OpenCrypto

Unchaining the JavaCard Ecosystem

Who we are

Vasilios Mavroudis
Doctoral Researcher, UCL

Petr Svenda
Assistant Professor, MUNI

George Danezis
Professor, UCL

Dan Cvrcek
Founder, EnigmaBridge

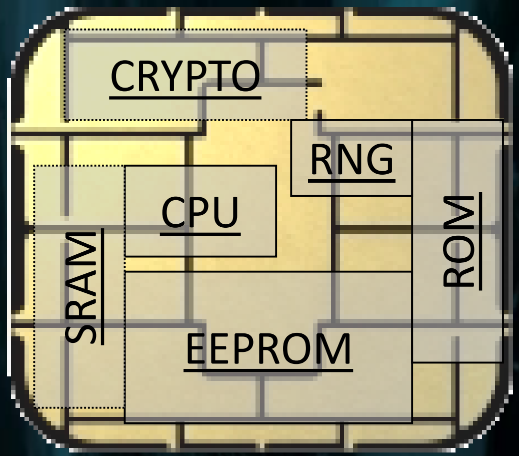
Contents

SmartCards

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

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

- 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
- And a compliance kit

Today, 20 years after:

- Oracle still releases the Java Card specifications
- and provides the SDK for applet development

An Omnipotent Specification

Defines the Java Card API:

- Straightforward to use
- Key encryption & authentication functions
- 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 for 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 devs 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:

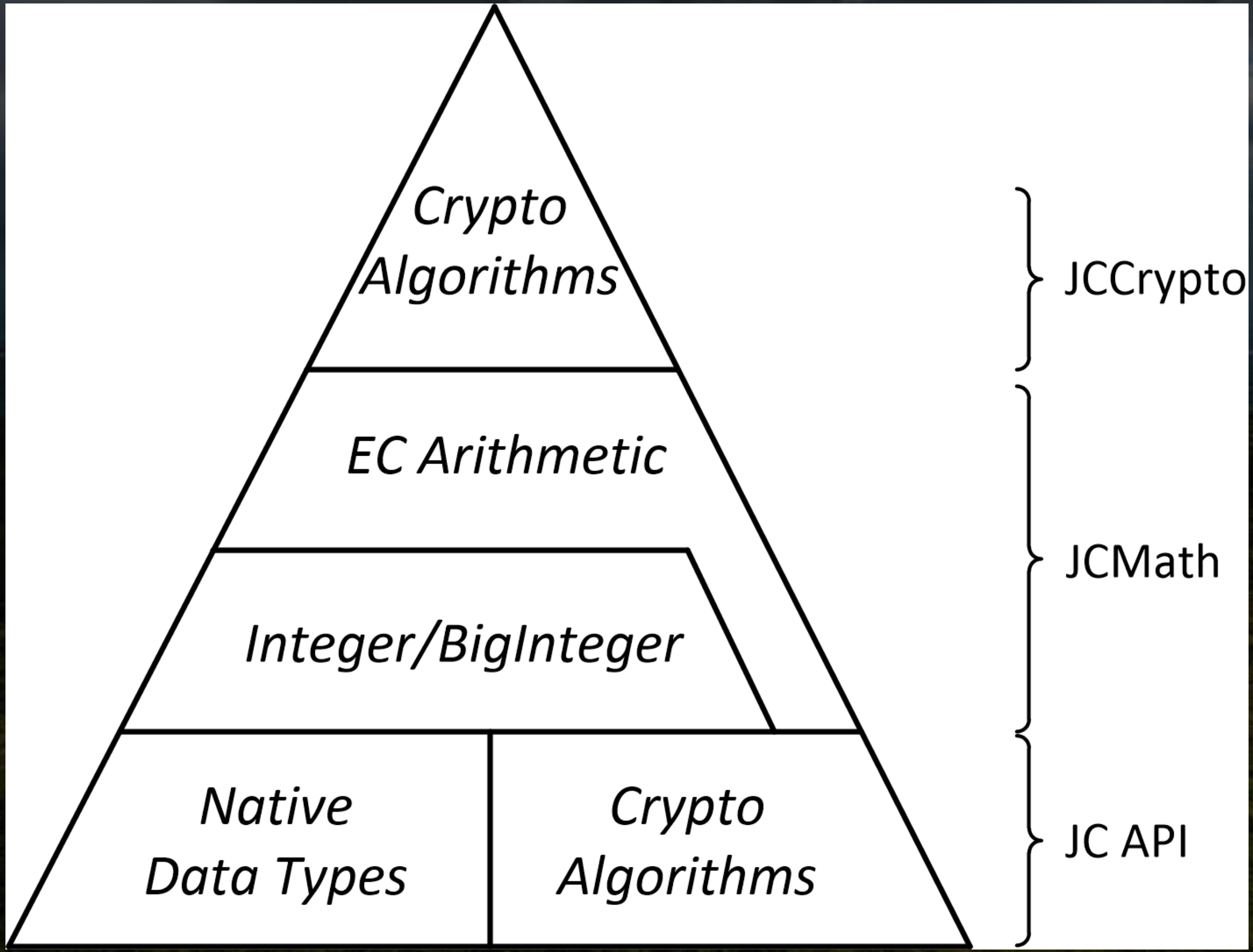
- IoT needs a platform with these characteristics
- Lots of small dev. houses
- Java devs in awe → No Integers, Primitive Garbage Collection
- People want to build new things!!

Things People Already Built!

- Store PGP private key so that it never leaves the card
- Bitcoin hardware wallet
- Generate one-time passwords
- Authenticate with 2 factors
- Store disk encryption keys
- SSH keys Protection

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

The OpenCrypto Project



In A Nutshell

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

JCMath

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

Building the Building Blocks

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

Coprocessor is fast! → No direct access ✗

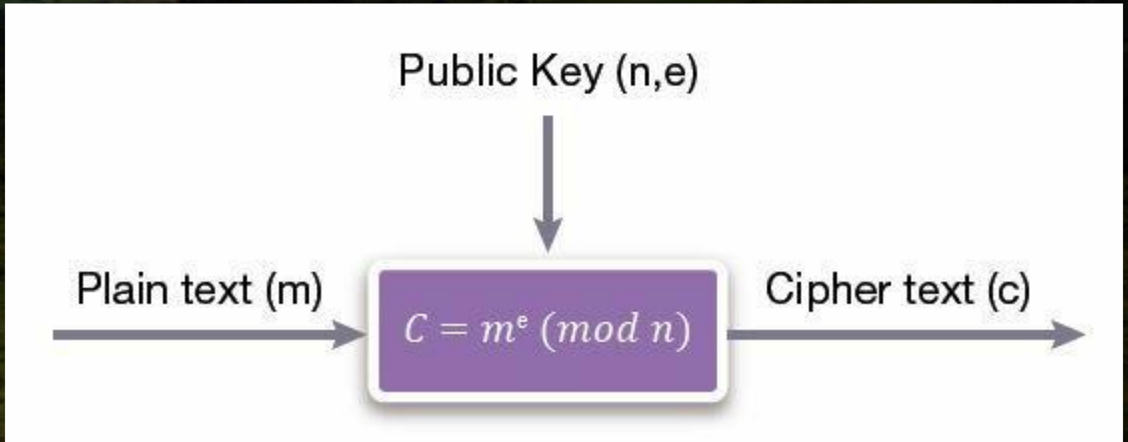
Hybrid solution

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

Simple Example

Modular Exponentiation with Big Numbers

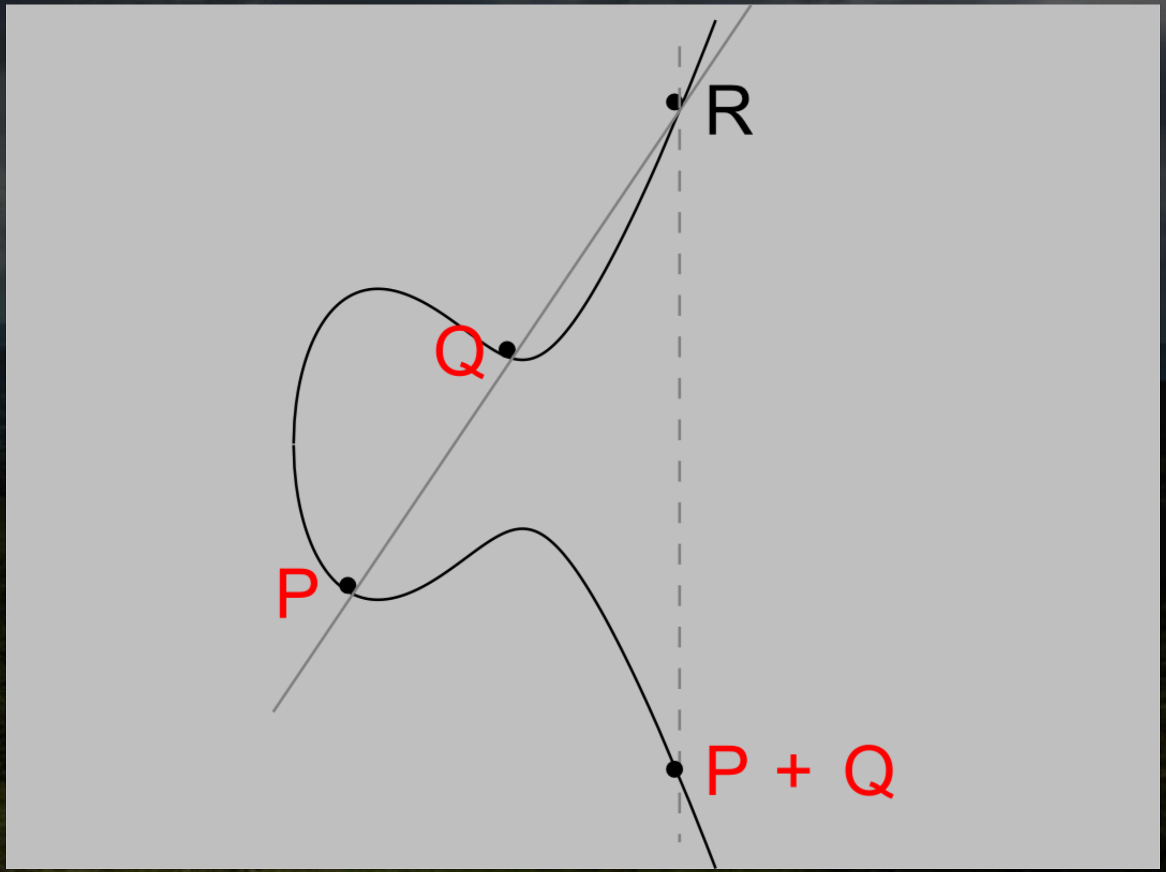
- Very slow to run on the CPU
- Any relevant calls in the API?
  - → RSA Encryption ✓
  - → Uses the coprocessor ✓
  - → Limitations on the modulo size ✗
  - → *Modulo* in CPU has decent speed ✓



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?

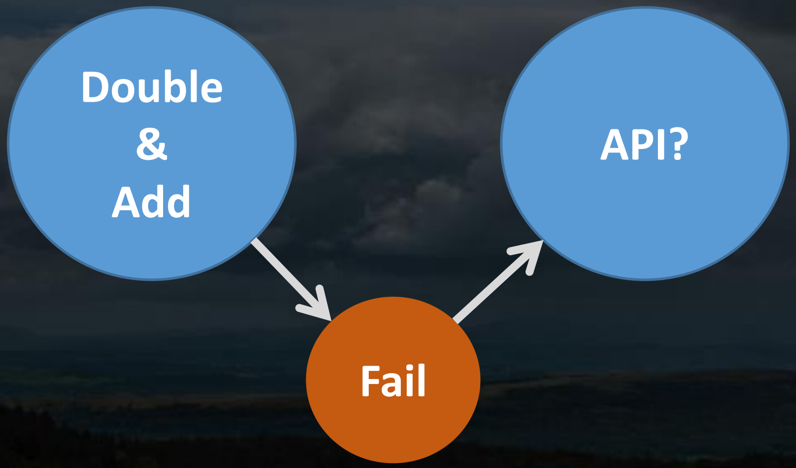


EC Point-scalar multiplication

Multiplication (5 times P) as:

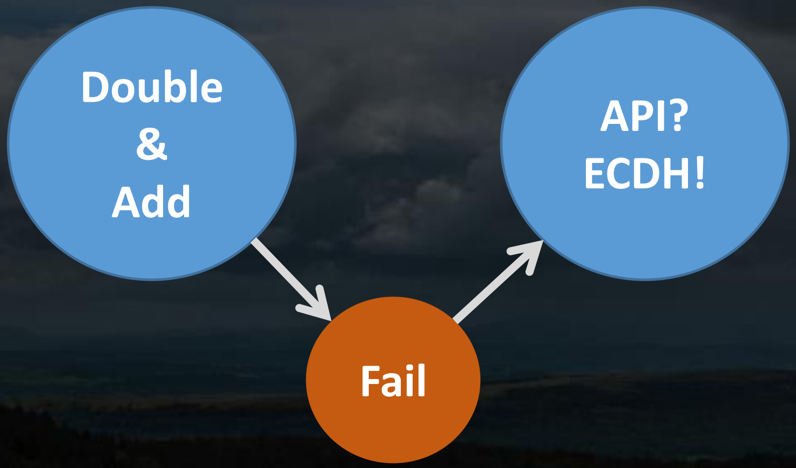
- Additions → 5P = P+P+P+P+P
- Additions and Doublings → 5P= 2P + 2P + P
- A smarter way → Double-n-Add Algorithm
  ↪ Uses less additions and doublings

EC Point-scalar multiplication



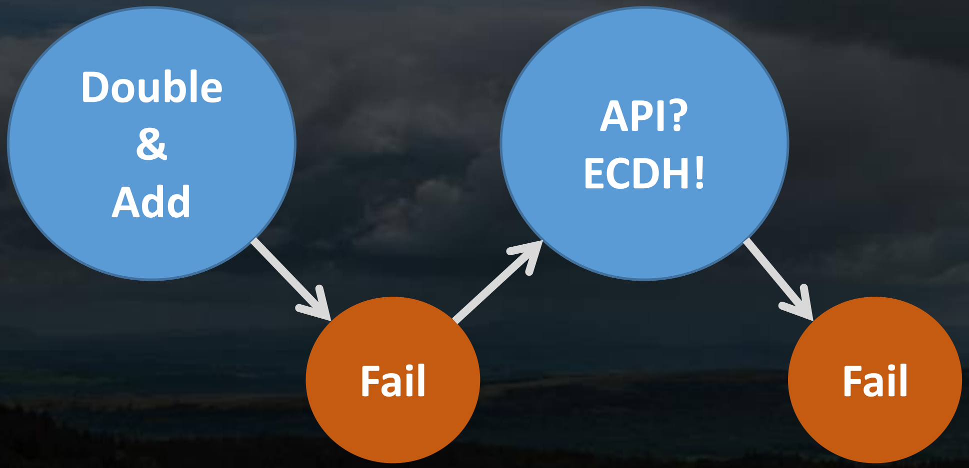
- Double & Add → Too many operations to use the CPU
- We need another operation that will use the coprocessor
- Back to the specification…

EC Point-scalar multiplication



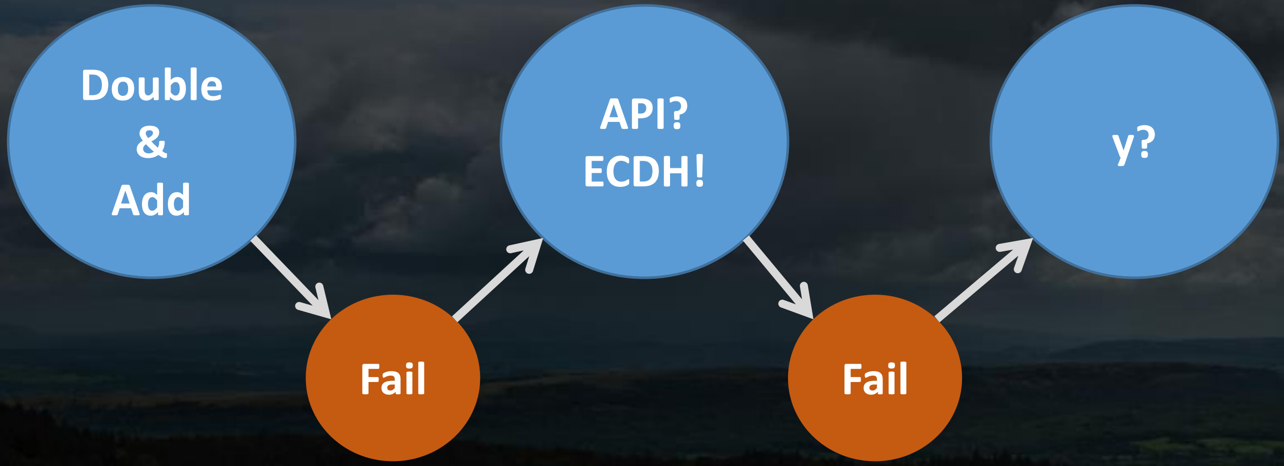
- Key agreement using ECDH ***is*** scalar multiplication!
- API Method: *ALG_EC_DH_PLAIN*
- Description: *Diffie-Hellman (DH) secret value derivation primitive as per NIST Special Publication 800-56Ar2.*

EC Point-scalar multiplication



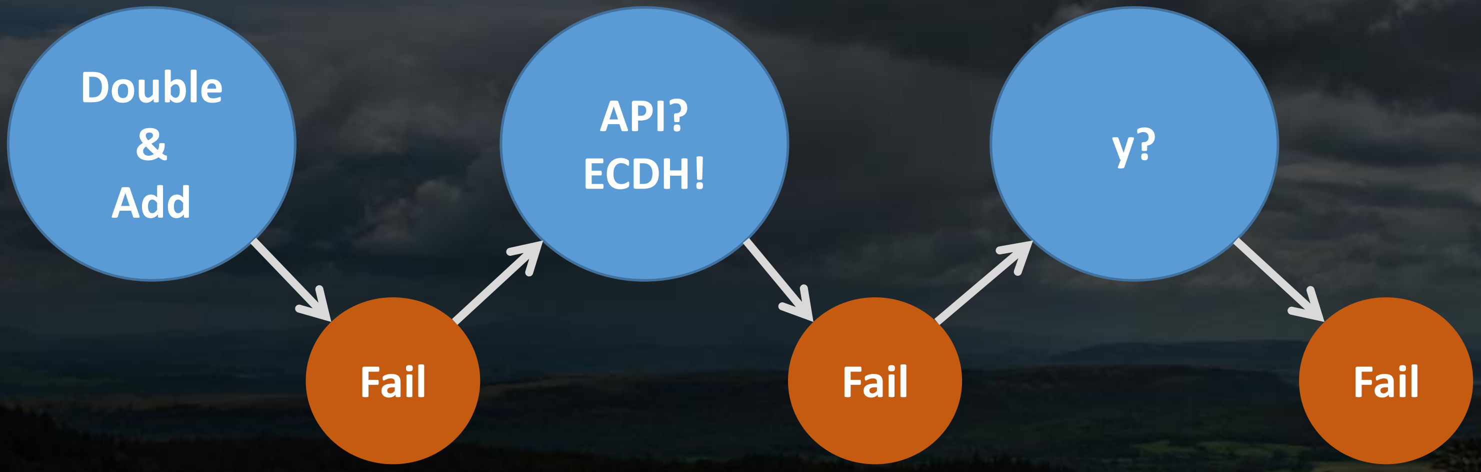
- In practice this means that the method returns only coordinate x.
- Remember: “Each point has coordinates (x,y)”
- We need y too.

EC Point-scalar multiplication



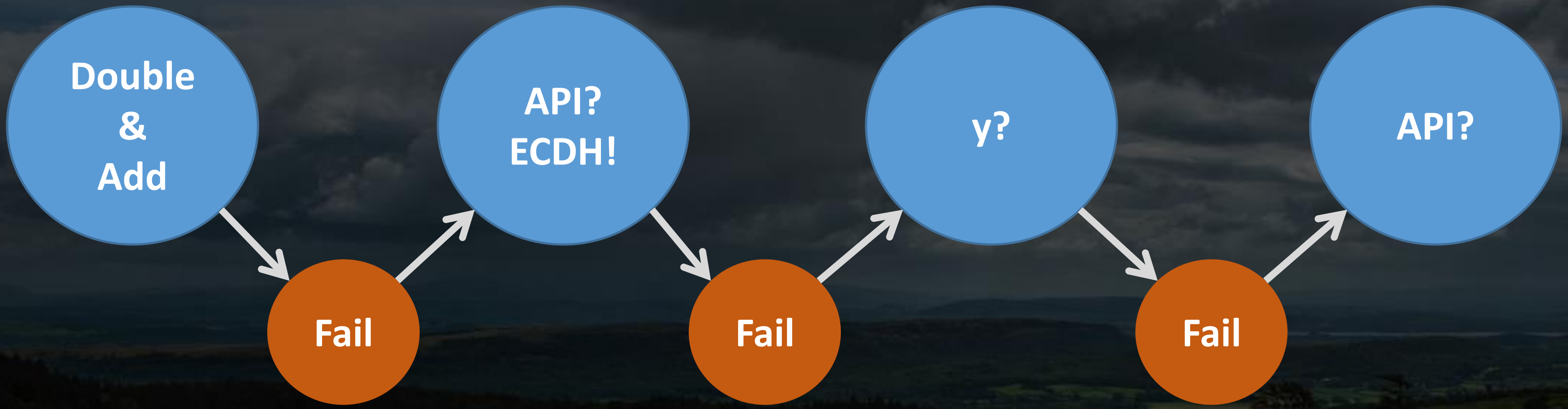
- Can we somehow infer y?
- EC formula: $y^2=x^3+Ax+B$
- We know everything except y!

EC Point-scalar multiplication



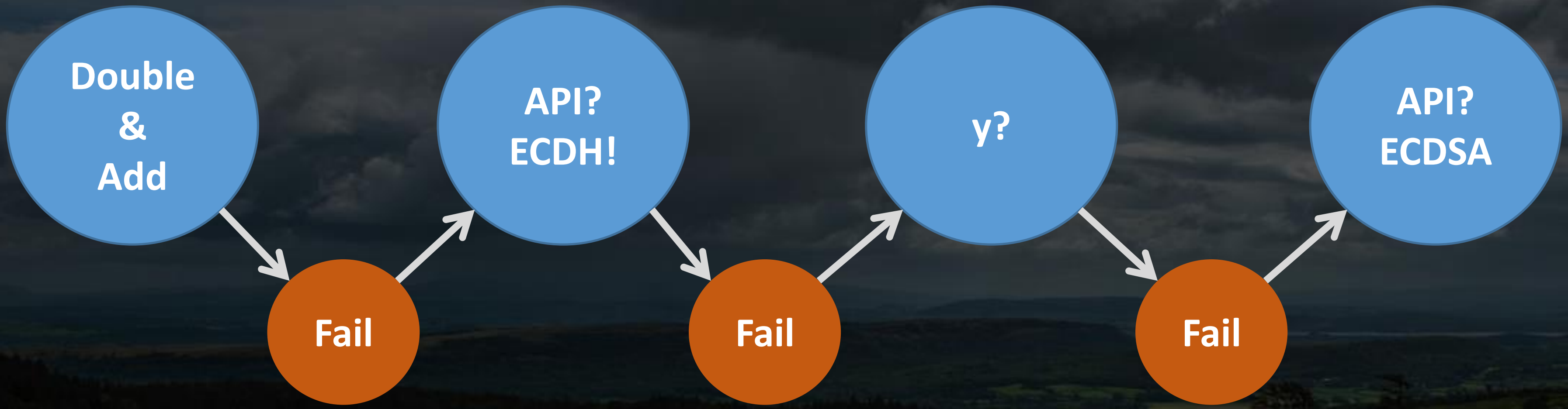
- EC formula: $y^2=x^3+Ax+B$ → Compute y^2
- Then compute the square root of y^2
- This will give us +y, -y.
- But which one is the correct one? → No way to know!

EC Point-scalar multiplication



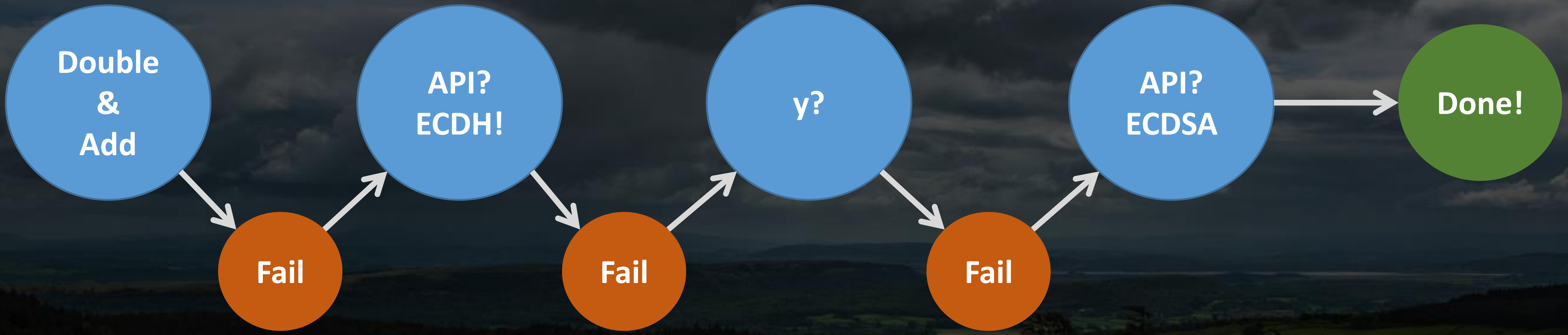
- Two candidate EC points: $P = (x,y)$ $P' = (x, -y)$
- How to distinguish the correct one?
- Back to the specification…

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



- Two candidate EC points **P = (x,y) P' = (x, -y)** and a scalar **x**
- ECDSA: Let's use scalar as a private key to sign a plaintext.
- Then try to verify with the two points and see which one it is. ■

EC Point-scalar multiplication

The full algorithm

1. Take scalar x and point P
2. Pretend doing an DH key exchange to get x (co-processor)
3. Compute the two candidate points P, P’ (CPU)
4. Sign with the scalar x (co-processor)
5. Try to verify with P (co-processor)
6. If it works → It’s P

   else → It’s P’
7. Return P or P’

Future: JCCrypto

- All the basic crypto operations are now available!
- Devs can implement crypto algorithms.
- Would be nice to have a collection of trusted implementations.

Anyone can help!


Q & A


OpenCrypto
Unchaining the
JavaCard Ecosystem