OpenCrypto

Unchaining the JavaCard Ecosystem

Vasilios Mavroudis
Doctoral Researcher, UCL

Who we are

Vasilios Mavroudis
Doctoral Researcher, UCL

Petr Svenda
Assistant Professor, MUNI

George Danezis
Professor, UCL

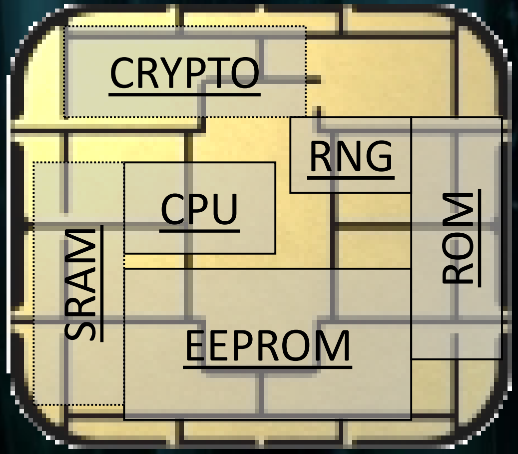
Dan Cvrcek
Founder, EnigmaBridge

Contents

1. Smart Cards & Java Cards
2. What’s the problem?
3. Our solution
4. Tools for developers
5. Future Work

SmartCards

- Pocket-sized card with integrated circuits embedded
- 8-32 bit CPU @ 10+MHz
- Cryptographic Coprocessor
- Persistent memory 32-150kB (EEPROM)
- Volatile fast RAM, usually <<10kB
- Secure Random Number Generator



SmartCards

Intended for physically unprotected environment

- Tamper protection
  - → Tamper-evidence (visible if physically manipulated)
  - → Tamper-resistance (can withstand physical attack)
  - → Tamper-response (erase keys…)
- Protection against side-channel attacks (power,EM,fault)
- Periodic tests of TRNG functionality

Use Cases

- Payments – ApplePay, Chip credit cards, …
- Government – ID cards, authentication, signing
- Cryptocurrencies – wallet protection
- Internet of Things

Why we like smartcards

- Secure – Small attack surface
- Certified to high levels of security (CC EAL5+, FIPS 140-2)
- Programmable secure execution environment
- Suitable for complicated business security transactions
- Inexpensive – 100 JCs form a low-end HSM

Operating Systems

MultOS

- Multiple supported languages
- Native compilation
- Certified to high-levels
- Often used in bank cards

.NET for smartcards

- Similar to JavaCard, but C#
- Limited market penetration

JavaCard

- Open platform from Sun/Oracle
- Applets portable between cards

JAVACARD

History

Until 1996:

- Every major smart card vendor had a proprietary solution
- Smart card issuers were asking for interoperability between vendors

In 1997:

- The Java Card Forum was founded
- Sun Microsystems was invited as owner of the Java technology
- And smart card vendors became Java Card licensees

The Java Card Spec is born

Sun was responsible for managing:

- The Java Card Platform Specifications
- The reference implementation
- A compliance kit

Today, 20 years after:

- Oracle releases the Java Card specifications (VM, RE, API)
- and provides the SDK for applet development

A success!

20 Billion Java Cards sold in total

~2 Billion Javacards sold per year

1 Billion contactless cards in 2016

Common Use Cases:

- Telecommunications
- Payments
- Loyalty Cards

JavaCardForum

The API Specification

- Lists all supported crypto algorithms and the relevant methods
- Straightforward to use for developers
- Ensures interoperability between manufacturers
- Implementations are certified for functionality and security

A full ecosystem with laboratories & certification authorities

The API Specification

- **3.0.5** 2015 - Diffie-Hellman modular exponentiation, RSA-3072, SHA3, plain ECDSA
- **3.0.4** 2011 - DES MAC8 ISO9797.
- **3.0.1** 2009 - SHA-224, SHA2 for all signature algorithms
- **2.2.2** 2006 - SHA-256, SHA-384, SHA-512, ISO9796-2, HMAC, Korean SEED
- **2.2.0** 2002 - EC Diffie-Hellman, ECC keys, AES, RSA with variable key length
- **2.1.1** 2000 - RSA without padding.

Wikipedia

Bad Omens I

Compliance

Vendors implement a subset of the API specification:

- No list of algorithms supported by each card
- The specific card must be tested by the developers

Examples

- RMI introduced in Java Card Spec. 2.2 (2003) → never adopted
- Java Card 3.0 Connected (2009) → never implemented
- Annotation framework for security interoperability → not adopted

Bad Omens II

Interoperability

- Most cards run a single applet
- Most applets written & tested for a single card
- Most applets run only on a single vendor's cards

Three years late

- 1 year to develop the new platform after the release of a specification
- 1 year to get functionality and security certification
- 1 year to produce and deploy the cards

Walled Gardens

Proprietary APIs

- Additional classes offering various desirable features
- Newer Algorithms, Math, Elliptic Curve Operations
- Vendor specific, interoperability is lost
- Only for large customers
- Small dev houses rarely gain access
- Very protective: NDAs, Very limited info on the internet

OPEN CRYPTO

Motivation

1. Time-to-Market: Speed-up availability of new cryptographic functions
2. Interoperability: provide a consistent library for different JC platforms
3. Learning curve: Make Java Card accessible to Java programmers

A new landscape:

- IoT needs a platform with these security characteristics
- Lots of small dev. houses
- They want to build various new things
- Java devs in awe → No Integers, Primitive Garbage Collection

Things People Already Built!

- Store and compute on PGP private key
- Bitcoin hardware wallet
- Generate one-time passwords
- 2 factor authentication
- Store disk encryption keys
- SSH keys secure storage

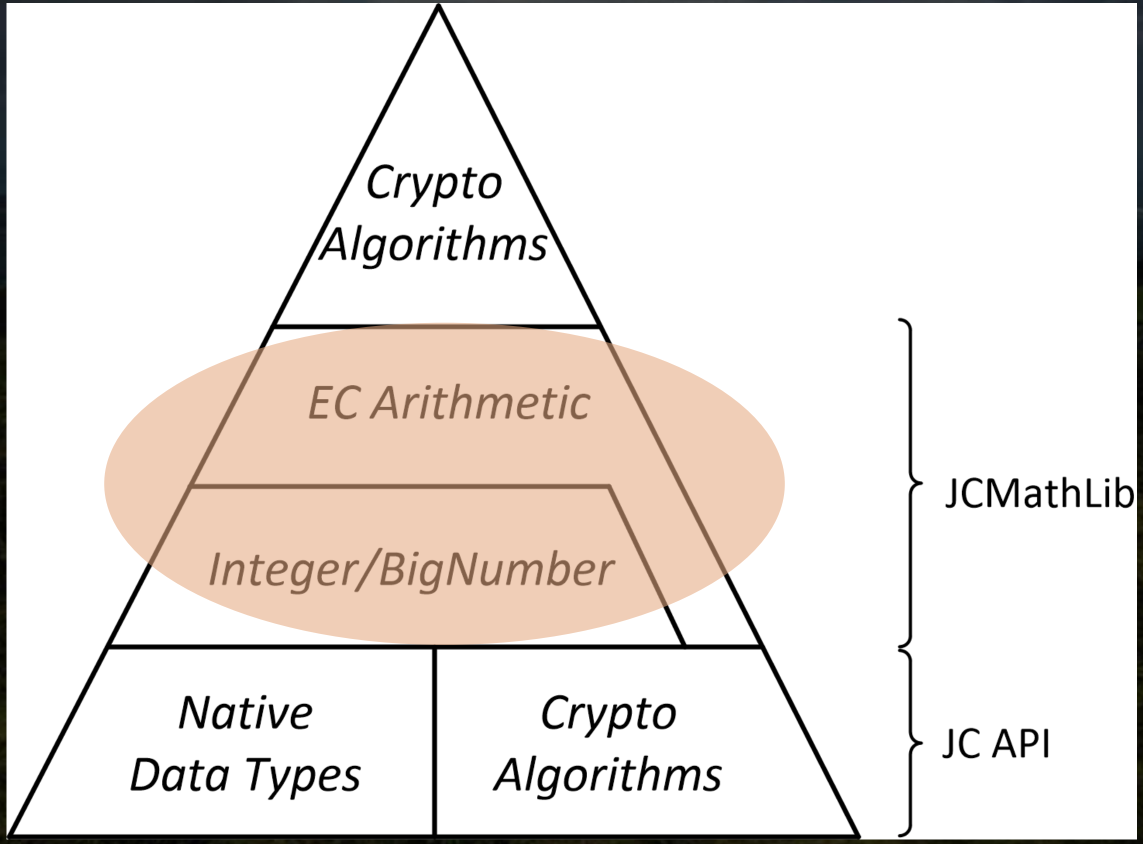
What if they had access to the full power of the cards?

List of JavaCard open-source apps: https://github.com/EnigmaBridge/javacard-curated-list

The OpenCrypto Project

Math Library

Dev Tools



Related Work

Project	Features	Details
OV-Chip 2.0	Big Natural Class	• Uses CPU • Card-specific • Not maintained
JCMath	Similar to Java BigInteger	• Part of project • Source code dump
E-Verification	MutableBigInteger Class	• Part of project • Source code dump

JCMath Lib

Class	Java	JC Spec.	JC Reality	JC MathLib
Integers	✓	✓	✗	✓
BigNumber	✓	✓	✗	✓
EC Curve	✓	~	~	✓
EC Point	✓	✗	✗	✓

OpencryptoJC.org/JCMathLib

JCMath Lib

Integer

- Addition
- Subtraction
- Multiplication
- Division
- Modulo
- Exponentiation

BigNumber

- Addition (+Modular)
- Subtract (+Modular)
- Multiplication (+Modular)
- Division
- Exponentiation (+Modular)
- ++, --

EC Arithmetic

- Point Negation
- Point Addition
- Point Subtraction
- Scalar Multiplication

```
package opencrypto.jcmathlib;

...




    public ECExample() {
        // Pre-allocate all helper structures
        ecc = new ECConfig((short) 256);
        // Pre-allocate standard SecP256r1 curve and two EC points on this curve
        curve = new ECCurve(false, SecP256r1.p, SecP256r1.a,
                            SecP256r1.b, SecP256r1.G, SecP256r1.r, ecc);
        point1 = new ECPoint(curve, ecc);
        point2 = new ECPoint(curve, ecc);
    }

...
```

```java
// NOTE: very simple EC usage example - no CLA/INS, no communication with host...
public void process(APDU apdu) {
    if (selectingApplet()) { return; }

    // Generate first point at random
    point1.randomize();
    // Set second point to predefined value
    point2.setW(ECPOINT, (short) 0, (short) ECPOINT.length);
    // Add two points together
    point1.add(point2);
    // Multiply point by large scalar
    point1.multiplication(SCALAR, (short) 0, (short) SCALAR.length);
}
```

Convenience Features

We handle the low-level/dirty stuff

- Unified memory management of shared objects
- Safe reuse of pre-allocated arrays:
  - → Resource Locking
  - → Automated erasure
- Adaptive data placement (RAM/EEPROM) for:
  - → performance
  - → memory usage

Building the Building Blocks

CPU is programmable! → But **very** slow ✗

Coprocessor is fast! → No direct access ✗

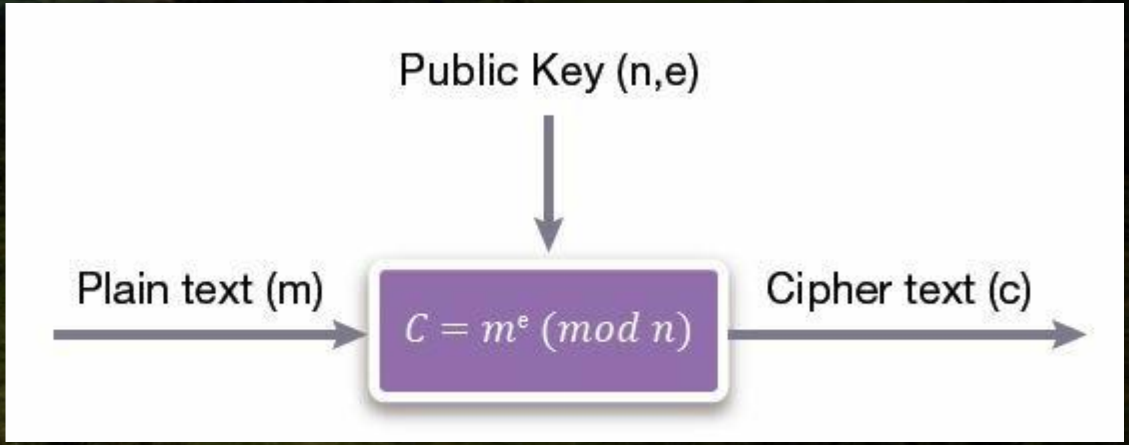
Hybrid solution

- Abuse API calls known to use the coprocessor
- CPU for everything else

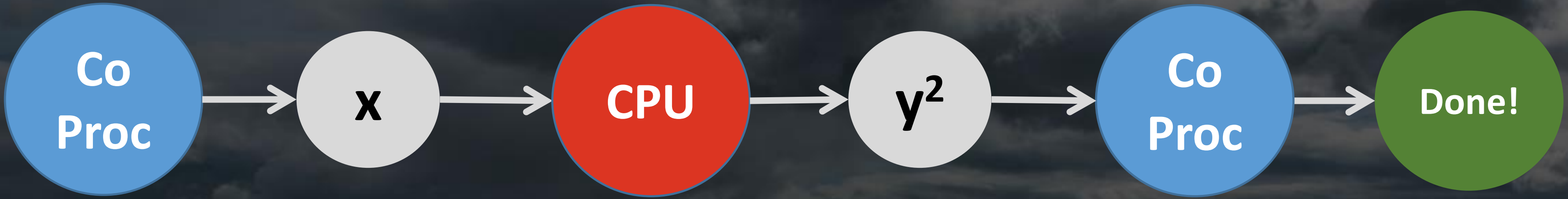
Simple Example

Modular Exponentiation with Big Numbers

- Very slow to run on the CPU
- Any relevant calls in the API?
  - → RSA Encryption ✓
  - → Uses the coprocessor ✓
  - → Limitations on the modulo size ✗
  - → *Modulo* on CPU has decent speed ✓

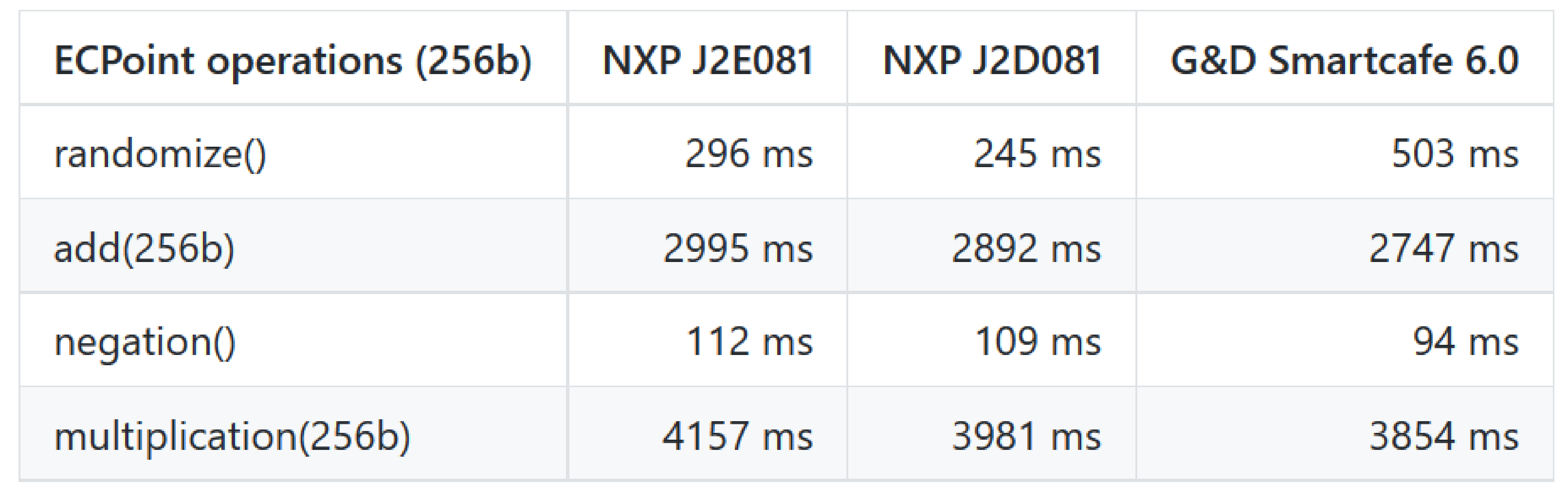


EC Point-scalar multiplication



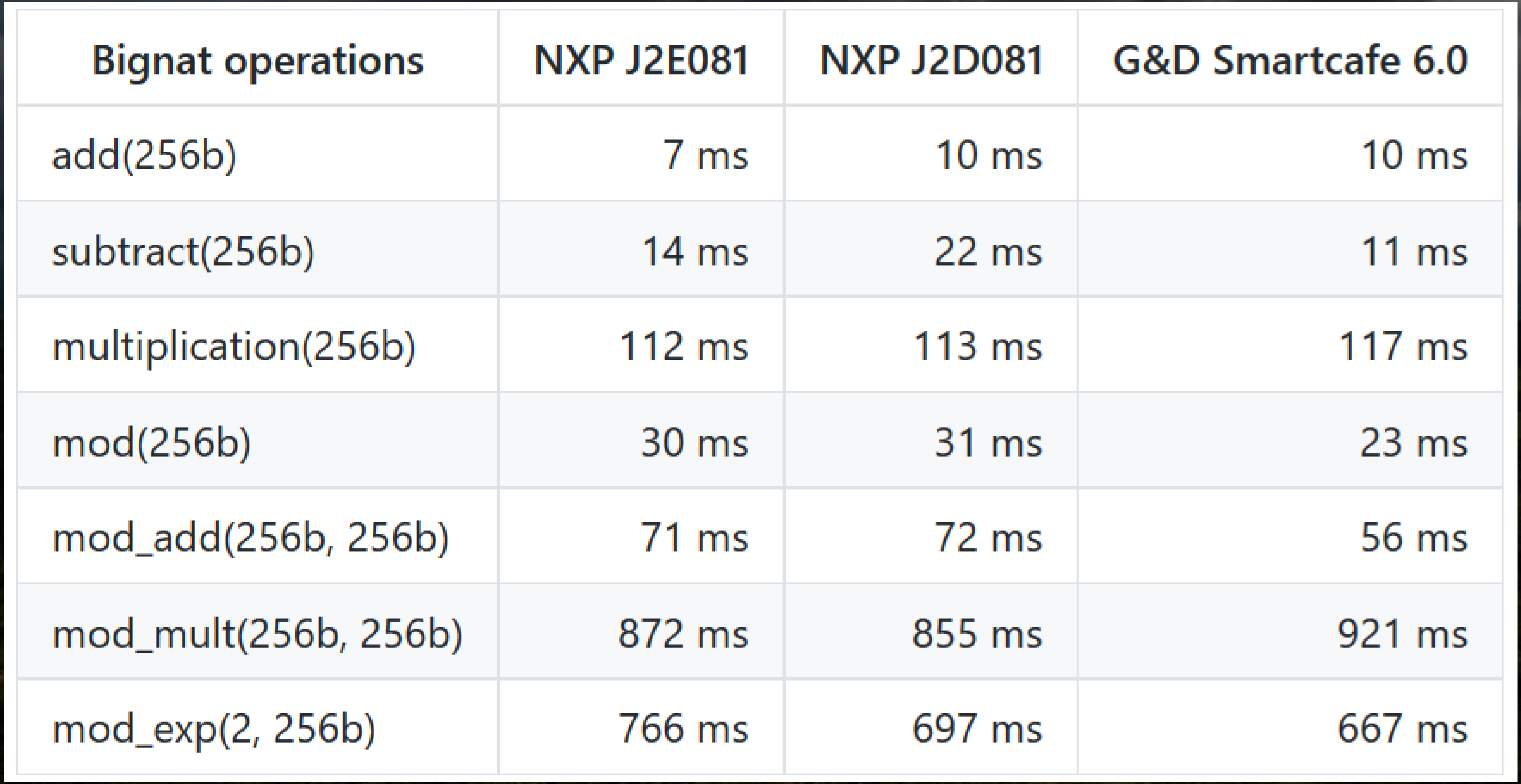
1. Input scalar x and point P
2. Abuse ECDH key exchange to get [x,+y,-y] (co-processor)
3. Compute the two candidate points P, P’ (CPU)
4. Sign with scalar x as priv key (co-processor)
5. Try to verify with P as pub key (co-processor)
6. If it works → It’s P
   else → It’s P’
7. return P or P’

Performance



ECPoint operations (256b)	NXP J2E081	NXP J2D081	G&D Smartcafe 6.0
randomize()	296 ms	245 ms	503 ms
add(256b)	2995 ms	2892 ms	2747 ms
negation()	112 ms	109 ms	94 ms
multiplication(256b)	4157 ms	3981 ms	3854 ms

Performance



Bignat operations	NXP J2E081	NXP J2D081	G&D Smartcafe 6.0
add(256b)	7 ms	10 ms	10 ms
subtract(256b)	14 ms	22 ms	11 ms
multiplication(256b)	112 ms	113 ms	117 ms
mod(256b)	30 ms	31 ms	23 ms
mod_add(256b, 256b)	71 ms	72 ms	56 ms
mod_mult(256b, 256b)	872 ms	855 ms	921 ms
mod_exp(2, 256b)	766 ms	697 ms	667 ms

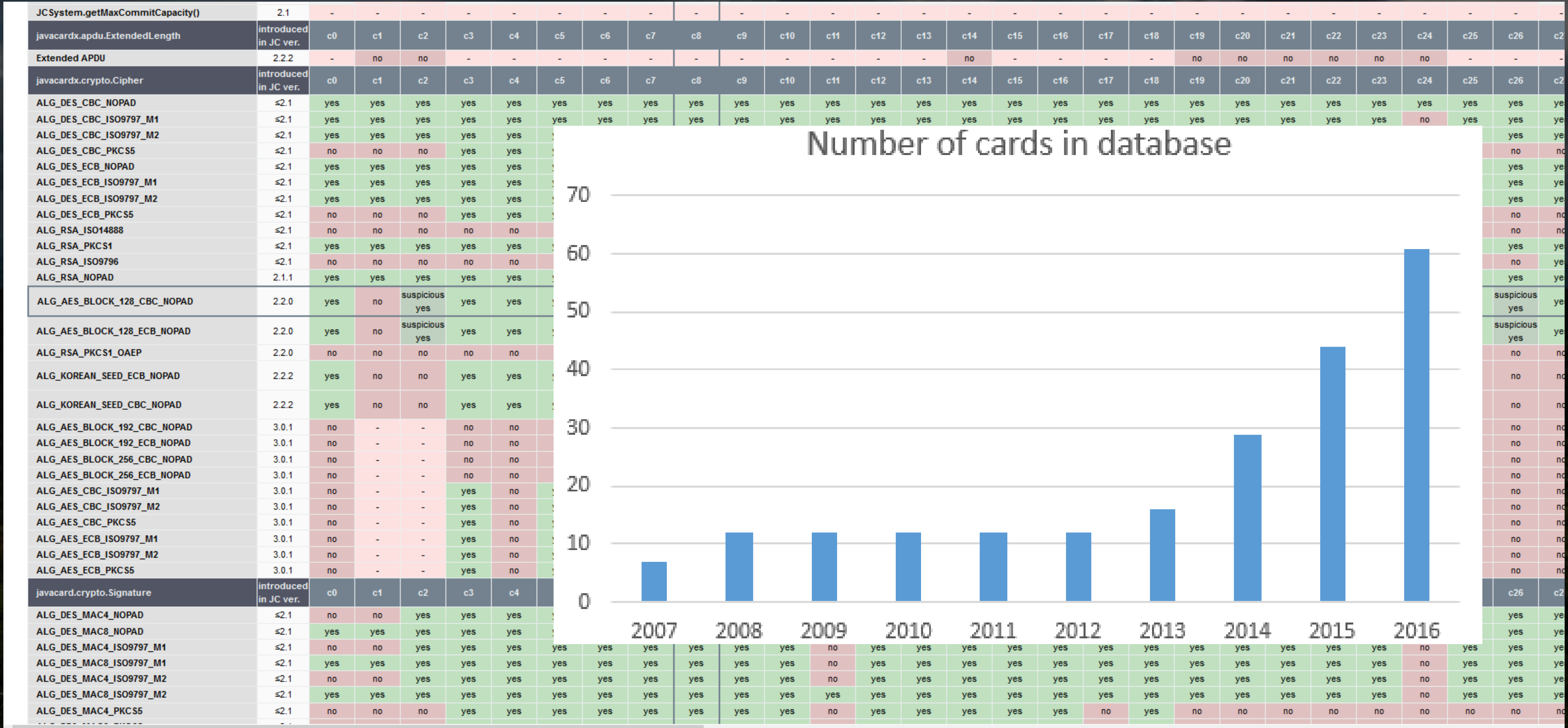
Profiler

- Speed optimization of on-card code notoriously difficult
- No free performance profiler available
- **OpencryptoJC.org/JCProfiler**

How-to:

1. Insert generic performance "traps" into source-code
2. Run automatic processor to create helper code for analysis
3. The profiler executes the target operation multiple times
4. Annotates the code with the measured timings

JCAlgTest.org



JCSystem.getMaxCommitCapacity()	2.1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
javacardx.apdu.ExtendedLength	introduced in JC ver.	c0	c1	c2	c3	c4	c5	c6	c7	c8	c9	c10	c11	c12	c13	c14	c15	c16	c17	c18	c19	c20	c21	c22	c23	c24	c25	c26
Extended APDU	2.2.2	-	no	no	-	-	-	-	-	-	-	-	-	-	-	no	-	-	-	-	no	no	no	no	no	no	-	-
javacardx.crypto.Cipher	introduced in JC ver.	c0	c1	c2	c3	c4	c5	c6	c7	c8	c9	c10	c11	c12	c13	c14	c15	c16	c17	c18	c19	c20	c21	c22	c23	c24	c25	c26
ALG_DES_CBC_NOPAD	≤2.1	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes
ALG_DES_CBC_ISO9797_M1	≤2.1	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	no	yes	yes
ALG_DES_CBC_ISO9797_M2	≤2.1	yes	yes	yes	yes	yes																						yes
ALG_DES_CBC_PKCS5	≤2.1	no	no	no	yes	yes																						no
ALG_DES_ECB_NOPAD	≤2.1	yes	yes	yes	yes	yes																						yes
ALG_DES_ECB_ISO9797_M1	≤2.1	yes	yes	yes	yes	yes																						yes
ALG_DES_ECB_ISO9797_M2	≤2.1	yes	yes	yes	yes	yes																						yes
ALG_DES_ECB_PKCS5	≤2.1	no	no	no	yes	yes																						no
ALG_RSA_ISO14888	≤2.1	no	no	no	no	no																						no
ALG_RSA_PKCS1	≤2.1	yes	yes	yes	yes	yes																						yes
ALG_RSA_ISO9796	≤2.1	no	no	no	no	no																						no
ALG_RSA_NOPAD	2.1.1	yes	yes	yes	yes	yes																						yes
ALG_AES_BLOCK_128_CBC_NOPAD	2.2.0	yes	no	suspicious yes	yes	yes																						suspicious yes
ALG_AES_BLOCK_128_ECB_NOPAD	2.2.0	yes	no	suspicious yes	yes	yes																						suspicious yes
ALG_RSA_PKCS1_OAEP	2.2.0	no	no	no	no	no																						no
ALG_KOREAN_SEED_ECB_NOPAD	2.2.2	yes	no	no	yes	yes																						no
ALG_KOREAN_SEED_CBC_NOPAD	2.2.2	yes	no	no	yes	yes																						no
ALG_AES_BLOCK_192_CBC_NOPAD	3.0.1	no	-	-	no	no																						no
ALG_AES_BLOCK_192_ECB_NOPAD	3.0.1	no	-	-	no	no																						no
ALG_AES_BLOCK_256_CBC_NOPAD	3.0.1	no	-	-	no	no																						no
ALG_AES_BLOCK_256_ECB_NOPAD	3.0.1	no	-	-	no	no																						no
ALG_AES_CBC_ISO9797_M1	3.0.1	no	-	-	yes	no																						no
ALG_AES_CBC_ISO9797_M2	3.0.1	no	-	-	yes	no																						no
ALG_AES_CBC_PKCS5	3.0.1	no	-	-	yes	no																						no
ALG_AES_ECB_ISO9797_M1	3.0.1	no	-	-	yes	no																						no
ALG_AES_ECB_ISO9797_M2	3.0.1	no	-	-	yes	no																						no
ALG_AES_ECB_PKCS5	3.0.1	no	-	-	yes	no																						no
javacard.crypto.Signature	introduced in JC ver.	c0	c1	c2	c3	c4																						c26
ALG_DES_MAC4_NOPAD	≤2.1	no	no	yes	yes	yes																						yes
ALG_DES_MAC8_NOPAD	≤2.1	yes	yes	yes	yes	yes																						yes
ALG_DES_MAC4_ISO9797_M1	≤2.1	no	no	yes	yes	yes	yes	yes	yes	yes	yes	yes	no	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	no	yes	yes
ALG_DES_MAC8_ISO9797_M1	≤2.1	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	no	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	no	yes	yes
ALG_DES_MAC4_ISO9797_M2	≤2.1	no	no	yes	yes	yes	yes	yes	yes	yes	yes	yes	no	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	no	yes	yes
ALG_DES_MAC8_ISO9797_M2	≤2.1	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	no	yes	yes
ALG_DES_MAC4_PKCS5	≤2.1	no	no	no	yes	yes	yes	yes	yes	yes	yes	yes	no	yes	yes	yes	yes	yes	no	yes	no	no	no	no	no	no	no	no

Development Cycle

1. Find a suitable card using our coverage table
2. Code using Eclipse, Netbeans, IntelliJ IDEA
3. Debugging using JCardSim simulator
4. Applet is built using Maven, ant-javacard scripts
5. Upload to real card using GlobalPlatformPro
6. Performance Profiling

More to come...

- Semi-automated porting to JavaCard
  - → JavaCard even more accessible to Java devs (e.g., IoT)
  - → Endpoint security: Java crypto code safer if run in smartcards

- JCCrypto Lib
  - → A collection of crypto algorithm implementations
  - → No 3-year lag anymore
  - → *Call for contributions!*

Takeaways

1. JCMath Library
    - → Developers now free to build
    - → Examples & Documentation
2. Performance Profiler
3. JC API Coverage & Performance Survey
    - → 60+ Cards
    - → 230 Algorithms

OpenCryptoJC.org

Q & A

OpenCrypto

Unchaining the JavaCard Ecosystem

OpenCryptoJC.org