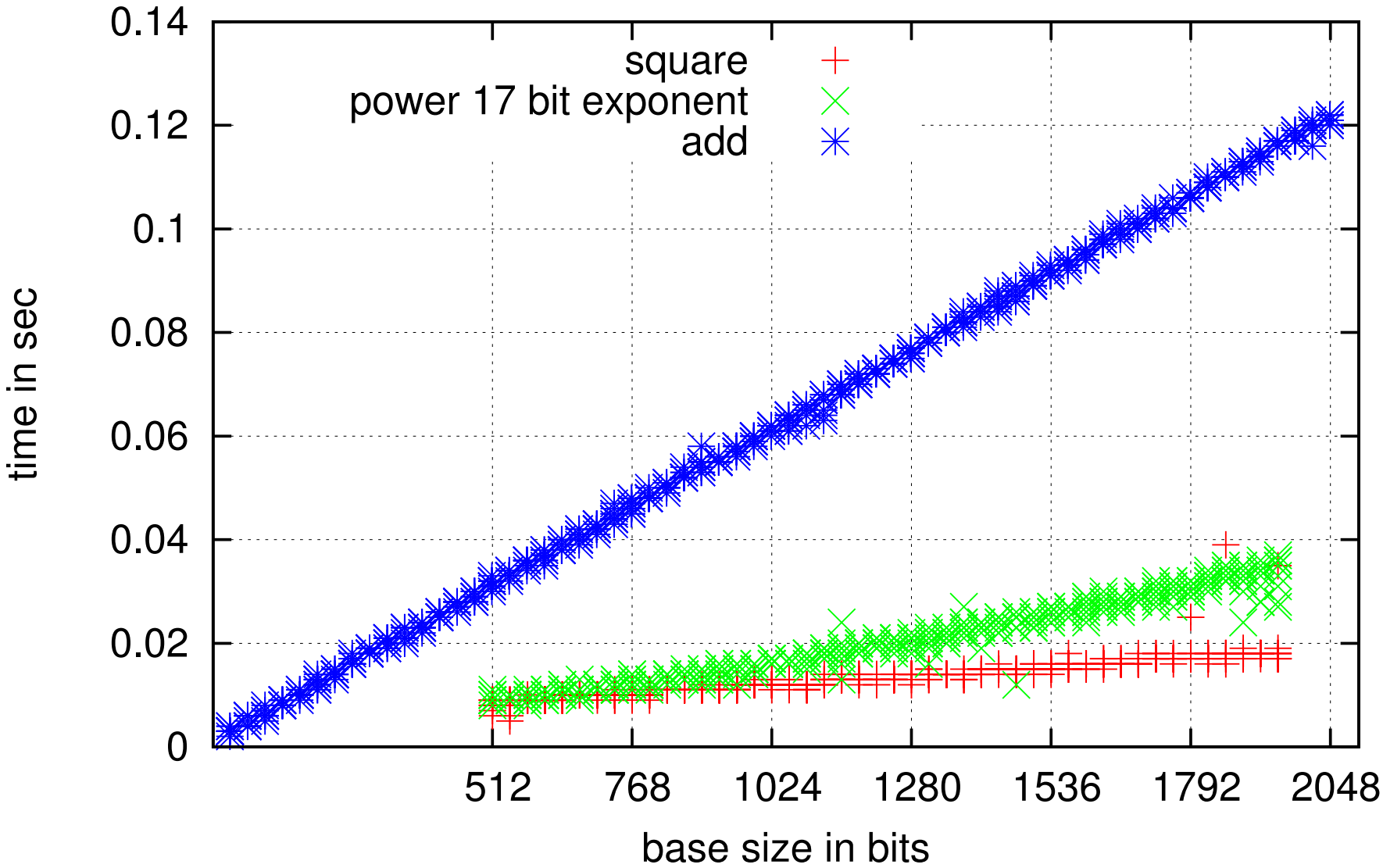
$(a+b)^2 - (a-b)^2 = 4ab$

- can be turned into a modular multiplication for odd moduli
- requires 2 squares, 2 subtractions, 1–3 additions, 1 right shift, 1 comparison
- called *squaring multiplication*

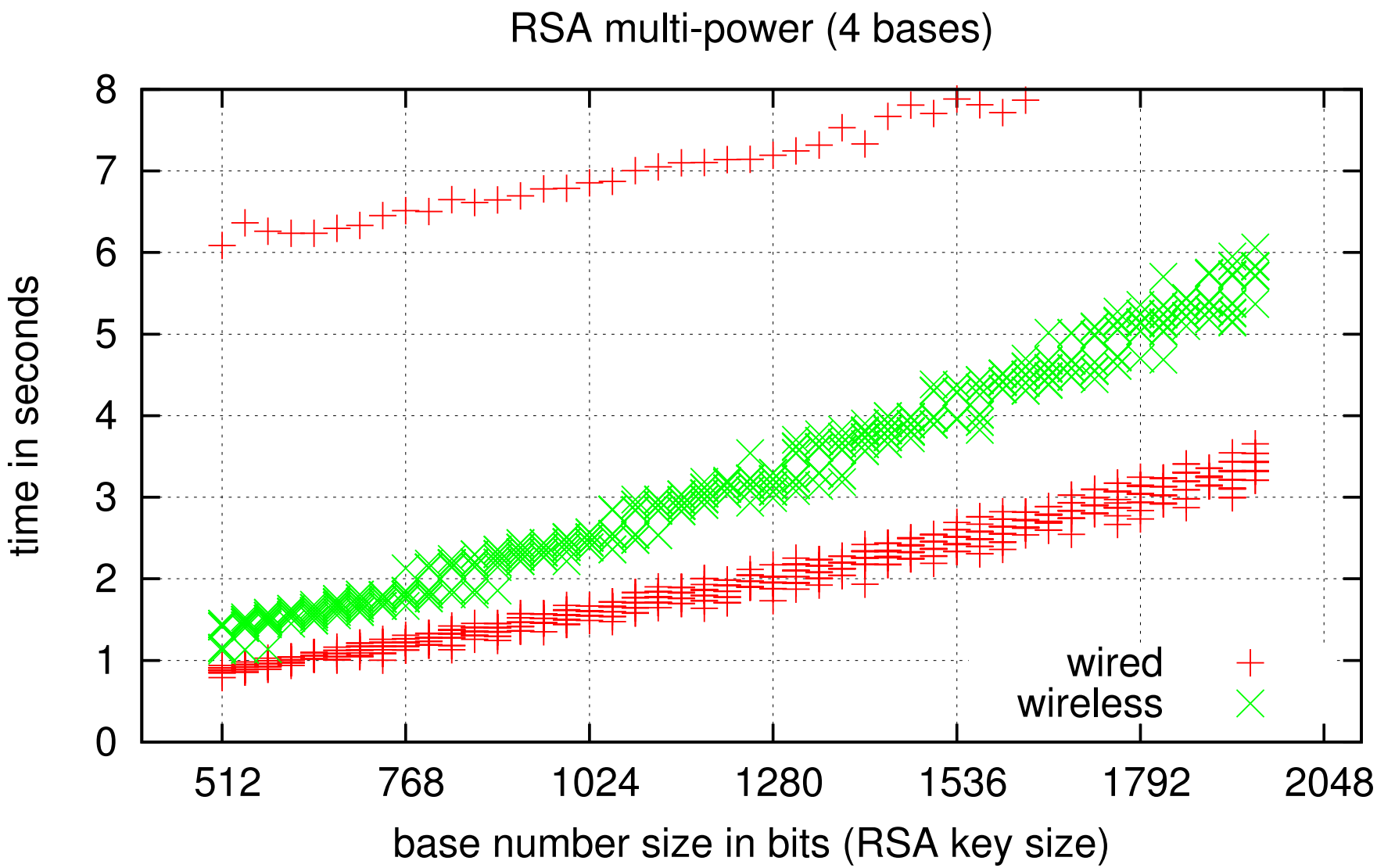
Performance of Squaring Multiplication



Addition and Squaring



Multi-Powers using the cryptographic coprocessor



Outline

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)
- ...

4 Applets

- **Pure** Reines JCVM applet, kein Coprocessor, äußerst langsam
- **Coprocessor + Montgomery Multiplikation**
- **Coprocessor + squaring Multiplikation I** nutzt $2ab = (a+b)^2 - a^2 - b^2$
- **Coprocessor + squaring Multiplikation II** nutzt $4ab = (a+b)^2 - (a-b)^2$
- **Ein Host driver** für alle Applets
- **GUI** für alle Applets

Implementation (cont.)

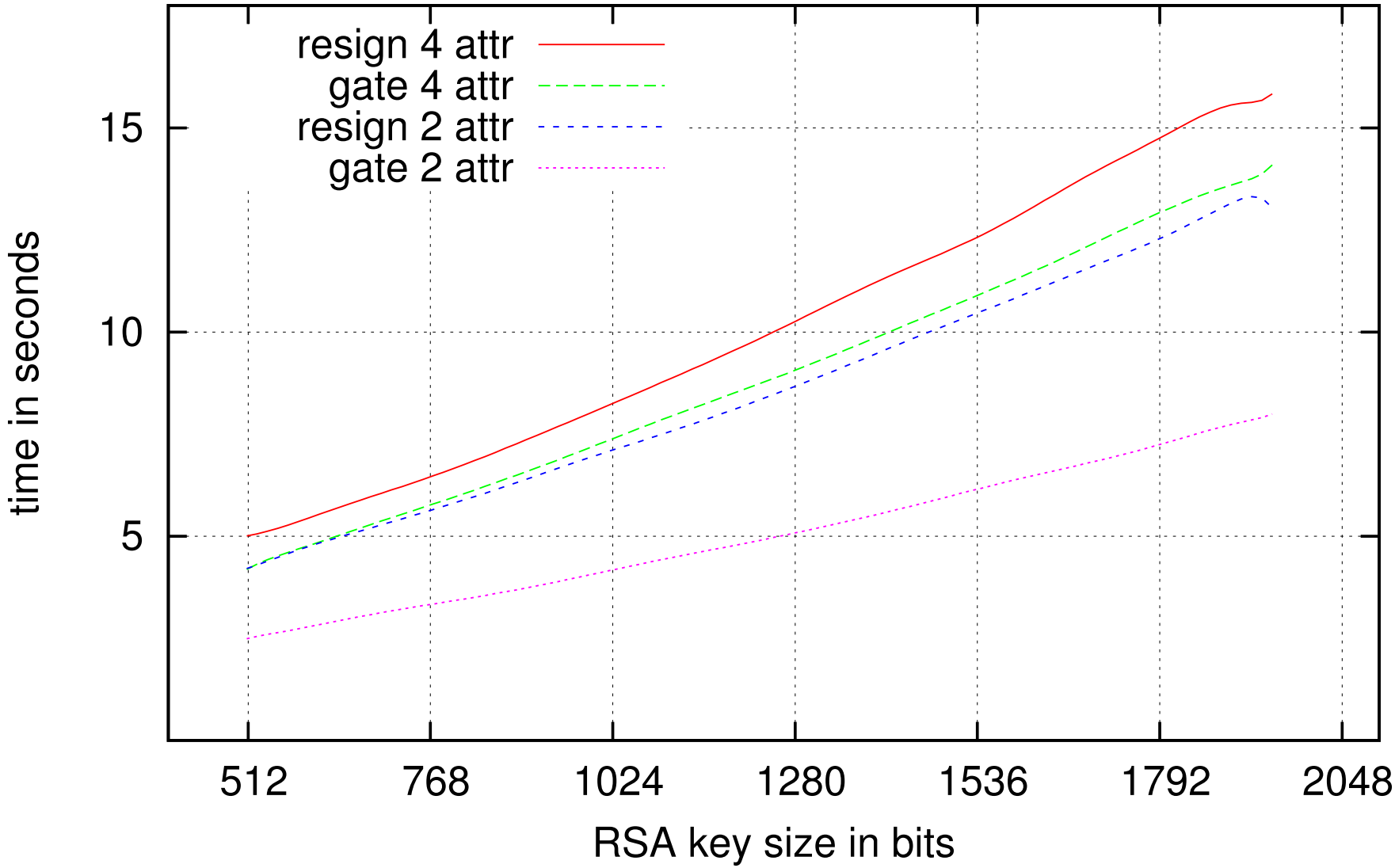
Features missing

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

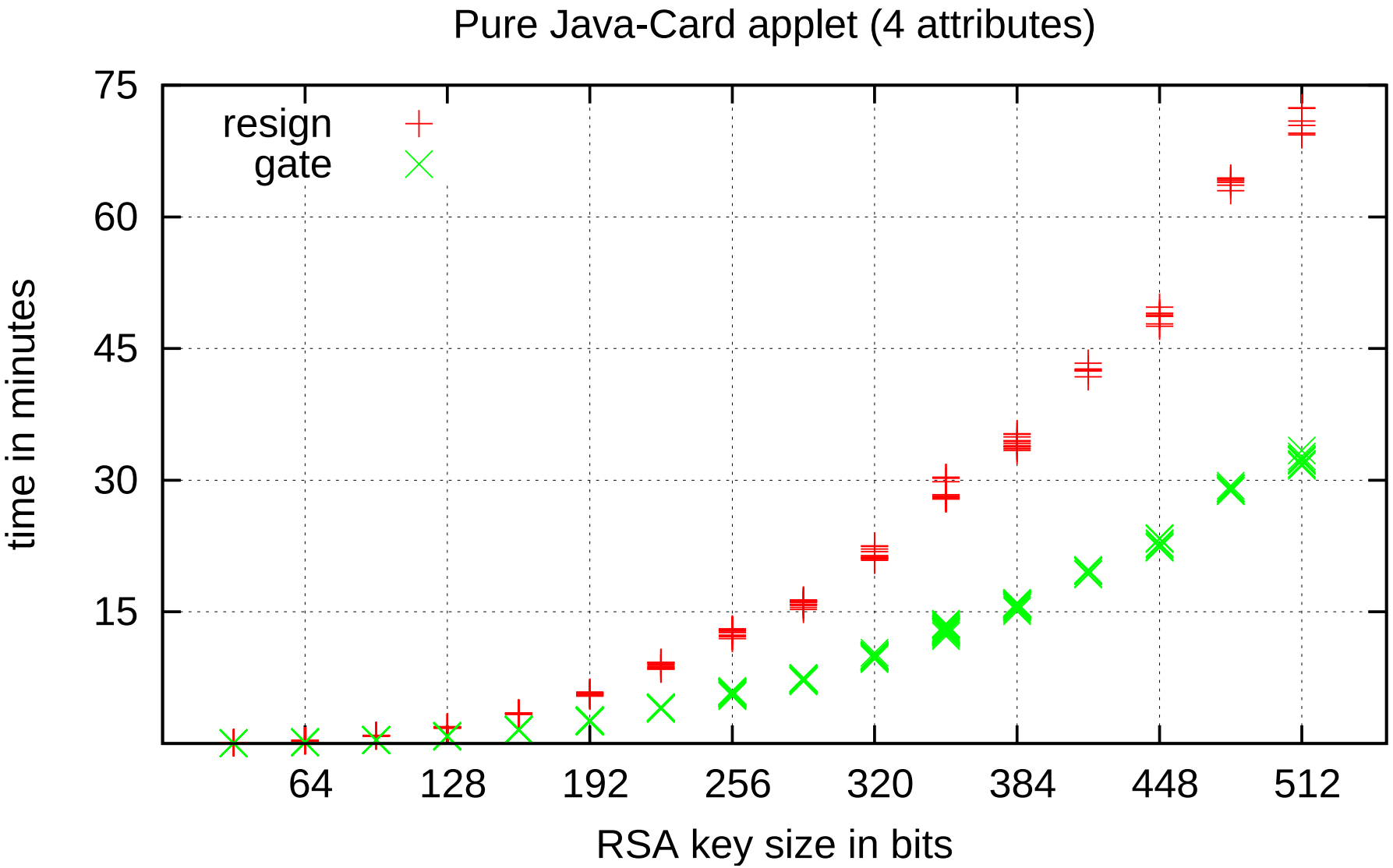
4 Applets

- **Pure** Reines JCVM applet, kein Coprocessor, äußerst langsam
- **Coprocessor + Montgomery Multiplikation**
- **Coprocessor + squaring Multiplikation I** nutzt $2ab = (a+b)^2 - a^2 - b^2$
- **Coprocessor + squaring Multiplikation II** nutzt $4ab = (a+b)^2 - (a-b)^2$
- **Ein Host driver** für alle Applets
- **GUI** für alle Applets

Performance of "Coprocessor + squaring II" applet



Pure Java Card Applet



Outline

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

Entwicklung nach Juli 2009

- Elliptische Kurven
- kurze Schlüssel (160–192 Bits)
- Diffie-Hellmann für Elliptische Kurven wird für skalare Multiplikation benutzt
- Laufzeit 2–3 Sekunden für ein Attribut