The adoption rate of JavaCard features by certified products and open-source projects

Lukáš Zaoral, Antonín Dufka, Petr Švenda

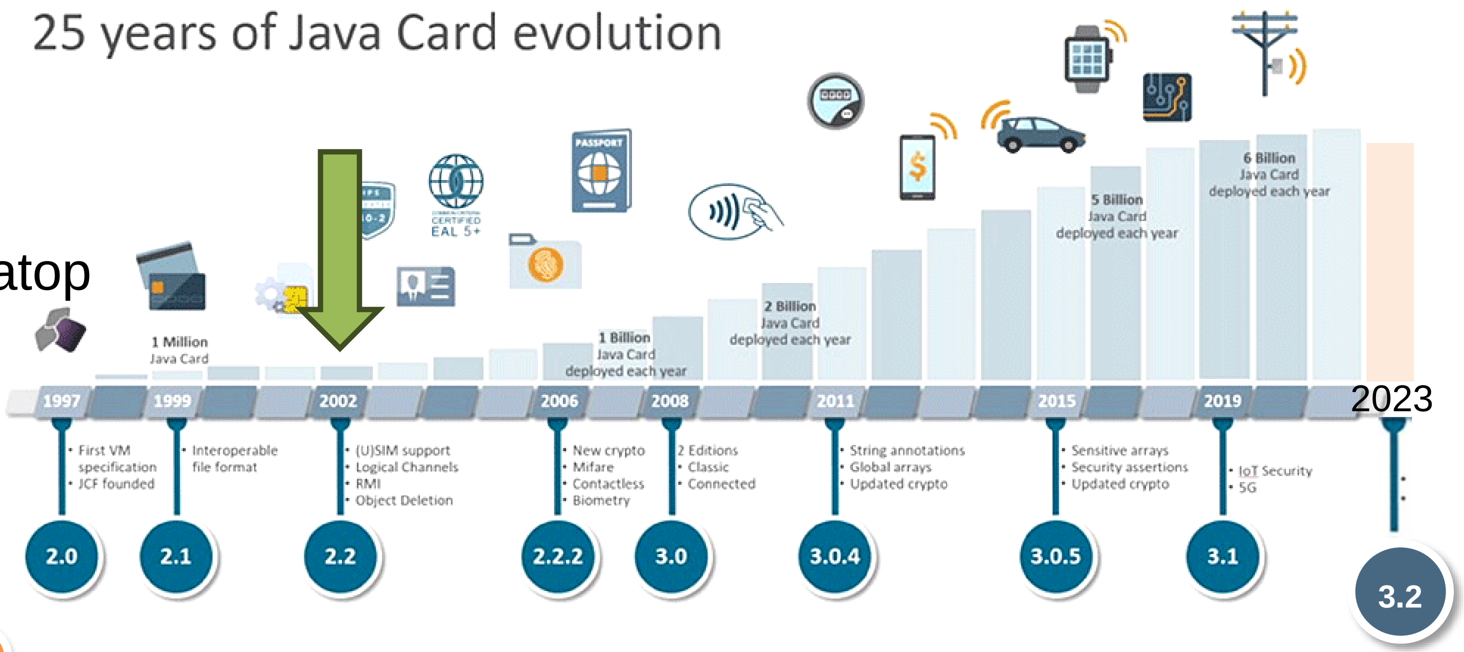
Centre for Research on Cryptography and Security, Masaryk University

CROCS
Centre for Research on
Cryptography and Security

JavaCard API specification (since 1997)

- Specification for on-card platform
  - Cryptographic algorithms
  - Data handling and utility methods
  - JCVM…
- Mandatory and optional parts
  - Implemented by platform provider atop of IC with hardware accelerators
- Allows for applet portability
  - JC bytecode + JCVM
  - (but proprietary packages)

RQ: How much is specification used in practice?



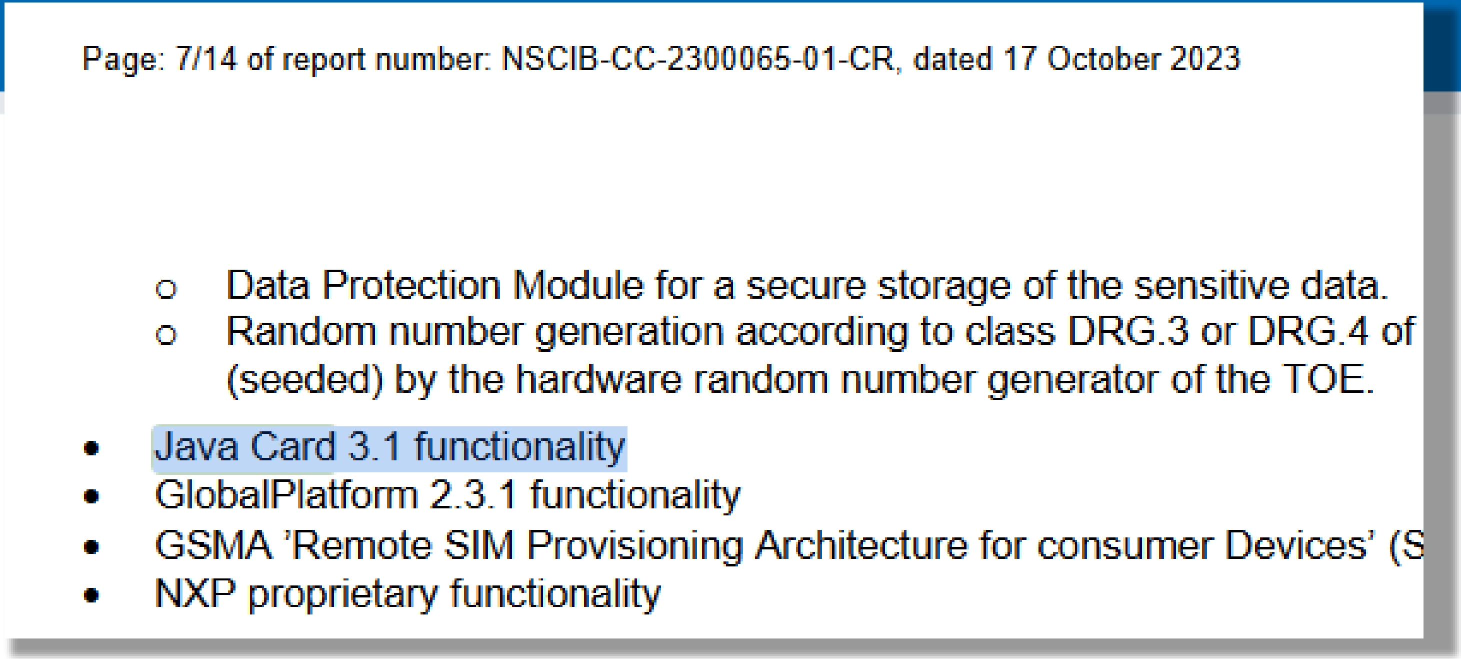
TLS1.3, EdDSA, RSA config…

https://www.oracle.com/news/announcement/blog/oracle-celebrates-the-java-card-forums-25th-anniversary-2022-10-20/

How to analyze usage of JavaCard technology?

1. Collect representative sample of users / projects (ideally “all”)
   - ~~All proprietary applets (where most JavaCard developers work)~~
   - All open-source JavaCard projects on GitHub / SourceForge (206 projects) (https://github.com/crocs-muni/javacard-curated-list)
2. Establish significance of projects
   - Number of users/developers/forks/stars, search trends on Google, sales stats…
   - Products certified under Common Criteria and FIPS140
3. Analyze projects for the level and style of use of technology
   - Static code analysis of JavaCard keywords and constants
   - Dynamic analysis on actual smartcard hardware
   - Trends in time if possible (e.g., code state in time via git commits)

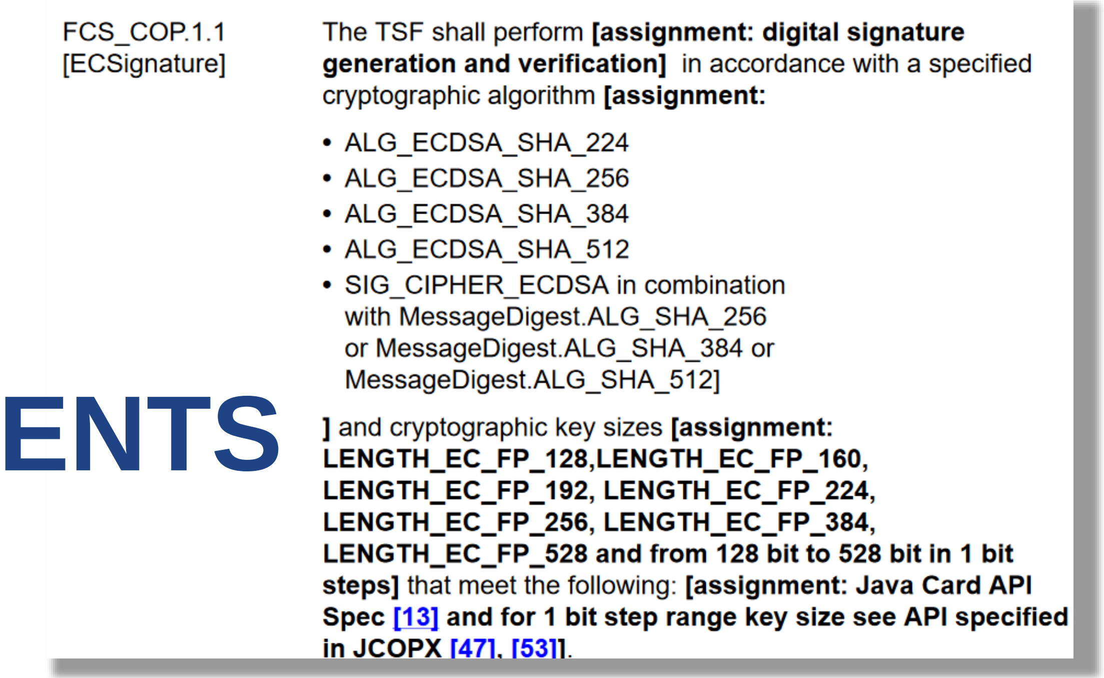
STATISTICS FROM CERTIFICATION DOCUMENTS



Page: 7/14 of report number: NSCIB-CC-2300065-01-CR, dated 17 October 2023

- Data Protection Module for a secure storage of the sensitive data.
- Random number generation according to class DRG.3 or DRG.4 of (seeded) by the hardware random number generator of the TOE.

- Java Card 3.1 functionality
- GlobalPlatform 2.3.1 functionality
- GSMA 'Remote SIM Provisioning Architecture for consumer Devices' (S
- NXP proprietary functionality



FCS_COP.1.1 [ECSignature]

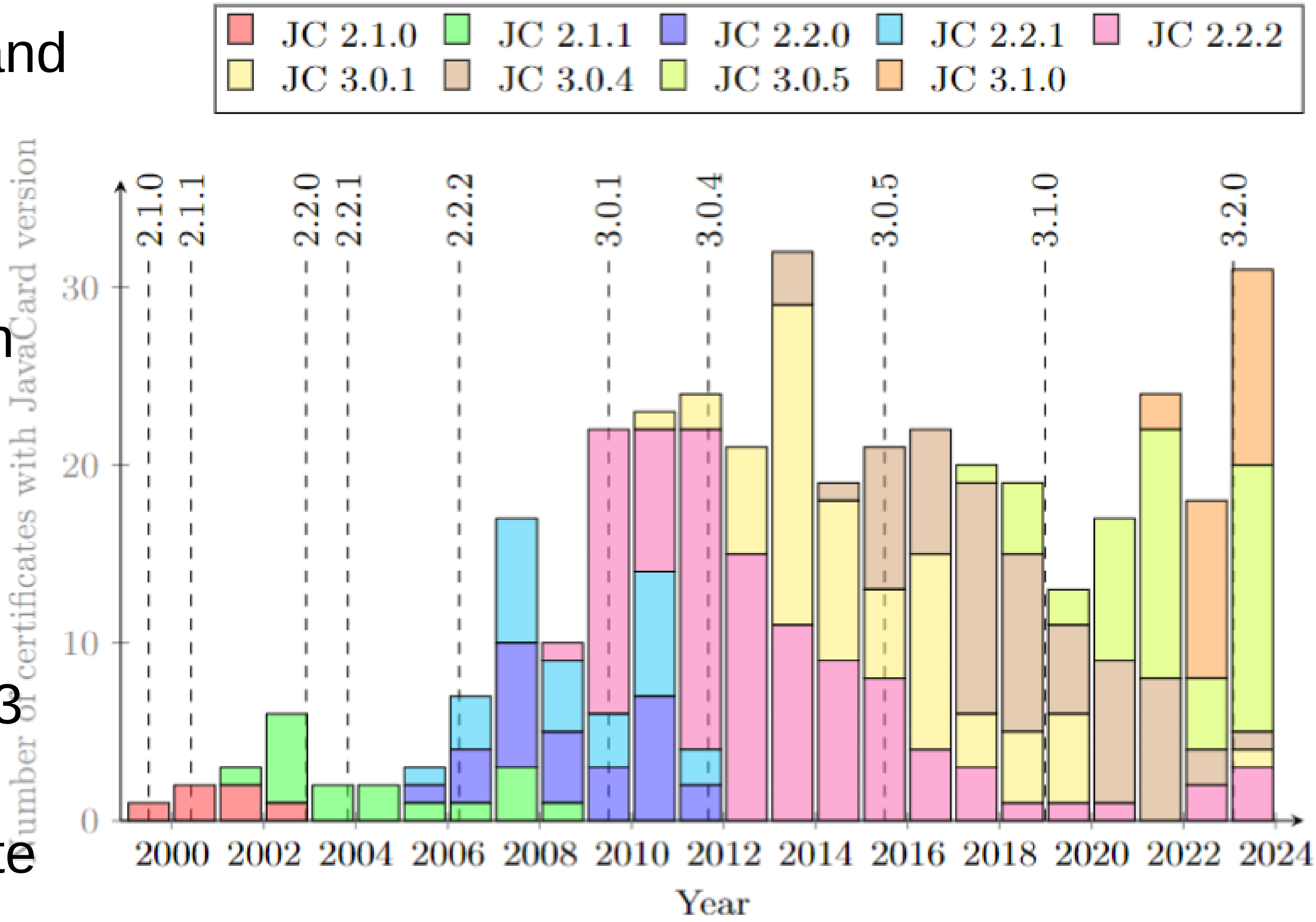
The TSF shall perform **[assignment: digital signature generation and verification]** in accordance with a specified cryptographic algorithm **[assignment:**

- ALG_ECDSA_SHA_224
- ALG_ECDSA_SHA_256
- ALG_ECDSA_SHA_384
- ALG_ECDSA_SHA_512
- SIG_CIPHER_ECDSA in combination with MessageDigest.ALG_SHA_256 or MessageDigest.ALG_SHA_384 or MessageDigest.ALG_SHA_512]

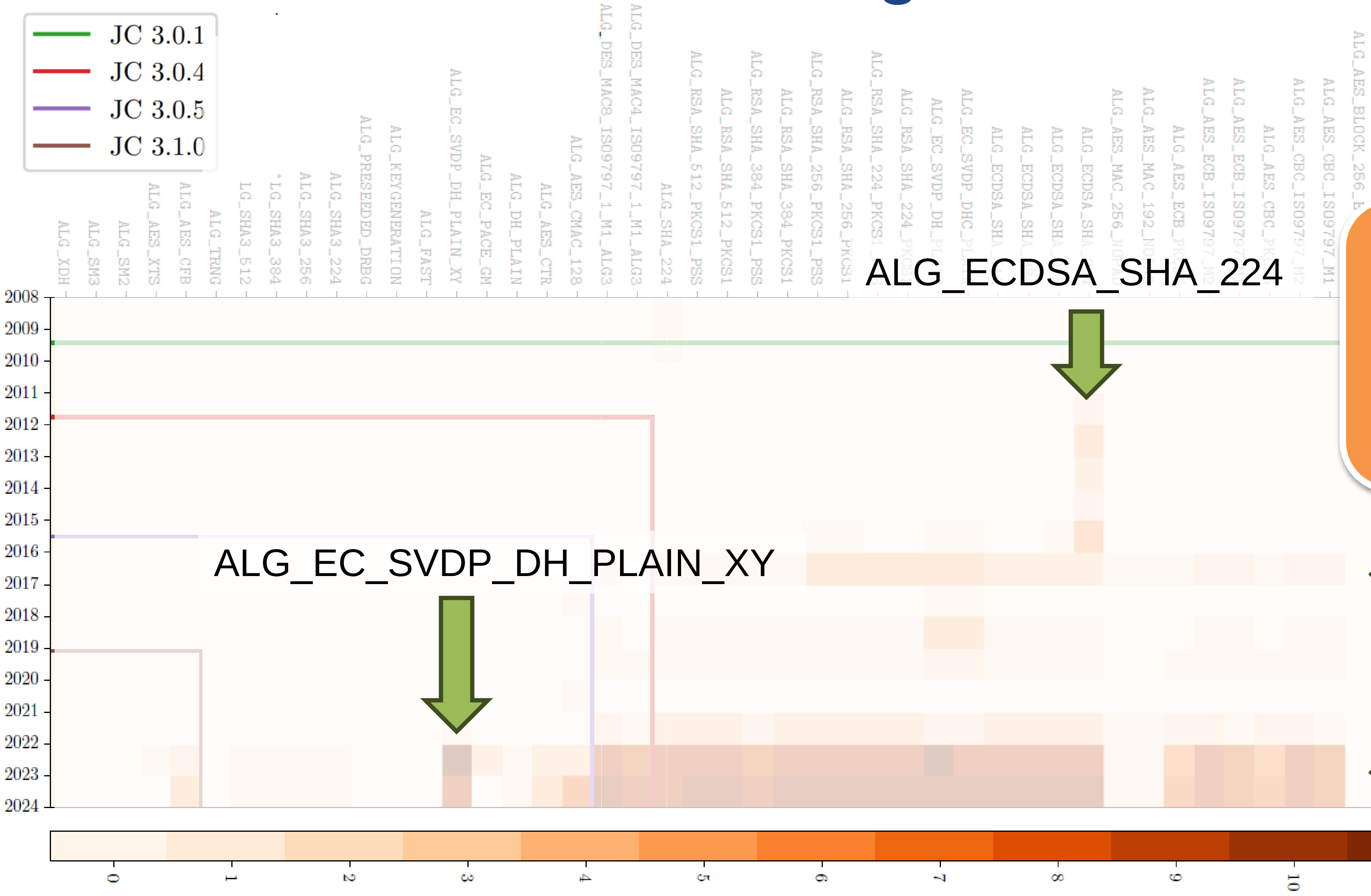
] and cryptographic key sizes **[assignment: LENGTH_EC_FP_128,LENGTH_EC_FP_160, LENGTH_EC_FP_192, LENGTH_EC_FP_224, LENGTH_EC_FP_256, LENGTH_EC_FP_384, LENGTH_EC_FP_528 and from 128 bit to 528 bit in 1 bit steps]** that meet the following: **[assignment: Java Card API Spec [13] and for 1 bit step range key size see API specified in JCOPX [47], [53]]**.

JavaCard API versions in certification docs

- Data from Common Criteria and FIPS140 (seccerts.org)
  - 2000-2009 (cards only FIPS140)
  - 2010-2023 (cards only CC)
- Regular expressions to match JavaCard API version
  - Only the latest API included
- Year 2023 only till 8.11.2023
  - But likely the best year ever ☺
- No 3.2.x yet, 2.2.2 still in 2023
- First certs 1.5-2 years after a JCAPI version publication date

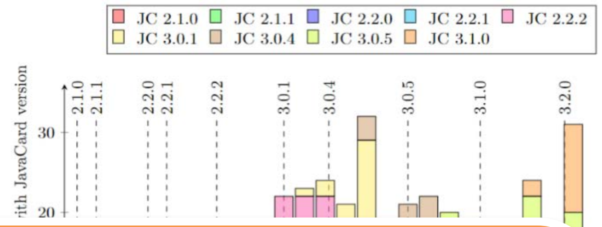


References for JC API algorithms in CC/FIPS140 docs



ALG_ECDSA_SHA_224

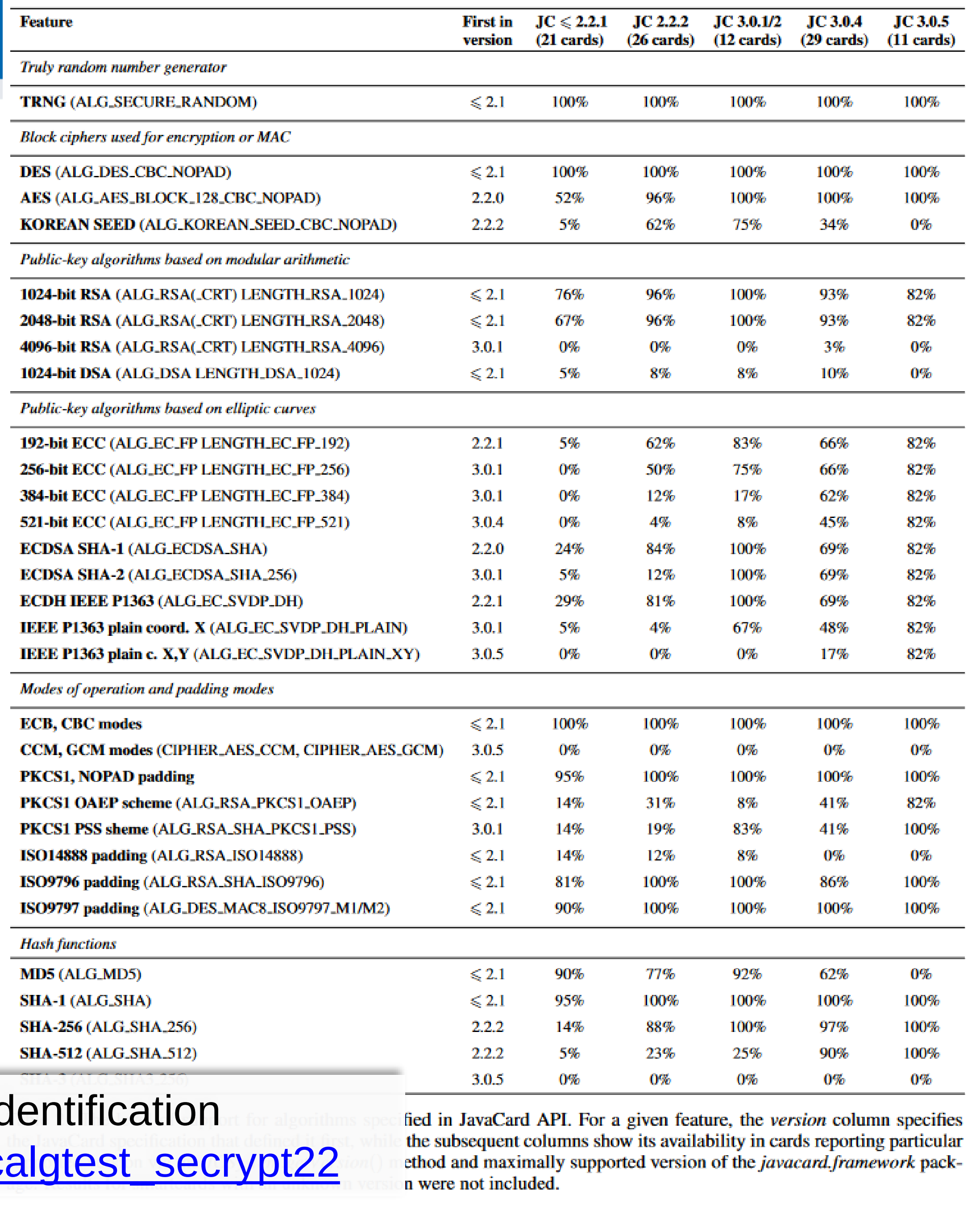
ALG_EC_SVDP_DH_PLAIN_XY



What if the algorithm is implemented but not mentioned in certification documents?

What real cards support?

- JCAlgTest project (since 2007) https://github.com/crocs-muni/JCAlgTest/
  - Support tested directly on real cards (100+)
  - Community-provided results in open db
- Study limitation
  - Only openly “available” cards tested
  - E.g., db’s top performer is JCOP4 on P71 yet JCOP8 on SN300 was recently certified



Feature	First in version	JC ⩽ 2.2.1 (21 cards)	JC 2.2.2 (26 cards)	JC 3.0.1/2 (12 cards)	JC 3.0.4 (29 cards)	JC 3.0.5 (11 cards)
Truly random number generator						
TRNG (ALG_SECURE_RANDOM)	⩽ 2.1	100%	100%	100%	100%	100%
Block ciphers used for encryption or MAC						
DES (ALG_DES_CBC_NOPAD)	⩽ 2.1	100%	100%	100%	100%	100%
AES (ALG_AES_BLOCK_128_CBC_NOPAD)	2.2.0	52%	96%	100%	100%	100%
KOREAN SEED (ALG_KOREAN_SEED_CBC_NOPAD)	2.2.2	5%	62%	75%	34%	0%
Public-key algorithms based on modular arithmetic						
1024-bit RSA (ALG_RSA(_CRT) LENGTH_RSA_1024)	⩽ 2.1	76%	96%	100%	93%	82%
2048-bit RSA (ALG_RSA(_CRT) LENGTH_RSA_2048)	⩽ 2.1	67%	96%	100%	93%	82%
4096-bit RSA (ALG_RSA(_CRT) LENGTH_RSA_4096)	3.0.1	0%	0%	0%	3%	0%
1024-bit DSA (ALG_DSA LENGTH_DSA_1024)	⩽ 2.1	5%	8%	8%	10%	0%
Public-key algorithms based on elliptic curves						
192-bit ECC (ALG_EC_FP LENGTH_EC_FP_192)	2.2.1	5%	62%	83%	66%	82%
256-bit ECC (ALG_EC_FP LENGTH_EC_FP_256)	3.0.1	0%	50%	75%	66%	82%
384-bit ECC (ALG_EC_FP LENGTH_EC_FP_384)	3.0.1	0%	12%	17%	62%	82%
521-bit ECC (ALG_EC_FP LENGTH_EC_FP_521)	3.0.4	0%	4%	8%	45%	82%
ECDSA SHA-1 (ALG_ECDSA_SHA)	2.2.0	24%	84%	100%	69%	82%
ECDSA SHA-2 (ALG_ECDSA_SHA_256)	3.0.1	5%	12%	100%	69%	82%
ECDH IEEE P1363 (ALG_EC_SVDP_DH)	2.2.1	29%	81%	100%	69%	82%
IEEE P1363 plain coord. X (ALG_EC_SVDP_DH_PLAIN)	3.0.1	5%	4%	67%	48%	82%
IEEE P1363 plain c. X,Y (ALG_EC_SVDP_DH_PLAIN_XY)	3.0.5	0%	0%	0%	17%	82%
Modes of operation and padding modes						
ECB, CBC modes	⩽ 2.1	100%	100%	100%	100%	100%
CCM, GCM modes (CIPHER_AES_CCM, CIPHER_AES_GCM)	3.0.5	0%	0%	0%	0%	0%
PKCS1, NOPAD padding	⩽ 2.1	95%	100%	100%	100%	100%
PKCS1 OAEP scheme (ALG_RSA_PKCS1_OAEP)	⩽ 2.1	14%	31%	8%	41%	82%
PKCS1 PSS sheme (ALG_RSA_SHA_PKCS1_PSS)	3.0.1	14%	19%	83%	41%	100%
ISO14888 padding (ALG_RSA_ISO14888)	⩽ 2.1	14%	12%	8%	0%	0%
ISO9796 padding (ALG_RSA_SHA_ISO9796)	⩽ 2.1	81%	100%	100%	86%	100%
ISO9797 padding (ALG_DES_MAC8_ISO9797_M1/M2)	⩽ 2.1	90%	100%	100%	100%	100%
Hash functions						
MD5 (ALG_MD5)	⩽ 2.1	90%	77%	92%	62%	0%
SHA-1 (ALG_SHA)	⩽ 2.1	95%	100%	100%	100%	100%
SHA-256 (ALG_SHA_256)	2.2.2	14%	88%	100%	97%	100%
SHA-512 (ALG_SHA_512)	2.2.2	5%	23%	25%	90%	100%
SHA-3 (ALG_SHA3_256)	3.0.5	0%	0%	0%	0%	0%

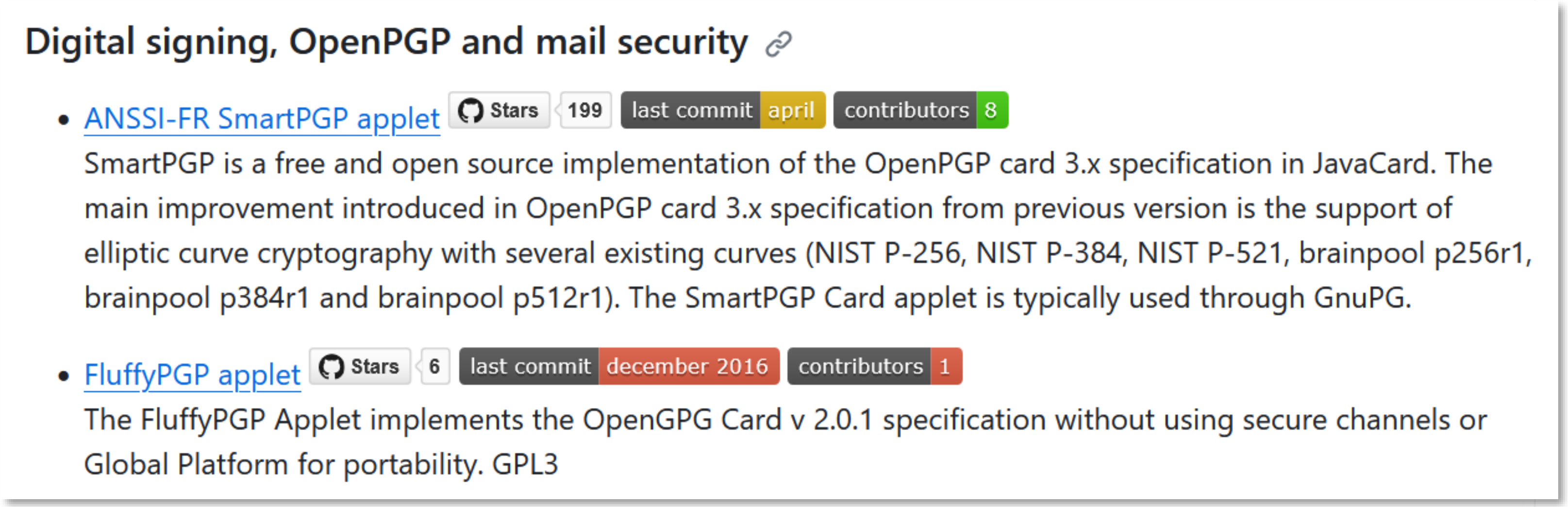
...rt for algorithms specified in JavaCard API. For a given feature, the *version* column specifies the JavaCard specification that defined it first, while the subsequent columns show its availability in cards reporting particular ...on v... ...ersion() method and maximally supported version of the *javacard.framework* package. Results for smartcards with an unknown version were not included.

P. Svenda, R. Kvasnovsky, I. Nagy, A. Dufka: JCAlgTest: Robust identification metadata for certified smartcards https://crocs.fi.muni.cz/papers/jcalgtest_secrypt22

ANALYZE ACTUAL USAGE IN CODE

Curated list of open-source applets (200+, from 2017)

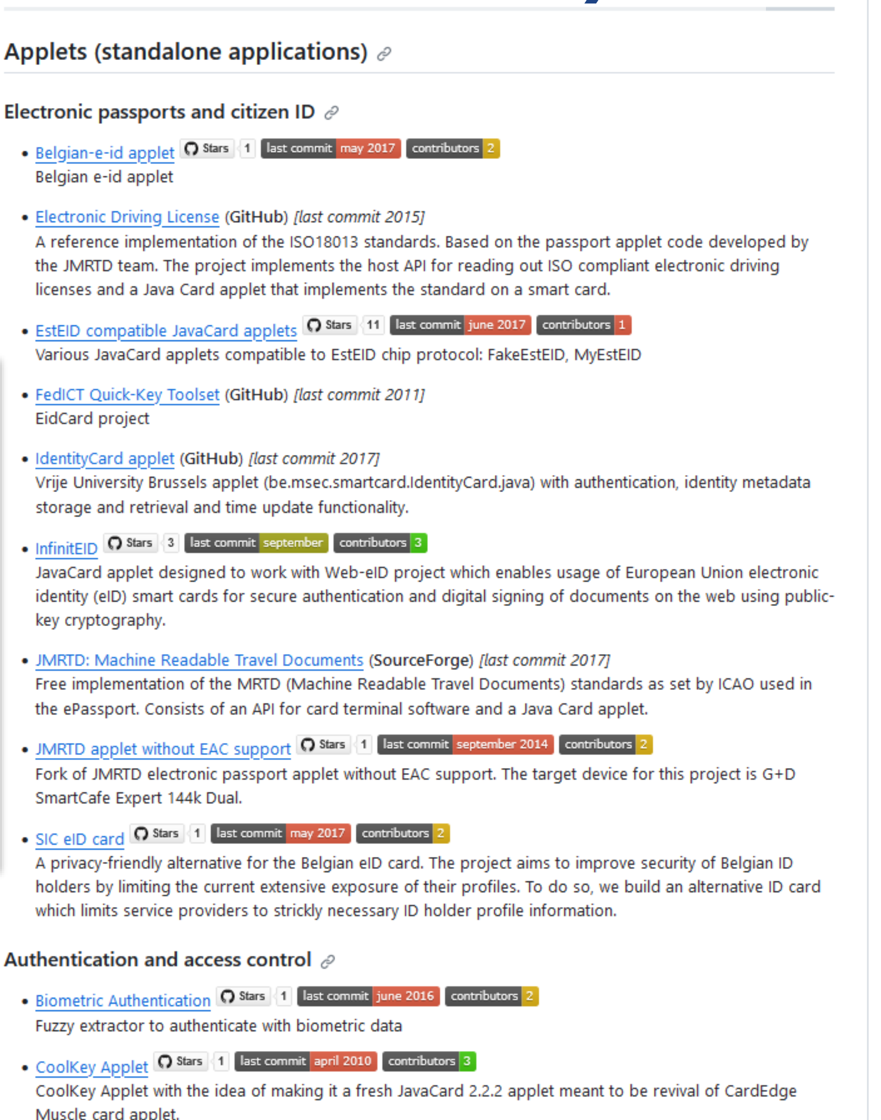
- https://github.com/crocs-muni/javacard-curated-list
- Standalone applets, library code, simulators/emulators, dev tools…



Digital signing, OpenPGP and mail security

- ANSSI-FR SmartPGP applet Stars 199 last commit april contributors 8
  SmartPGP is a free and open source implementation of the OpenPGP card 3.x specification in JavaCard. The main improvement introduced in OpenPGP card 3.x specification from previous version is the support of elliptic curve cryptography with several existing curves (NIST P-256, NIST P-384, NIST P-521, brainpool p256r1, brainpool p384r1 and brainpool p512r1). The SmartPGP Card applet is typically used through GnuPG.
- FluffyPGP applet Stars 6 last commit december 2016 contributors 1
  The FluffyPGP Applet implements the OpenGPG Card v 2.0.1 specification without using secure channels or Global Platform for portability. GPL3

- Contribute please! 🙏



Applets (standalone applications)

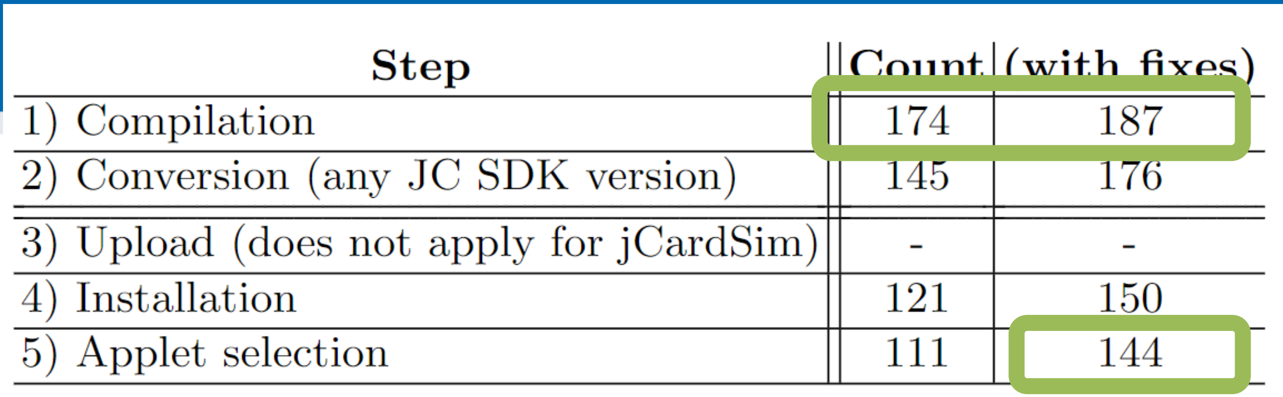
Electronic passports and citizen ID

- Belgian-e-id applet Stars 1 last commit may 2017 contributors 2
  Belgian e-id applet
- Electronic Driving License (GitHub) *[last commit 2015]*
  A reference implementation of the ISO18013 standards. Based on the passport applet code developed by the JMRTD team. The project implements the host API for reading out ISO compliant electronic driving licenses and a Java Card applet that implements the standard on a smart card.
- EstEID compatible JavaCard applets Stars 11 last commit june 2017 contributors 1
  Various JavaCard applets compatible to EstEID chip protocol: FakeEstEID, MyEstEID
- FedICT Quick-Key Toolset (GitHub) *[last commit 2011]*
  EidCard project
- IdentityCard applet (GitHub) *[last commit 2017]*
  Vrije University Brussels applet (be.msec.smartcard.IdentityCard.java) with authentication, identity metadata storage and retrieval and time update functionality.
- InfinitEID Stars 3 last commit september contributors 3
  JavaCard applet designed to work with Web-eID project which enables usage of European Union electronic identity (eID) smart cards for secure authentication and digital signing of documents on the web using public-key cryptography.
- JMRTD: Machine Readable Travel Documents (SourceForge) *[last commit 2017]*
  Free implementation of the MRTD (Machine Readable Travel Documents) standards as set by ICAO used in the ePassport. Consists of an API for card terminal software and a Java Card applet.
- JMRTD applet without EAC support Stars 1 last commit september 2014 contributors 2
  Fork of JMRTD electronic passport applet without EAC support. The target device for this project is G+D SmartCafe Expert 144k Dual.
- SIC eID card Stars 1 last commit may 2017 contributors 2
  A privacy-friendly alternative for the Belgian eID card. The project aims to improve security of Belgian ID holders by limiting the current extensive exposure of their profiles. To do so, we build an alternative ID card which limits service providers to strickly necessary ID holder profile information.

Authentication and access control

- Biometric Authentication Stars 1 last commit june 2016 contributors 2
  Fuzzy extractor to authenticate with biometric data
- CoolKey Applet Stars 1 last commit april 2010 contributors 3
  CoolKey Applet with the idea of making it a fresh JavaCard 2.2.2 applet meant to be revival of CardEdge Muscle card applet.

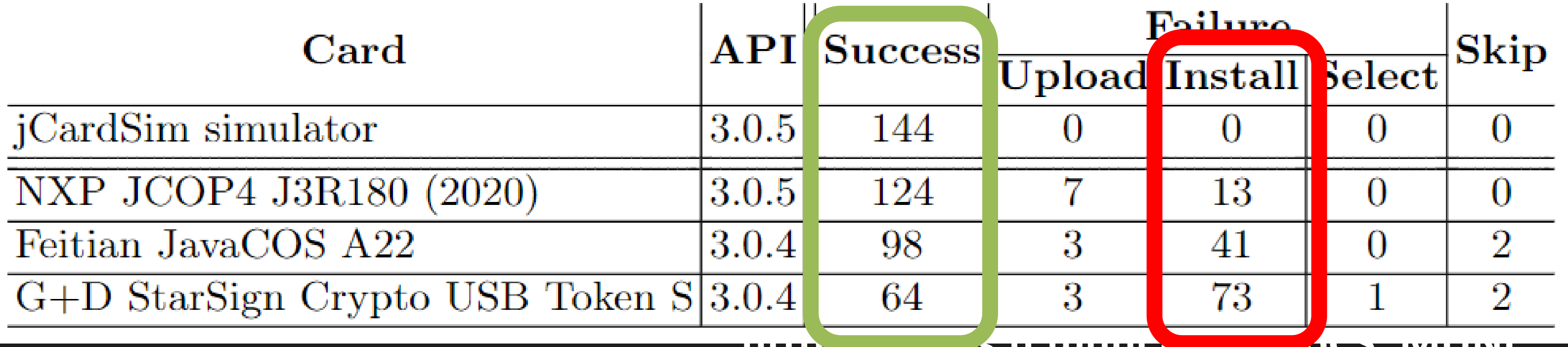
JCProfilerNext tool



Step	Count	(with fixes)
1) Compilation	174	187
2) Conversion (any JC SDK version)	145	176
3) Upload (does not apply for jCardSim)	-	-
4) Installation	121	150
5) Applet selection	111	144

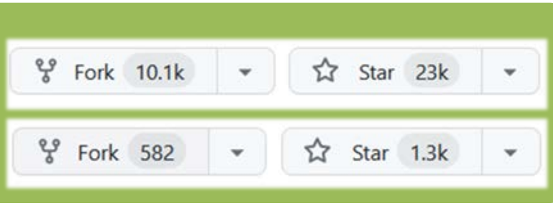
- https://github.com/lzaoral/JCProfilerNext
- Static and dynamic analyzer for JavaCard applets (Spoon-based)
  - Used packages, methods, constants...
  - Instrumentation of applet constructor to detect memory consumption
- Test on simulated card (jcardsim) – no limitations of actual hardware
- Test on real cards (NXP JCOP4, JavaCOS A22, G+D StarSign 7.0)
  - Memory consumption of allocations done in constructor
  - Portability of applets

https://spoon.gforge.inria.fr/

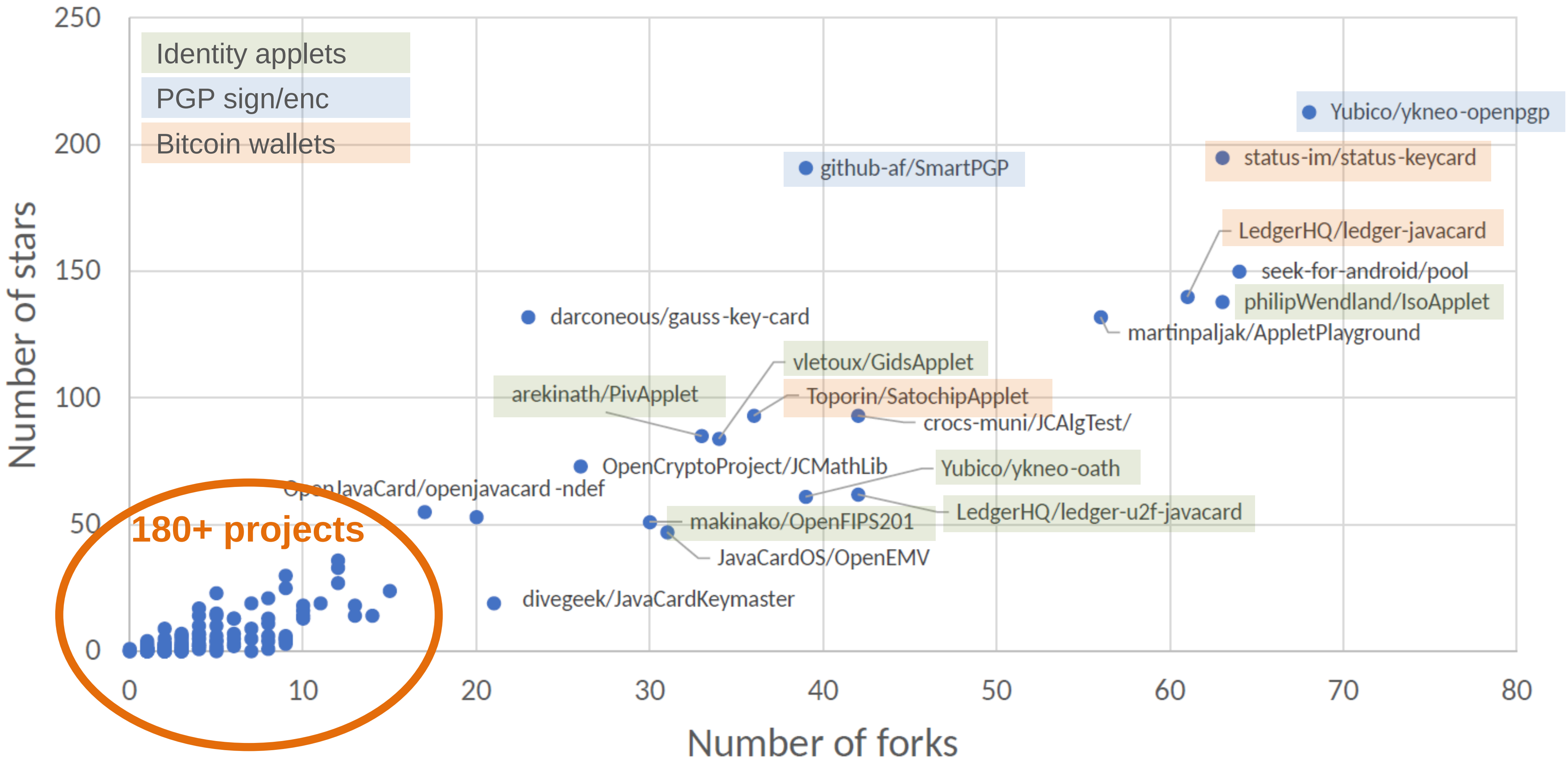


Card	API	Success	Failure			Skip
			Upload	Install	Select	
jCardSim simulator	3.0.5	144	0	0	0	0
NXP JCOP4 J3R180 (2020)	3.0.5	124	7	13	0	0
Feitian JavaCOS A22	3.0.4	98	3	41	0	2
G+D StarSign Crypto USB Token S	3.0.4	64	3	73	1	2

(just for comparison)
OpenSSL crypto lib
Credit Card Reader NFC



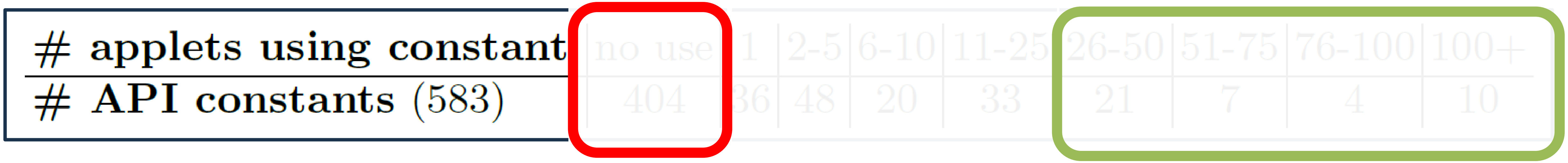
Popularity of open-source JavaCard projects



180+ projects

Frequency of use of JavaCard API constants in JCOSS

- JavaCard API 2.1.0 - 3.2 introduced 583 constants (in total)
- 404 constants (69.3%) are completely unused (in open-source applets)
- Only 42 constants (7.2%) are used in more than 25 projects

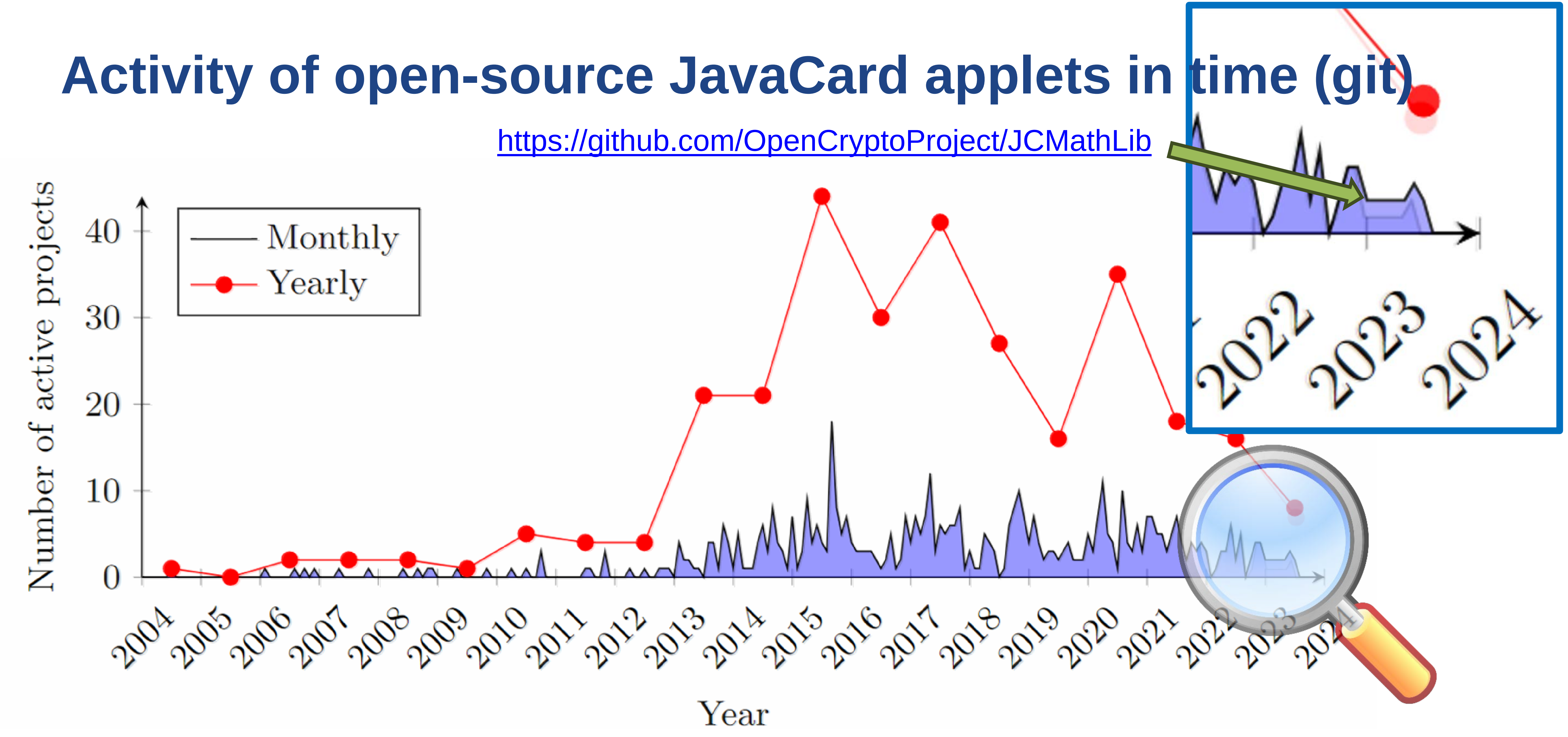


# applets using constant	no use	1	2-5	6-10	11-25	26-50	51-75	76-100	100+
# API constants (583)	404	36	48	20	33	21	7	4	10

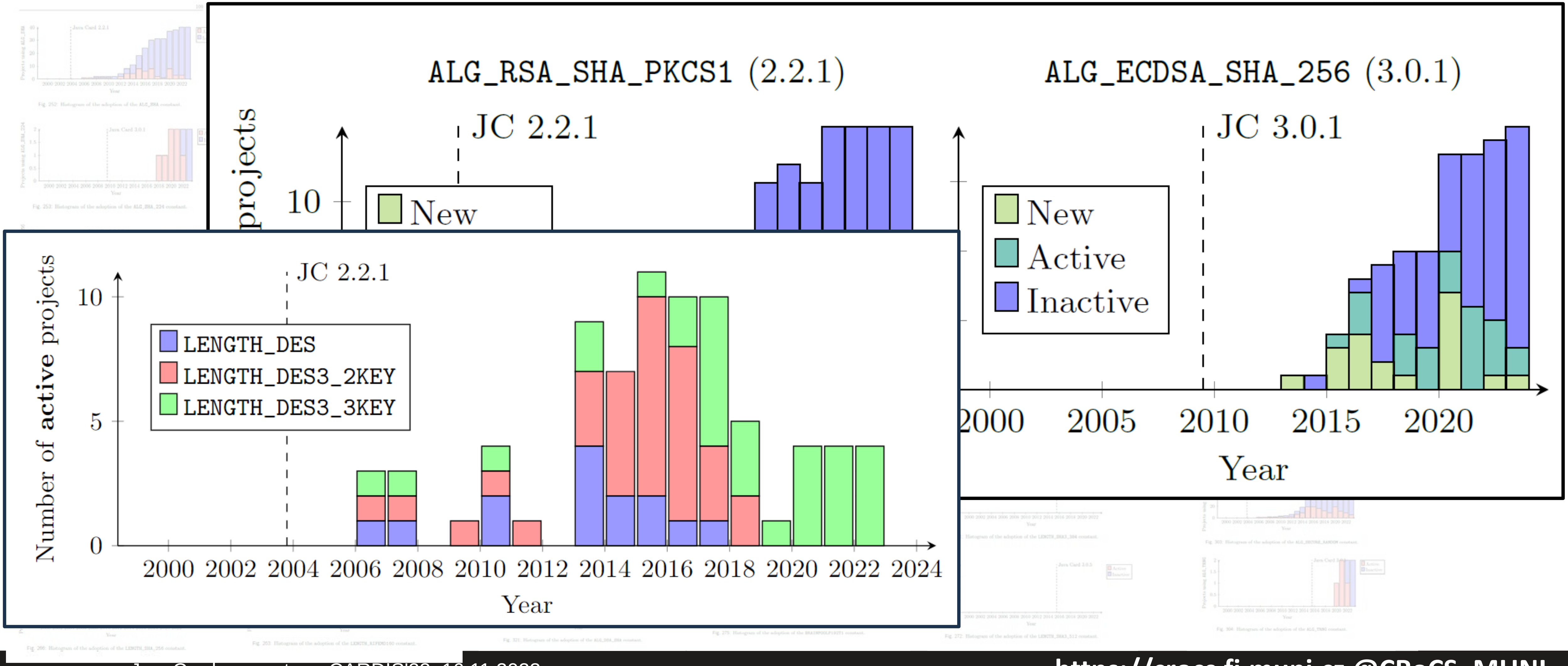
- “Popularity” of constants may also change in time
  – Analyze using `git checkout last_commit` in given month / year

Activity of open-source JavaCard applets in time (git)

https://github.com/OpenCryptoProject/JCMathLib

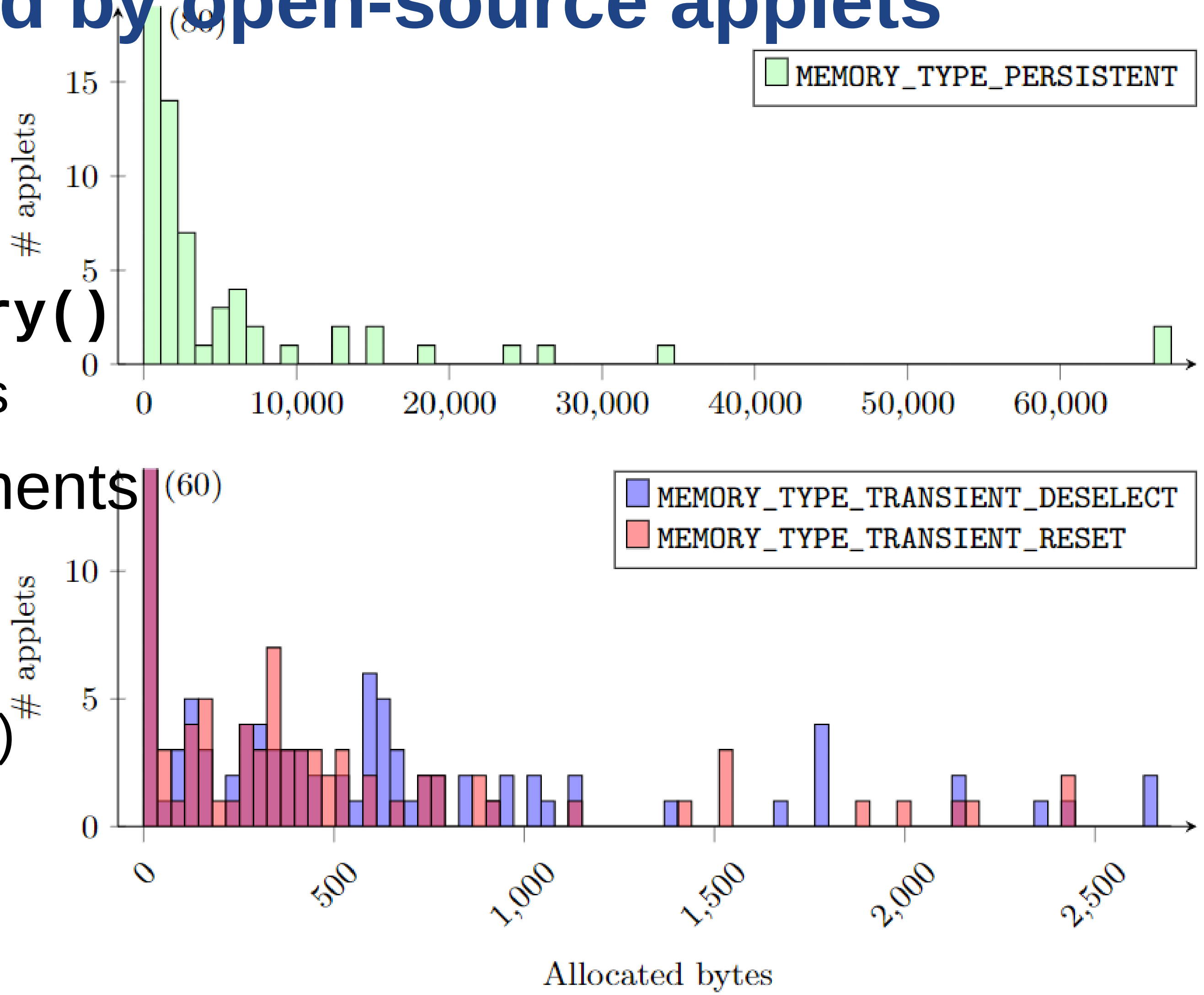


Algorithm support in time (all at paper artifacts page)



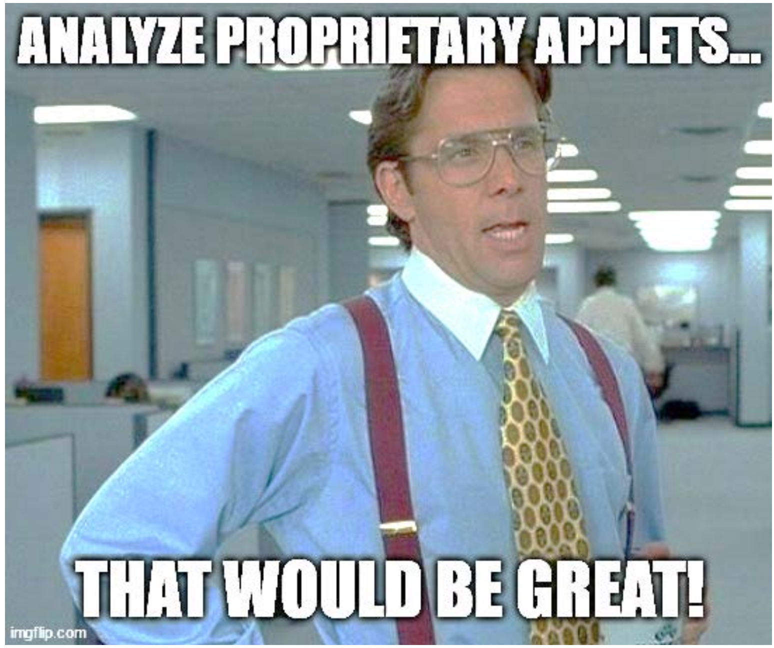
Memory resources required by open-source applets

- Tested on 3 real cards
  - Instrumented constructor
  - **`JCSystem.getAvailableMemory()`**
  - Only small variation between cards
- Most applets have low requirements
  - <10kB EEPROM
  - <1kB RAM
  - Some applets cannot fit (9kB RAM)



Limitations of our study

- Is open-source ecosystem representative of the whole domain?
  - – Likely two orders of magnitude more developers outside open-source domain
  - – Proprietary applets with access to proprietary API may be different
    - Earlier adoption of features, different algorithms…
    - Different memory footprint (smaller/larger)
- New cards not tested
  - – Analyzed certification documents provide some insights
- Applets tested only up to **`select()`** method
  - – Complete testing requires tests with high coverage
  - – But allocation in constructor is good practice



Conclusions for JavaCard (open-source) ecosystem

- More JavaCard-related items certified in 2023 than ever
- Large majority of JavaCard API constants unused by OSS (~69%)
- Open-source ecosystem peaked in 2015-2017, now declining
- Unavailability of new cards and “lower” level API likely negatively impact development of interesting ideas (ALG_EC_SVDP_DH_PLAIN_XY)
- Analysis of proprietary applets is important missing part (contact us!)
- All tools and results available https://crocs.fi.muni.cz/papers/cardis2023

Questions