
OpenCrypto
Unchaining the
JavaCard Ecosystem
https://boucycrypto.com

Who we are

Vasilios Mavroudis
Doctoral Researcher, UCL

George Danezis
Professor, UCL

Petr Svenda
Assistant Professor, MUNI
Co-founder, Enigma Bridge

Dan Cvrcek
Co-founder, Enigma Bridge

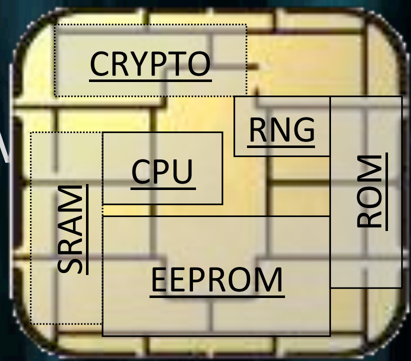
Contents

SmartCards

- GSM SIM modules
- Digital signatures
- Bank payment card (EMV standard)
- System authentication
- Operations authorizations
- ePassports
- Secure storage and encryption devic

The Hardware

- 8-32 bit processor @ 10+MHz
- Persistent memory 32-150kB (EEPROM
- Volatile fast RAM, usually <<10kB
- Truly Random Number Generator
- Cryptographic Coprocessor (3DES,AES,RSA-2048,...)
- Limited interface, small trusted computing base



The Hardware

Intended for physically unprotected environment

- NIST FIPS140-2 standard, Level 4
- Common Criteria EAL4+/5+/6

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

Why we like smartcards

- Number of vendors and independent HW platforms
- High-level of security (CC EAL5+, FIPS 140-2)
- Secure memory and storage
- Fast cryptographic coprocessor
- Programmable secure execution environment
- High-quality and very fast RNG
- On-card asymmetric key generation

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 Specification
- 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

The API Specification

- Encryption, authentication, & other algorithms
- Ensures interoperability
- JC straightforward to use
- Implementations are certified for functionality and security

A full ecosystem with laboratories & certification authorities

A success!

20 Billion Java Cards sold in total

3 Billion Javacards sold per year

1 Billion contactless cards in 2016

Common Use Cases:

- Telecommunications
- Payments
- Loyalty Cards

JavaCardForum

Timeline

- **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

- 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

- Vendors implement a subset of the Java Card specification:
  - No list of algorithms supported
  - The specific card must be tested

Bad Omens II

Three years late

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

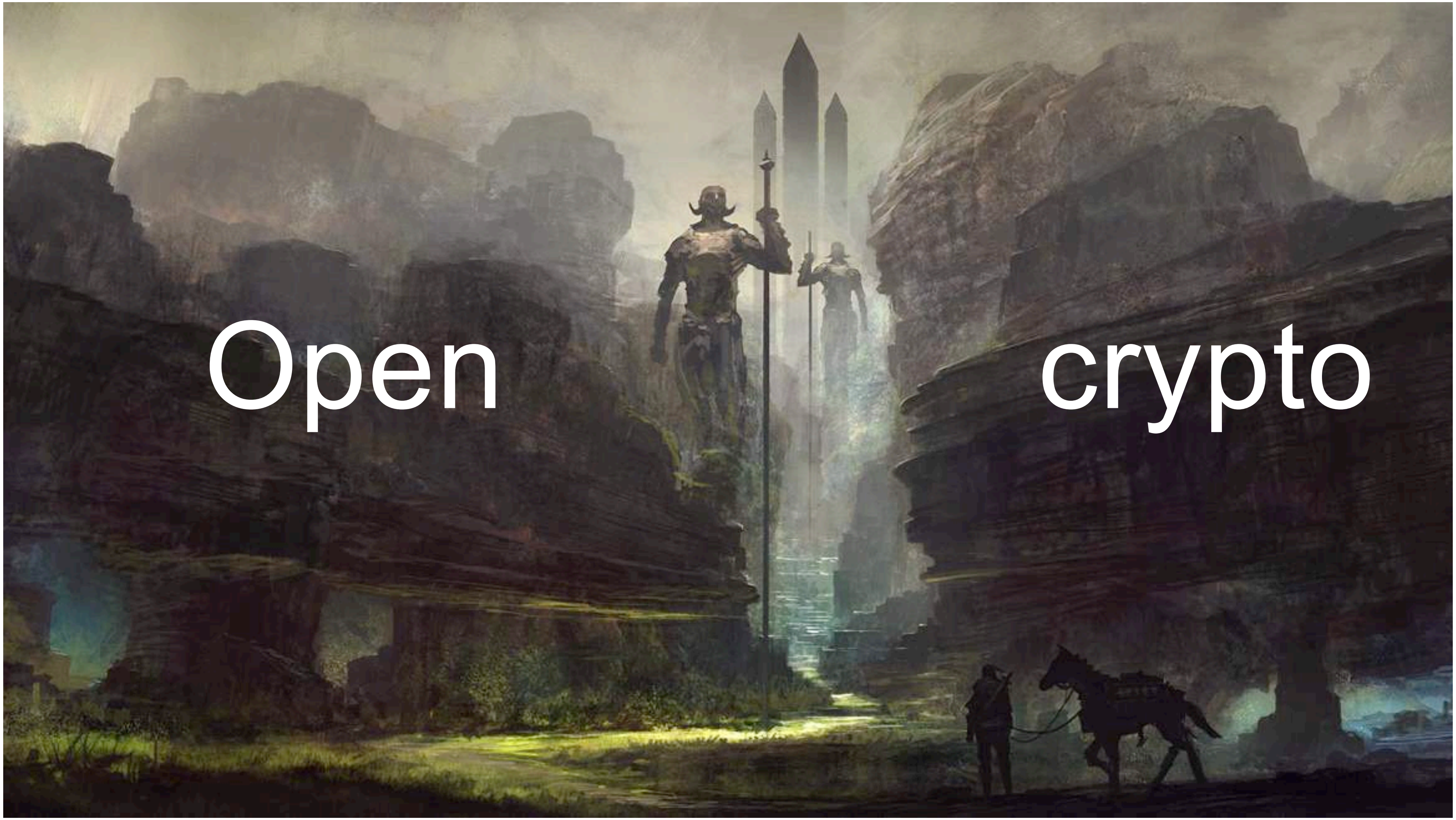
Interoperability

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

Walled Gardens

Proprietary APIs

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


Open
crypto

Motivation

- Technology moves increasingly fast, 3 years is a long time
- Patchy coverage of the latest crypto algorithms
- in-the-spec ≠ in-the-market

A new landscape:

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

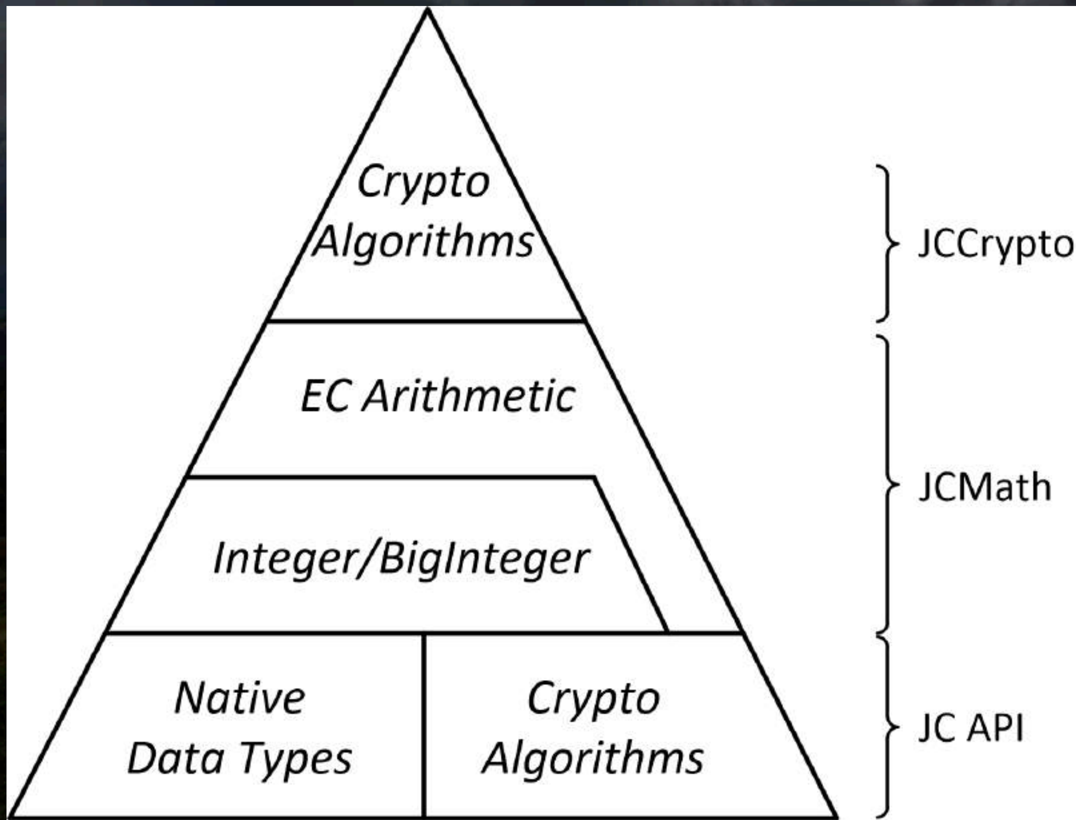
Things People Already Built!

- 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?

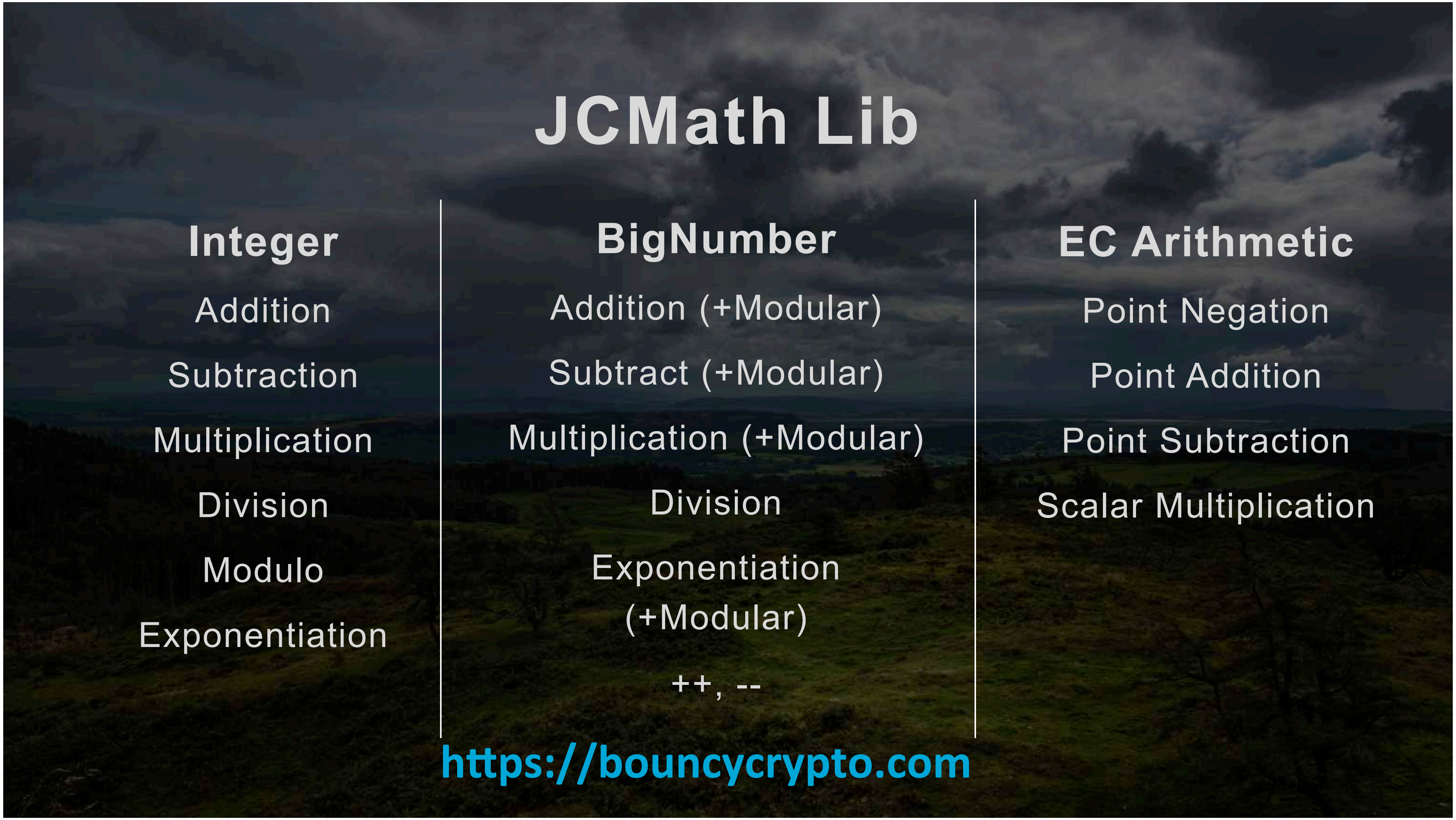
List of JavaCard open-source apps: https://github.com/EnigmaBridge/javacard-curated-list

The OpenCrypto Project



JCMath Lib

Class	Java	JC Spec.	JC Reality	JCMathLib
Integers	✔	✔	✘	✔
BigNumber	✔	✔	✘	✔
EC Curve	✔	~	~	✔
EC Point	✔	✘	✘	✔


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
https://bouncycrypto.com

```java
package opencrypto.jcmathlib;
public class ECExample extends javacard.framework.Applet {
    final static byte[] ECPOINT = {(byte)0x04, (byte) 0x3B... };
    final static byte[] SCALAR = {(byte) 0xE8, (byte) 0x05... };

    MLConfig       mlc;
    ECCurve        curve;
    ECPoint        point1, point2;

    public ECExample() {
        // Pre-allocate all helper structures
        mlc = new MLConfig((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, mlc);
        point1 = new ECPoint(curve, mlc);
        point2 = new ECPoint(curve, mlc);
    }

...
```

```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);
}
```

Building the Building Blocks

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

Coprocessor is fast! → No direct access ✘

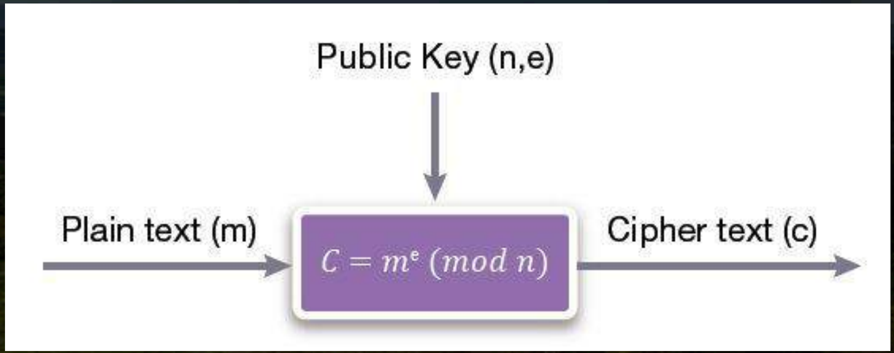
Hybrid solution

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

Simple Example

Modular Exponentiation with Big Numbers

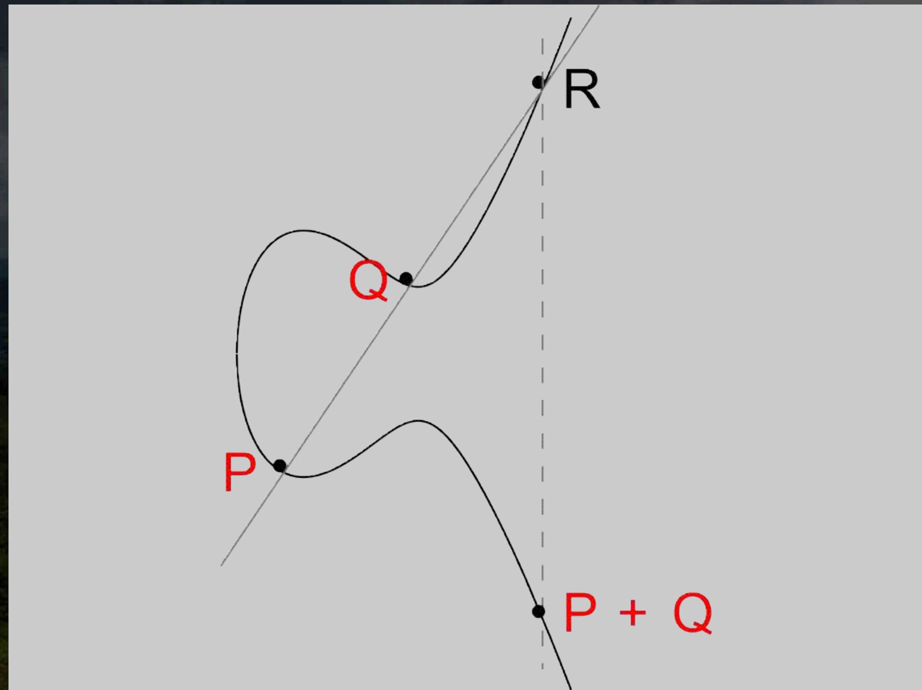
- Very slow to run on the CPU
- Any relevant calls in the API?
  - → RSA Encryption ✔
  - → Uses the coprocessor ✔
  - → Limitations on the modulo size ✘
  - → *Modulo* on CPU has decent speed ✔



EC Point-scalar multiplication

Elliptic Curves in 30 seconds:

- P, Q are elliptic curve points
- Each point has coordinates (x,y)
- P+Q: Just draw two lines
- P+P: Very similar
- P+P = 2P
- What about 3P, 4P, 1000P?

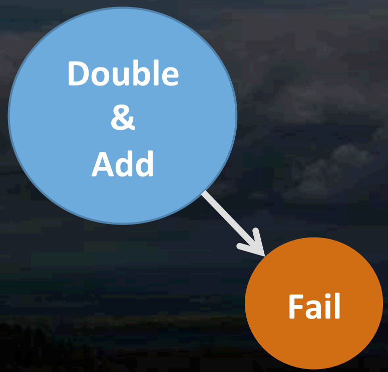


EC Point-scalar multiplication

Multiplication (5 times P) as:

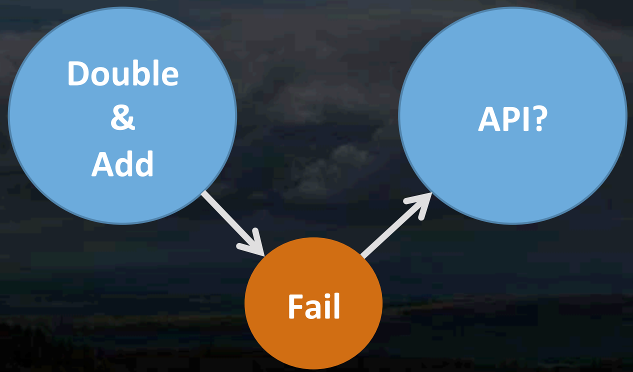
- Additions → 5P = P+P+P+P+P (5 operations)
- Additions and Doublings → 5P= 2P + 2P + P (3 operations)
- A smarter way → Double-n-Add Algorithm
  - ↪ Uses less additions and doublings

EC Point-scalar multiplication



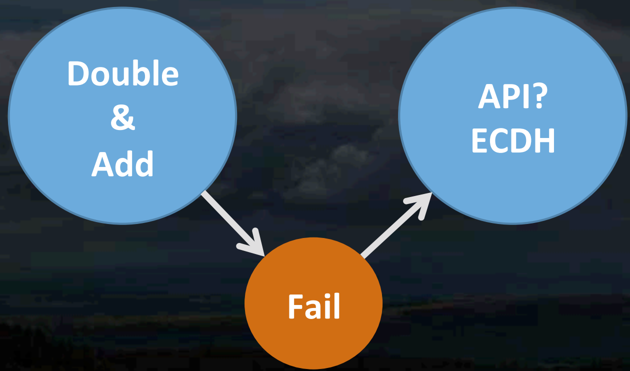
- Double & Add → Too many operations to use the CPU
- We need another operation that will use the coprocessor

EC Point-scalar multiplication



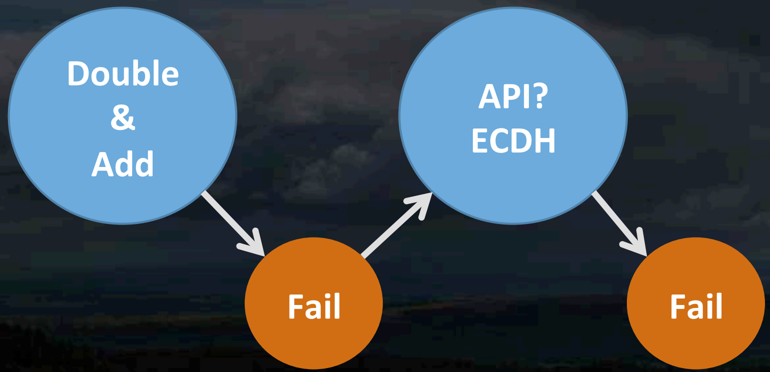
- Double & Add → Too many operations to use the CPU
- We need another operation that will use the coprocessor
- Back to the API specification…

EC Point-scalar multiplication



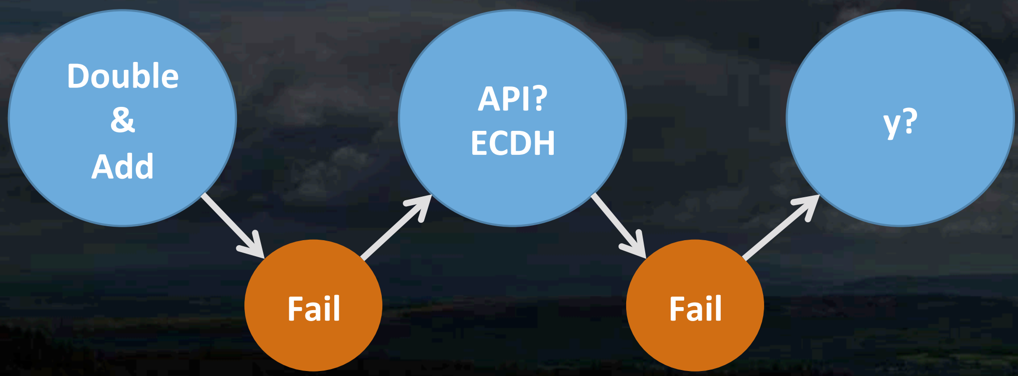
- Key agreement using ECDH ***is*** scalar multiplication!
- API Method: *ALG_EC_DH_PLAIN*
- Description: *Diffie-Hellman (DH) secret value derivation primitive as per NIST Special Publication 800-56Ar2.*

EC Point-scalar multiplication



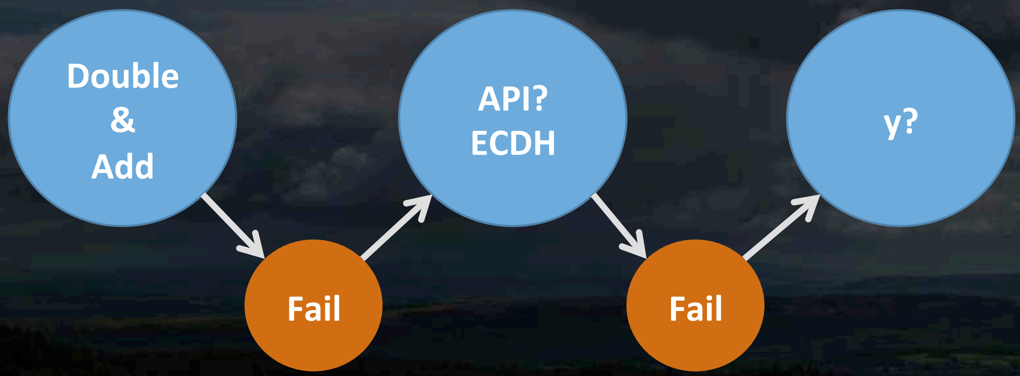
- In practice this means that the method returns only coordinate x.
- Remember: “Each point has coordinates (x,y)”
- We need y too.

EC Point-scalar multiplication



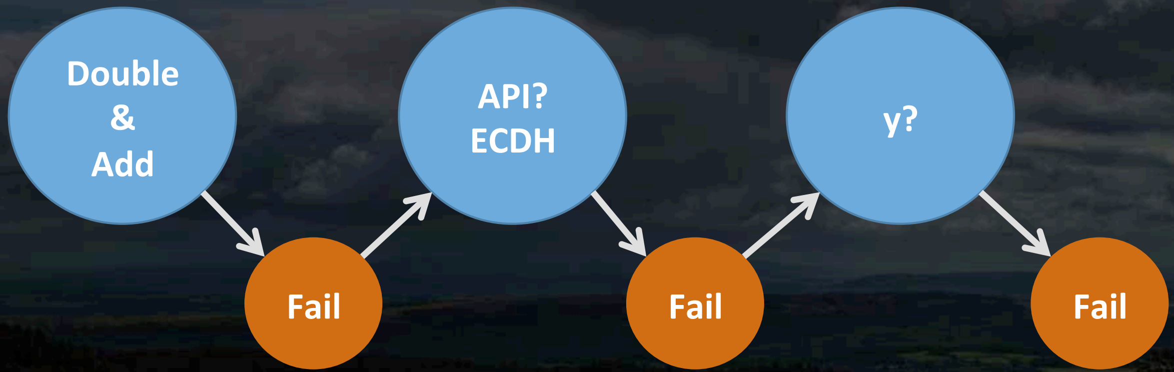
- Can we somehow infer y?
- EC formula: $y^2=x^3+Ax+B$
- We know all unknown variables except y!

EC Point-scalar multiplication

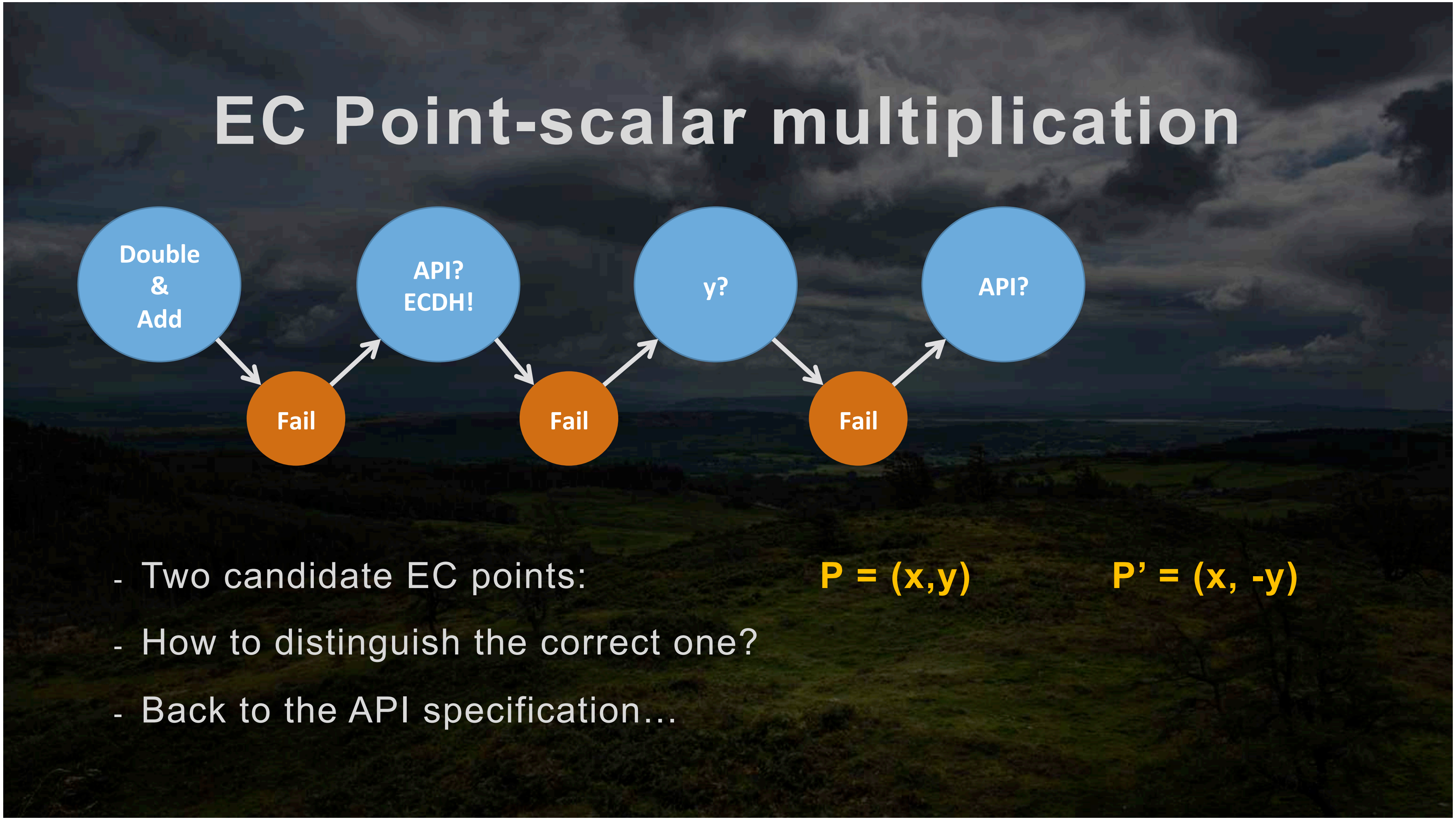


- EC formula: $y^2=x^3+Ax+B \rightarrow$ Compute y^2
- Then compute the square root of y^2
- This will give us $+y$, $-y$.
- But which one is the correct one?

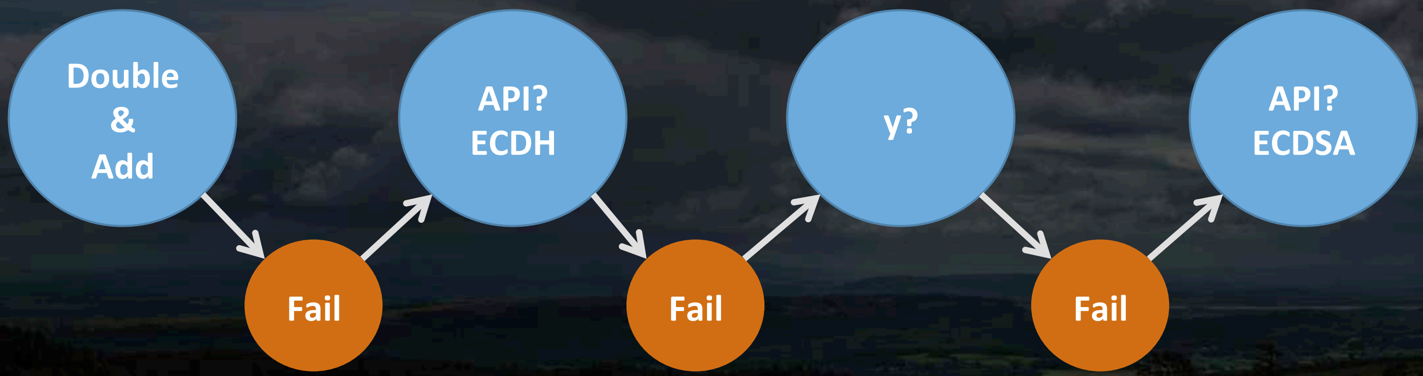
EC Point-scalar multiplication



- EC formula: $y^2=x^3+Ax+B$ → Compute y^2
- Then compute the square root of y^2
- This will give us $+y$, $-y$.
- But which one is the correct one? → No way to know!

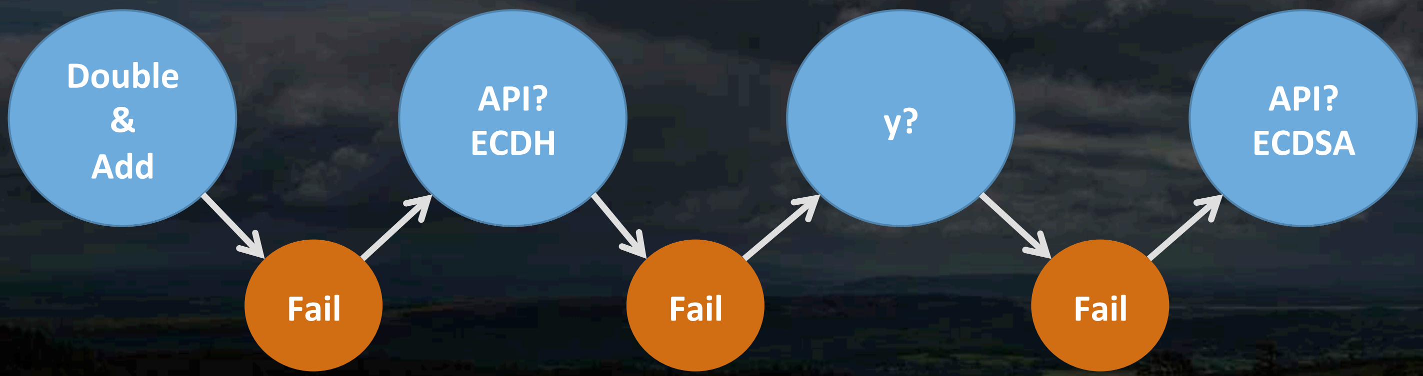

EC Point-scalar multiplication
Double & Add
Fail
API? ECDH!
Fail
y?
Fail
API?
- Two candidate EC points:
P = (x,y)
P' = (x, -y)
- How to distinguish the correct one?
- Back to the API specification…

EC Point-scalar multiplication



- Two candidate EC points: **P = (x,y)** **P’ = (x, -y)**
- How to distinguish the correct one?
- Let use ECDSA!

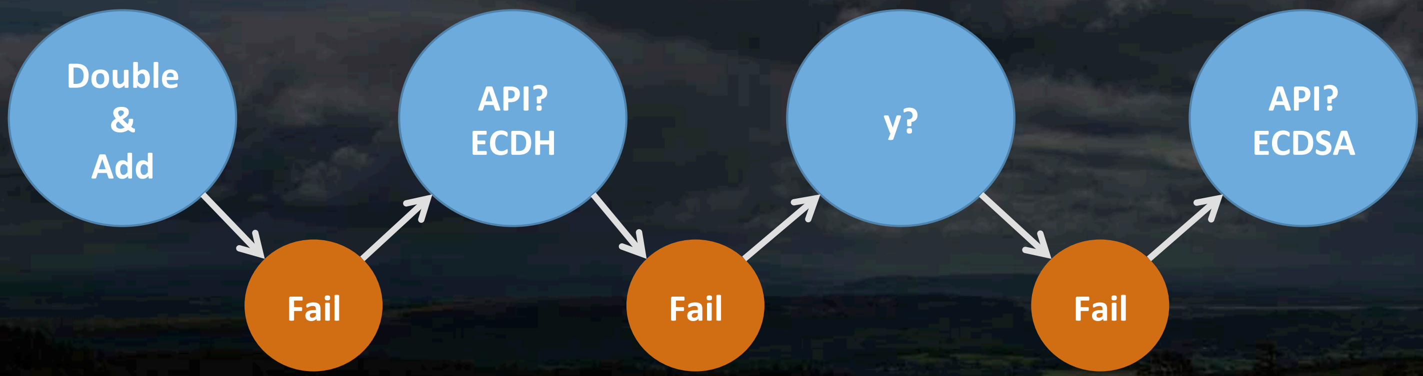
EC Point-scalar multiplication



- ECDSA uses:
  - → A private key to sign a plaintext.
  - → A public key to verify a signature.

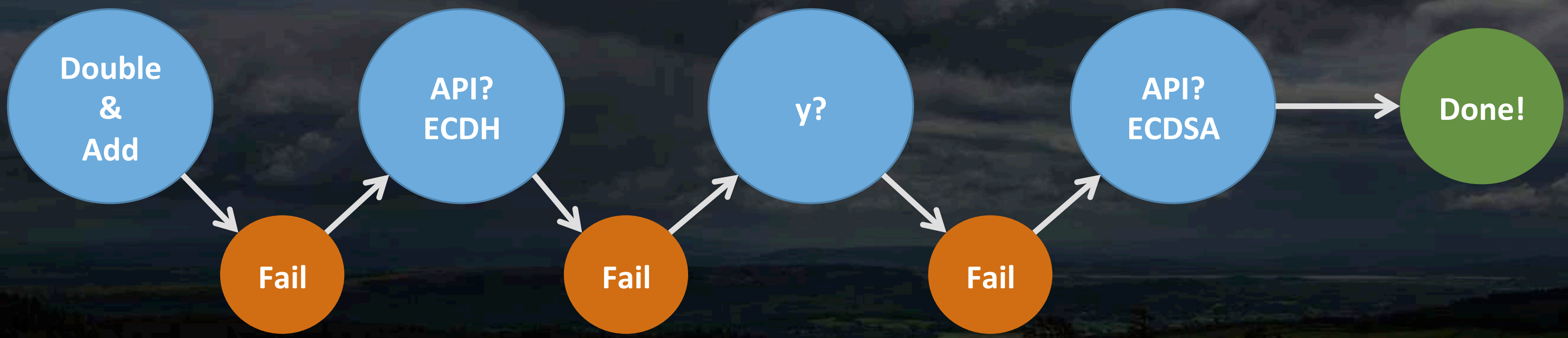
Two candidate EC points **P = (x,y) P’ = (x, -y)** and a scalar x

EC Point-scalar multiplication



- Two candidate EC points **P = (x,y) P' = (x, -y)** and a scalar x
- ECDSA abuse:
  - → A private key to sign a plaintext ← **This is our scalar**
  - → A public key to verify a signature. ← **This is our P and P'**
-

EC Point-scalar multiplication



- Two candidate EC points **P = (x,y) P' = (x, -y)** and a scalar x
- ECDSA abuse:
  → A private key to sign a plaintext ← **This is our scalar**
  → A public key to verify a signature. ← **This is our P and P'**
- Then try to verify with the two points and see which one it is. ■

EC Point-scalar multiplication

The full algorithm

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'

JCMathLib Performance

Depends on

- The card's processor
- The algorithms the card supports
  - → E.g., if card supports ALG_EC_SVDP_DH_PLAIN_XY (3.0.5) native speed
  - → Else we have to use ALG_EC_SVDP_DH_PLAIN and be slower

Measurements

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

JCMathLib Performance

Measurements

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

JCMathLib Convenience Features

- We take care of the low-level/dirty stuff:
  - Unified memory management of shared objects
  - Safe reuse of pre-allocated arrays with locking and automated erasure
  - Adapt placement of data in RAM or EEPROM for optimal performance

- Supports both physical cards and simulators
  - JCardSim pull requests

Profiler

- Speed optimization of on-card code notoriously difficult
- No free performance profiler available

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
5. Bonus: Helps to detect where exactly generic exceptions occur

Performance profiler

```
private short multiplication_x_KA(Bignat scalar, byte[] outBuffer, short outBufferOffset) {
      PM.check(PM.TRAP_ECPOINT_MULT_X_1); // 40 ms (gd60,1500968219581)
      priv.setS(scalar.as_byte_array(), (short) 0, scalar.length());
      PM.check(PM.TRAP_ECPOINT_MULT_X_2); // 12 ms (gd60,1500968219581)

      keyAgreement.init(priv);
      PM.check(PM.TRAP_ECPOINT_MULT_X_3); // 120 ms (gd60,1500968219581)

      short len = this.getW(point_arr1, (short) 0);
      PM.check(PM.TRAP_ECPOINT_MULT_X_4); // 9 ms (gd60,1500968219581)
      len = keyAgreement.generateSecret(point_arr1, (short) 0, len, outBuffer, outBufferOffset);
      PM.check(PM.TRAP_ECPOINT_MULT_X_5); // 186 ms (gd60,1500968219581)

      return COORD_SIZE;
}
```

Toolchain

1. Code using standard Java dev tools (any IDE + javac)
2. Code is debugged JCardSim simul
3. Communication with card using standard javax.smartcardio.*
4. Applet is converted using ant-javacard scr
5. Upload to real card using GlobalPlatformPro
6. Find a suitable card using the table in *jcalgtest.org*

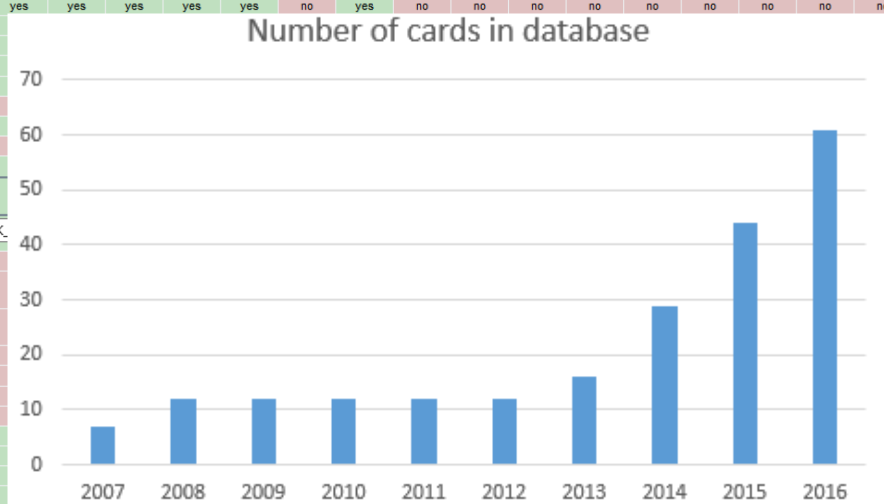
How to start with JavaCard for Java developers

1. Download BouncyCrypto / JCMathAlg from GitHub
2. Use examples and Maven/ant scripts to build them
3. Start using JavaCards and test with JCardSim simulator
4. You may use cloud JavaCards – more info in GitHub soon
5. You buy some real JavaCards
6. Use available scripts in the BouncyCrypto repo
7. Deploy as needed

JCAlgTest.org – a large project analyzing capabilities of JavaCards

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
Extended APDU	2.2.2	-	no	no	-	-	-	-	-	-	-	-	-	-	-
javacardx.crypto.Cipher	introduced in JC ver.	c0	c1	c2	c3	c4	c5	c6	c7	c8	c9	c10	c11	c12	c13
ALG_DES_CBC_NOPAD	≤2.1	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
ALG_DES_CBC_ISO9797_M2	≤2.1	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes
ALG_DES_CBC_PKCS5	≤2.1	no	no	no	yes	yes	yes	yes	yes	yes	yes	yes	no	yes	yes
ALG_DES_ECB_NOPAD	≤2.1	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes		
ALG_DES_ECB_ISO9797_M1	≤2.1	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes		
ALG_DES_ECB_ISO9797_M2	≤2.1	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes		
ALG_DES_ECB_PKCS5	≤2.1	no	no	no	yes	yes	yes	yes	yes	yes	yes	yes	no		
ALG_RSA_ISO14888	≤2.1	no	no	no	no	no	no	no	no	no	no	no	no		
ALG_RSA_PKCS1	≤2.1	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes		
ALG_RSA_ISO9796	≤2.1	no	no	no	no	no	no	no	no	no	no	no	no		
ALG_RSA_NOPAD	2.1.1	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes		
ALG_AES_BLOCK_128_CBC_NOPAD	2.2.0	yes	no	suspicious yes	yes	yes	yes	yes	yes	yes	yes	yes	yes		
ALG_AES_BLOCK_128_ECB_NOPAD	2.2.0	yes	no	suspicious yes	yes	yes	yes	yes	yes	y Feitian JavaCOS A22 : ALG_AES_BLOCK_					
ALG_RSA_PKCS1_OAEP	2.2.0	no	no	no	no	no	no	no	no	no	no	no	no		
ALG_KOREAN_SEED_ECB_NOPAD	2.2.2	yes	no	no	yes	yes	yes	yes	yes	yes	yes	no	no		
ALG_KOREAN_SEED_CBC_NOPAD	2.2.2	yes	no	no	yes	yes	yes	yes	yes	yes	yes	no	no		
ALG_AES_BLOCK_192_CBC_NOPAD	3.0.1	no	-	-	no	no	no	no	no	no	no	no	no		
ALG_AES_BLOCK_192_ECB_NOPAD	3.0.1	no	-	-	no	no	no	no	no	no	no	no	no		
ALG_AES_BLOCK_256_CBC_NOPAD	3.0.1	no	-	-	no	no	no	no	no	no	no	no	no		
ALG_AES_BLOCK_256_ECB_NOPAD	3.0.1	no	-	-	no	no	no	no	no	no	no	no	no		
ALG_AES_CBC_ISO9797_M1	3.0.1	no	-	-	yes	no	yes	yes	yes	yes	yes	yes	no		
ALG_AES_CBC_ISO9797_M2	3.0.1	no	-	-	yes	no	yes	yes	yes	yes	yes	yes	no		
ALG_AES_CBC_PKCS5	3.0.1	no	-	-	yes	no	yes	yes	yes	yes	yes	yes	no		
ALG_AES_ECB_ISO9797_M1	3.0.1	no	-	-	yes	no	yes	yes	yes	yes	yes	yes	no		
ALG_AES_ECB_ISO9797_M2	3.0.1	no	-	-	yes	no	yes	yes	yes	yes	yes	yes	no	yes	yes
ALG_AES_ECB_PKCS5	3.0.1	no	-	-	yes	no	yes	yes	yes	yes	yes	yes	no	yes	yes
javacard.crypto.Signature	introduced in JC ver.	c0	c1	c2	c3	c4	c5	c6	c7	c8	c9	c10	c11	c12	c13
ALG_DES_MAC4_NOPAD	≤2.1	no	no	yes	yes	yes	yes	yes	yes	yes	yes	yes	no	yes	yes
ALG_DES_MAC8_NOPAD	≤2.1	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	no	yes	yes
ALG_DES_MAC4_ISO9797_M1	≤2.1	no	no	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
ALG_DES_MAC4_ISO9797_M2	≤2.1	no	no	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
ALG_DES_MAC4_PKCS5	≤2.1	no	no	no	yes	yes	yes	yes	yes	yes	yes	yes	no	yes	yes

JCSystem.getMaxCommitCapacity()	-	-	-	-	-	-	-	-	-	-	-	-	-	-
javacardx.apdu.ExtendedLength	c14	c15	c16	c17	c18	c19	c20	c21	c22	c23	c24	c25	c26	c2
Extended APDU	no	-	-	-	-	no	no	no	no	no	no	-	-	-
javacardx.crypto.Cipher	c14	c15	c16	c17	c18	c19	c20	c21	c22	c23	c24	c25	c26	c2
ALG_DES_CBC_NOPAD	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	ye
ALG_DES_CBC_ISO9797_M1	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	no	yes	yes	ye
ALG_DES_CBC_ISO9797_M2	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	no	yes	yes	ye
ALG_DES_CBC_PKCS5	yes	yes	yes	no	yes	no	no	no	no	no	no	no	no	no
ALG_AES_ECB_ISO9797_M2	-	no	no	no	no	-	-	-	-	-	-	no	no	no
ALG_AES_ECB_PKCS5	-	no	no	no	no	-	-	-	-	-	-	no	no	no
javacard.crypto.Signature	c14	c15	c16	c17	c18	c19	c20	c21	c22	c23	c24	c25	c26	c2
ALG_DES_MAC4_NOPAD	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	ye
ALG_DES_MAC8_NOPAD	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	ye
ALG_DES_MAC4_ISO9797_M1	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	no	yes	yes	ye
ALG_DES_MAC8_ISO9797_M1	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	no	yes	yes	ye
ALG_DES_MAC4_ISO9797_M2	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	no	yes	yes	ye
ALG_DES_MAC8_ISO9797_M2	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	no	yes	yes	ye
ALG_DES_MAC4_PKCS5	yes	yes	yes	no	yes	no	no	no	no	no	no	no	no	no



Takeaways

- JavaCards are an affordable and convenient way of separating security critical code and secrets
- You can use JavaCard code in local hardware, cloud JavaCards, or in simulators (local or listening on an TCP/IP port)
- JCMath Lib fills the gap in modern crypto algorithms - ECC
  - → Developers now free to build
  - → Examples & Documentation
  - → No 3-year lag anymore
- Profiler & Complete Toolchain
- Working on…

Takeaways

- JavaCards are an affordable and convenient way of separating security critical code and secrets
- You can use JavaCard code in local hardware, cloud JavaCards, or in simulators (local or listening on an TCP/IP port)
- JCMath Lib fills the gap in modern crypto algorithms - ECC
  - → Developers now free to build
  - → Examples & Documentation
  - → No 3-year lag anymore
- Profiler & Complete Toolchain
- Working on…
- Toolchain, examples, quick get-started integration scenarios and templates


Q & A


OpenCrypto
Unchaining the JavaCard Ecosystem
https://bouncycrypto.com

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