# The Million-Key Question: Investigating the Origins of RSA Public Keys

**Based on paper "The Million-Key Question: Investigating the Origins of RSA Public Keys"**
**25th Usenix Security Symposium, 2016. Received Best Paper Award**

Petr Švenda, Matúš Nemec, Peter Sekan, Rudolf Kvašňovský, David Formánek, David Komárek and Vashek Matyáš
*svenda@fi.muni.cz* *@rngsec*
Faculty of Informatics, Masaryk University, Czech Republic

CROCS
Centre for Research on Cryptography and Security

Crypto library code

$$P \times Q = N$$

## RSA public key

**N =** 9782D7123C330444C88E279BF321EE84AC39524F1D84026327B04F32E1E930FC81588010178
DC75FCBF8258A068071317245D08817988813C4173495A922A41DA429A964F738020076EFFE7ED
5811088873C6E58EEF1CDC900596681[illegible]BE72368B51A821FC699E9C3FD66B377E2DF2485DC4
01DD99CC125890E5D969A6AC8B

**e =** 10001

OpenSSL

Infineon Crypto Java Ca

# Overview

- Motivation – information leakage in RSA public keys
- Learning phase: analysis of large number of RSA keypairs
- Classification phase: identify source library from public key
- Applications of classification capability
- Random numbers in cryptographic smartcards
- Smartcards and RSA keypair generation
- Summary and future work

Analysis of large number of RSA keys

# LEARNING PHASE

## 22 software libraries and versions

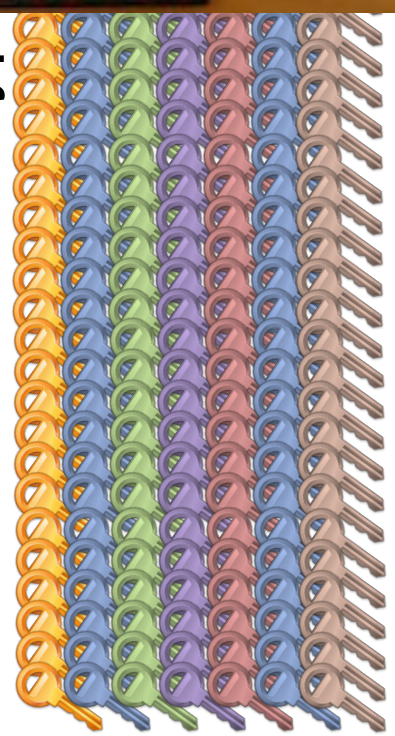
ARM mbed
FlexiProvider
Harnessing the power of the Java Cryptography Architecture
Nettle
GnuPG
cryptlib
OpenSSL
OpenJDK
STRONG CRYPTIX CRYPTO
GNU CRYPTO
Microsoft
wolfSSL
OPEN SOURCE, OPEN ACADEMIA, OPEN MINDS
LibTom PROJECTS
Botan
Crypto++
PGP
FIPS VALIDATED 140-2

## 16 types of smart

G&D Crypto Java Card
Infin Crypto
Gemalto Crypto Java Card
NXP Crypto Java Card
Oberthur Crypto Java Card
Feitian Crypto Java Card

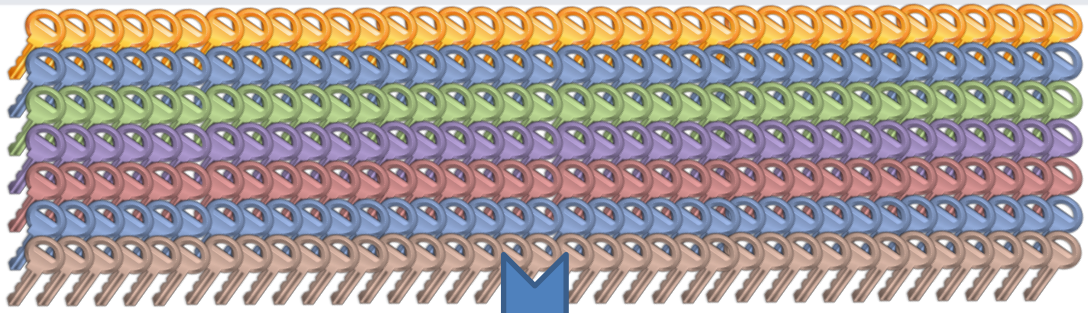
**1 000 000 x**
**`Gen_RSA_keypair()`**

60+ million RSA keyp

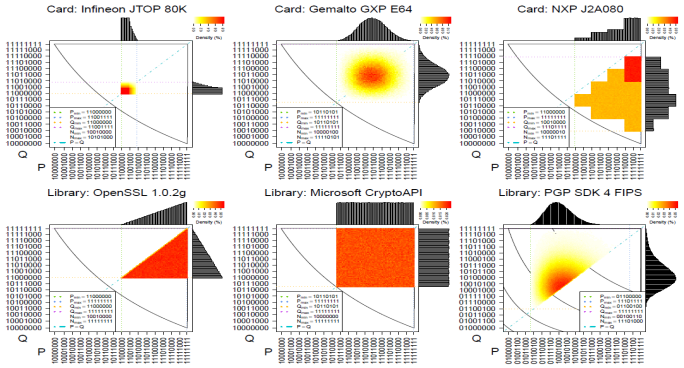


# 60+ million fresh RSA keypairs

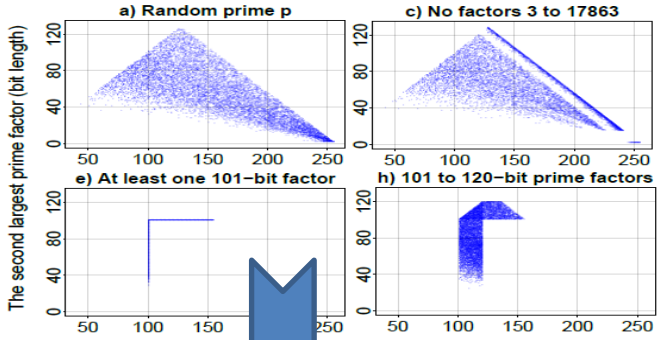
22 sw. libraries
16 smart cards



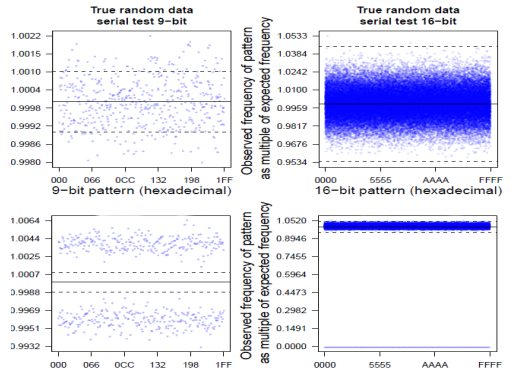
## Distribution of primes (MSB)



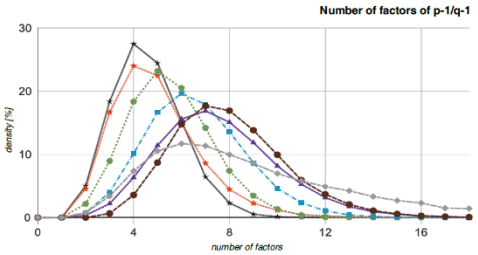

## Large factors of p-1 / p+1




## Bit stream statistics



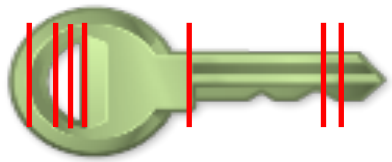

## Number of factors




and more…

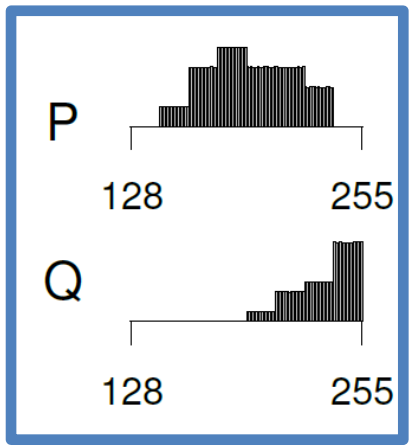
# 7 implementation choices observable in public keys

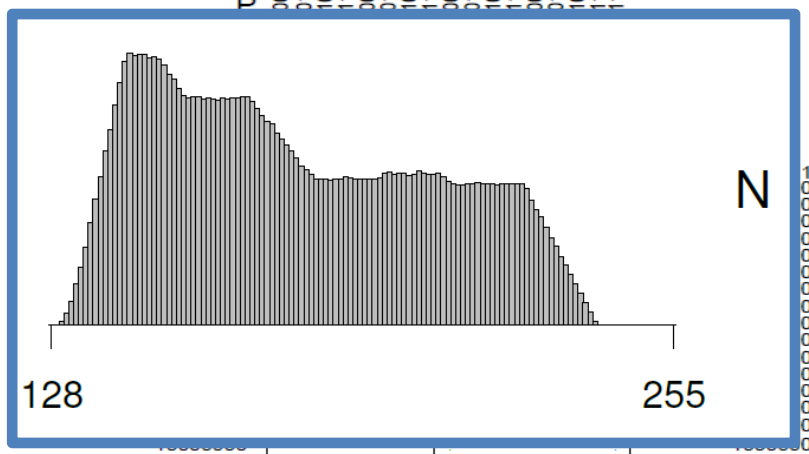
(biased bits of public modulus, “mask”)

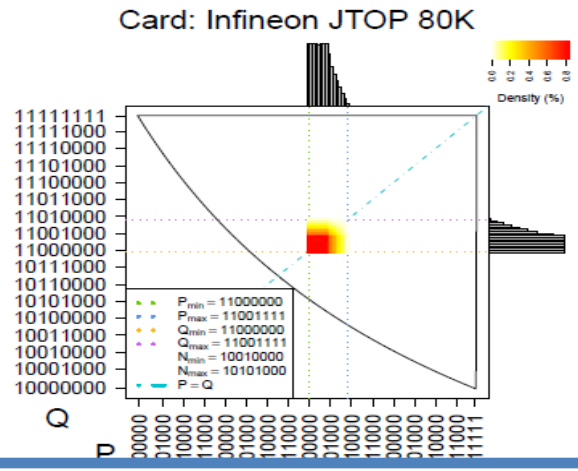


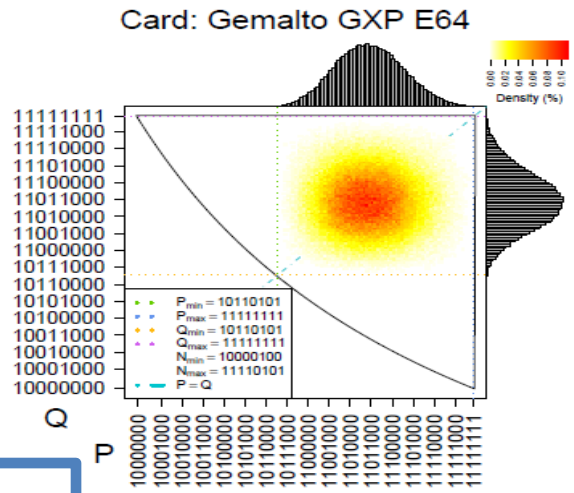
# Heatmap of primes' most significant byte

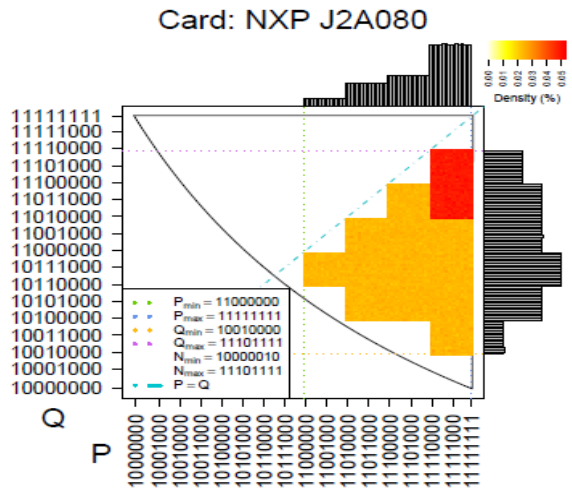
$$P \times Q = N$$

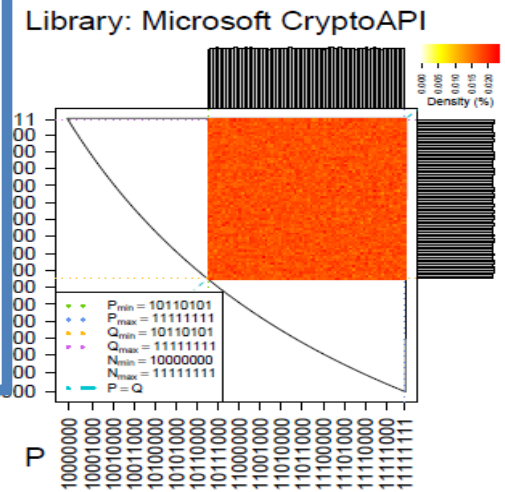


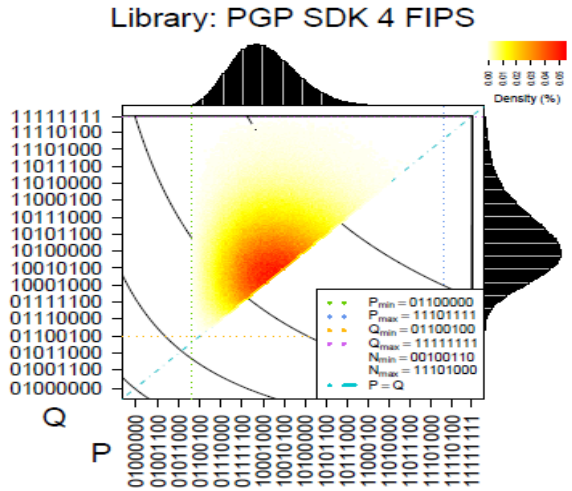



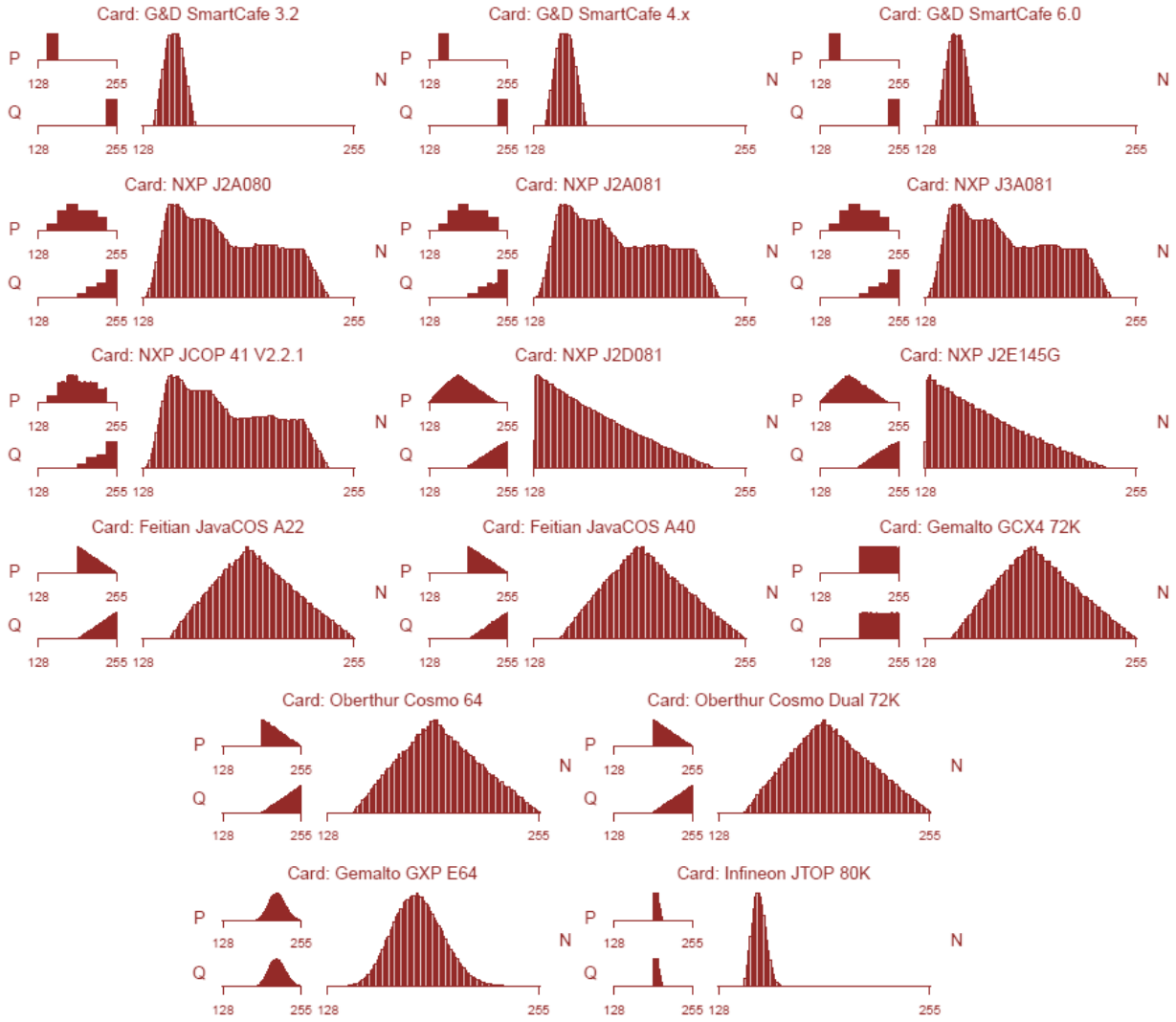



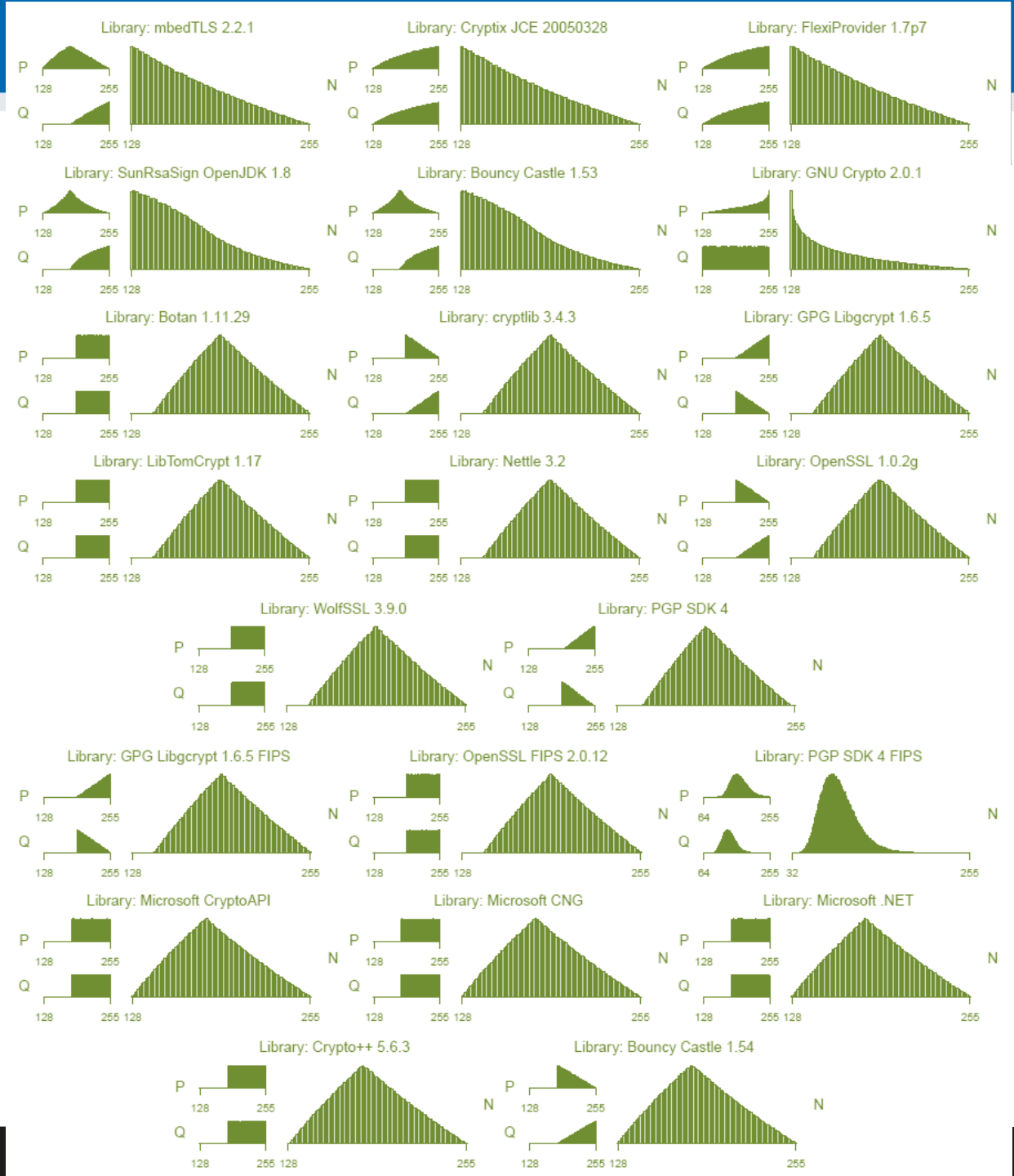



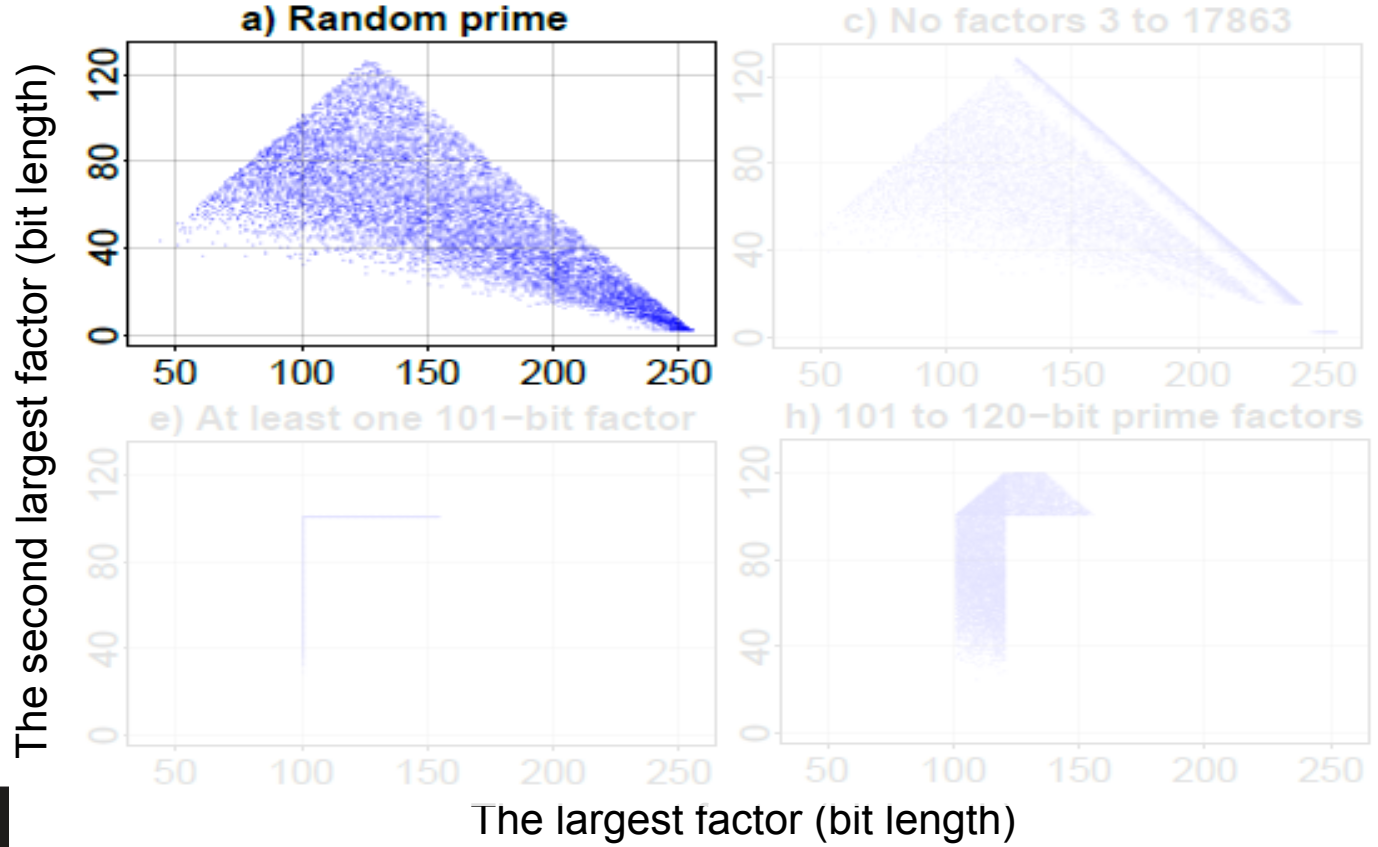
# MSB of modulus – libs/cards



# Factors of P−1/Q−1 (and its impact on modulus N)

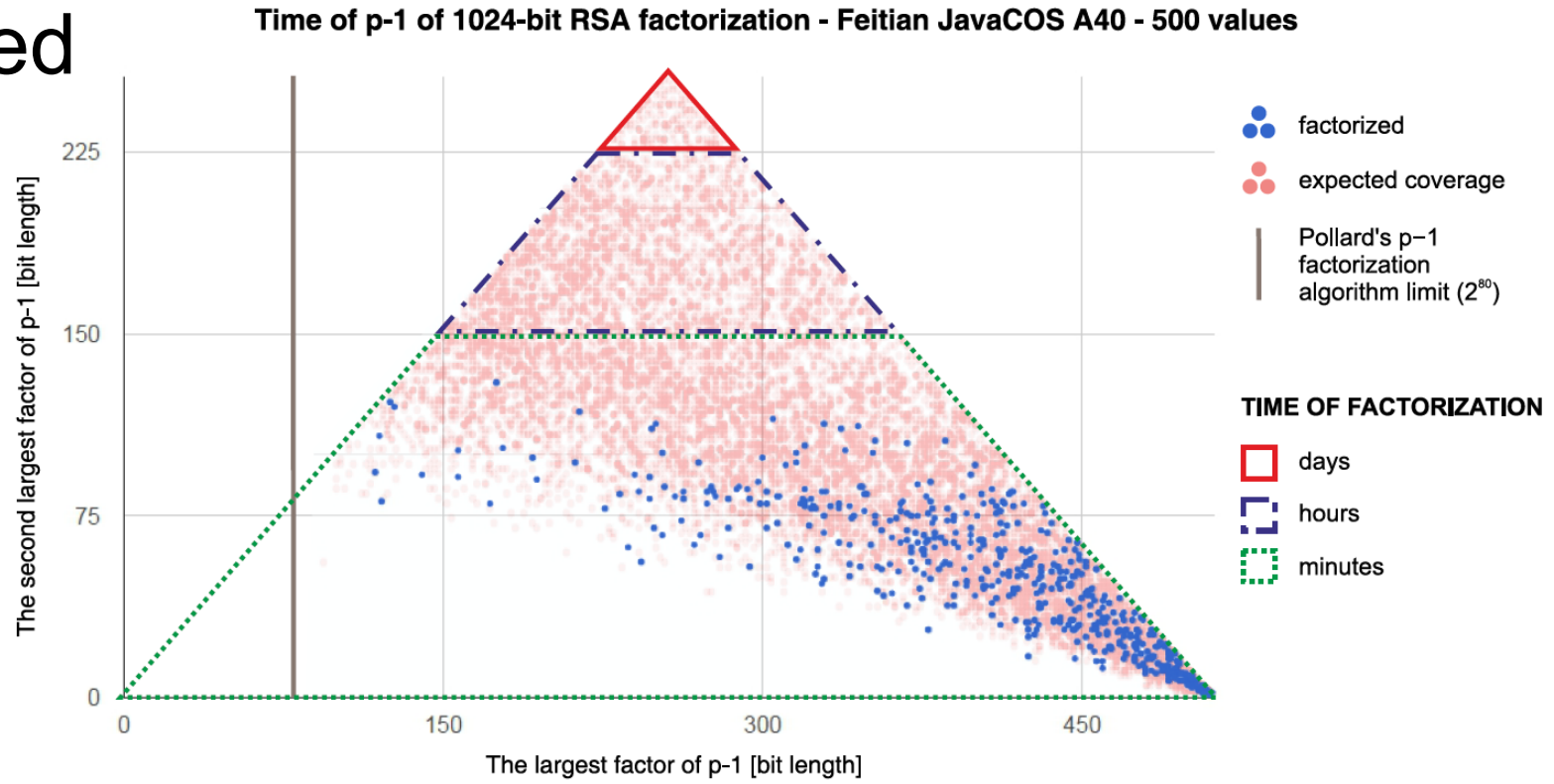
- For RSA512b, length of prime is 256bits => **P−1/Q−1** can be factorized
- We factorized 10k primes for every source with YAFU and…
- Small factors avoided
  - Significant bias on lower bits of N
  - Used by I. Mironov (OpenSSL)
- FIPS primes (specific range)
  - Not observable in modulus N




*MIRONOV, I. Factoring RSA Moduli II. https://windowsontheory.org/2012/05/17/factoring-rsa-moduli-part-ii/*

# Have 512b keys same properties as 1024/2048-bit keys?

- Can be checked in code for open-source libraries
- Extrapolation + check for black-box sources
- No difference detected

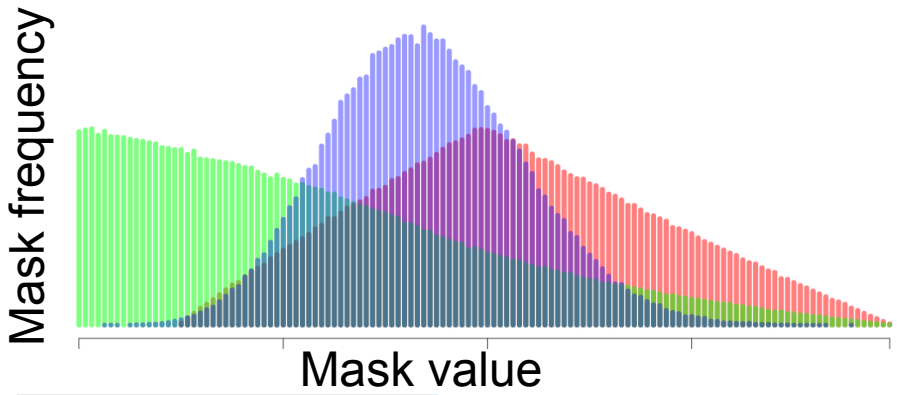


# Implementation choices observable in public key

Significance

1. Direct manipulation of the primes' highest bits
2. Avoidance of small factors in P-1 and Q-1
3. Requirement for moduli to be Blum integers
4. Restriction of the primes' bit length
5. Specific method to construct strong or provable primes
6. Use of another non-traditional algorithm – functionally unknown, but statistically observable
7. Type of action after candidate prime rejection

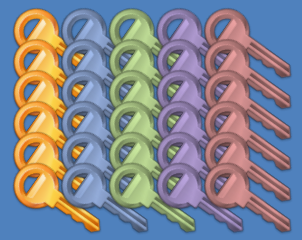
# Building classification matrix




15x 000000000
175x 000000001
...
83x 111111121

E.g.,
101101110

Harvest keys from known sources (learning set)



Apply mask to learning set

Count mask frequency

Group sources with very similar frequencies

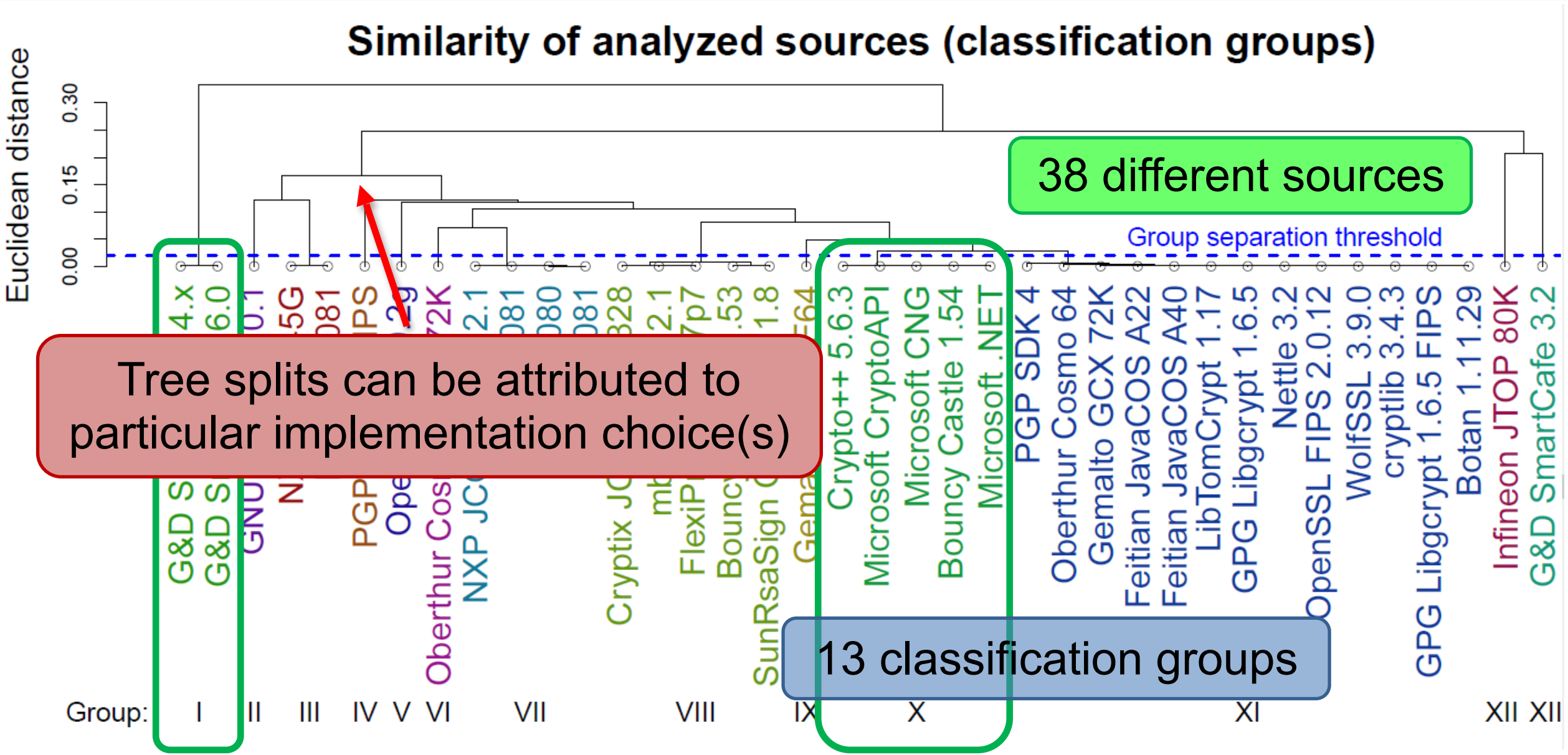
Normalize mask vectors of groups

Identification of biased modulus bits (mask, 9bits)

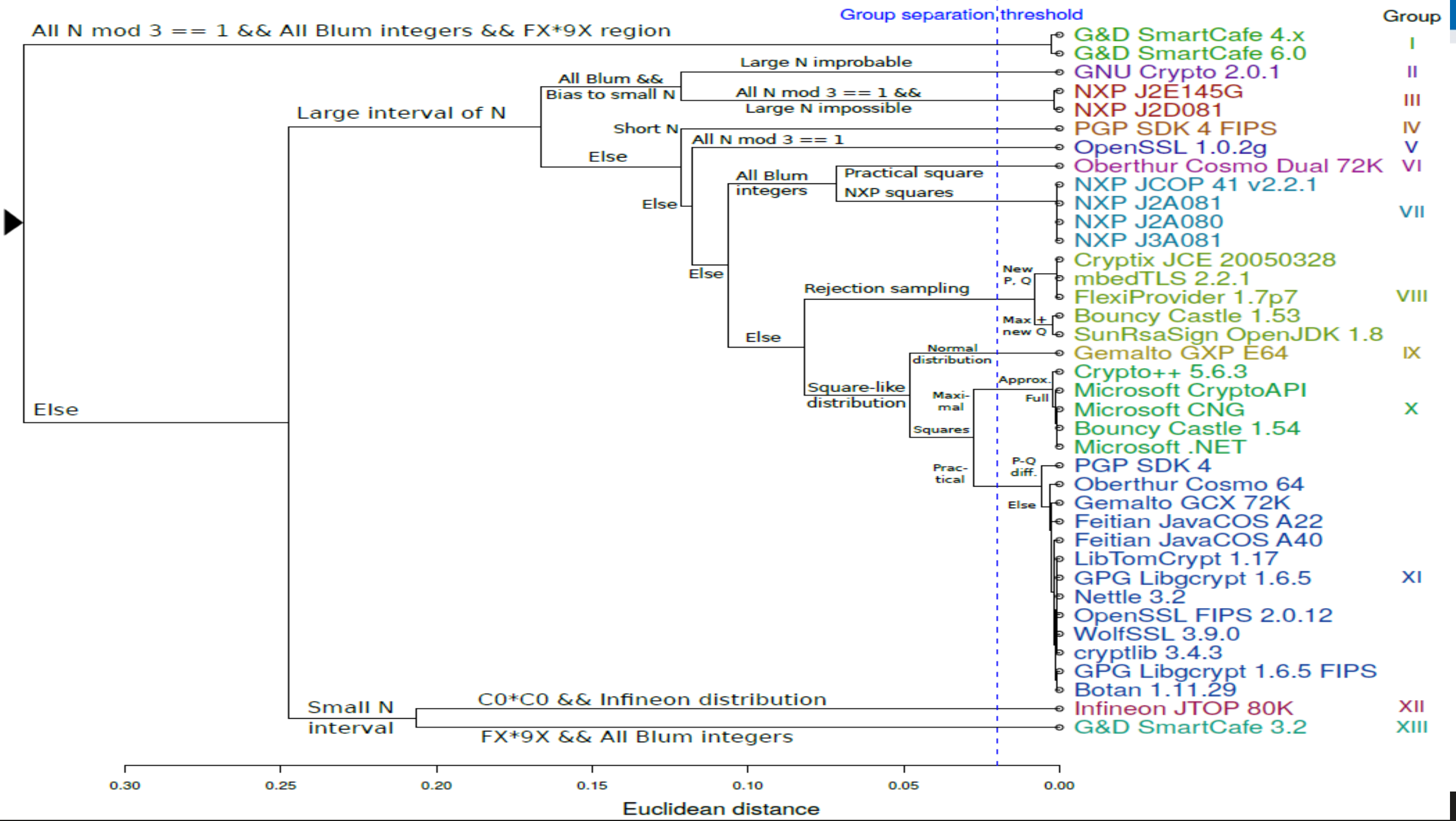
*2nd-7th MSb | 2nd LSb | modulus mod 3 | len(modulus) mod 2*

## Classification matrix

| Mask value | Group I | Group II | … | Group XII | Group XIII |
|---|---|---|---|---|---|
| 000000000 | 0.124 | 0.347 | | 0.105 | 0.012 |
| 000000001 | 0.004 | 0.038 | | 0.236 | 0.454 |
| 000000011 | 0.046 | 0.002 | | 0.447 | 0.112 |
| … | | | | | |
| 111111110 | 0.394 | 0.044 | | 0.320 | 0.002 |
| 111111111 | 0.046 | 0.347 | | 0.015 | 0.312 |


Similarity of analyzed sources (classification groups)
Euclidean distance
0.00
0.15
0.30
38 different sources
Group separation threshold
Tree splits can be attributed to particular implementation choice(s)
13 classification groups
Group:
I
II
III
IV
V
VI
VII
VIII
IX
X
XI
XII
XII
Crypto++ 5.6.3
Microsoft CryptoAPI
Microsoft CNG
Bouncy Castle 1.54
Microsoft .NET
PGP SDK 4
Oberthur Cosmo 64
Gemalto GCX 72K
Feitian JavaCOS A22
Feitian JavaCOS A40
LibTomCrypt 1.17
GPG Libgcrypt 1.6.5
Nettle 3.2
OpenSSL FIPS 2.0.12
WolfSSL 3.9.0
cryptlib 3.4.3
GPG Libgcrypt 1.6.5 FIPS
Botan 1.11.29
Infineon JTOP 80K
G&D SmartCafe 3.2

# Similarity of analyzed sources (classification groups) with annotated differences

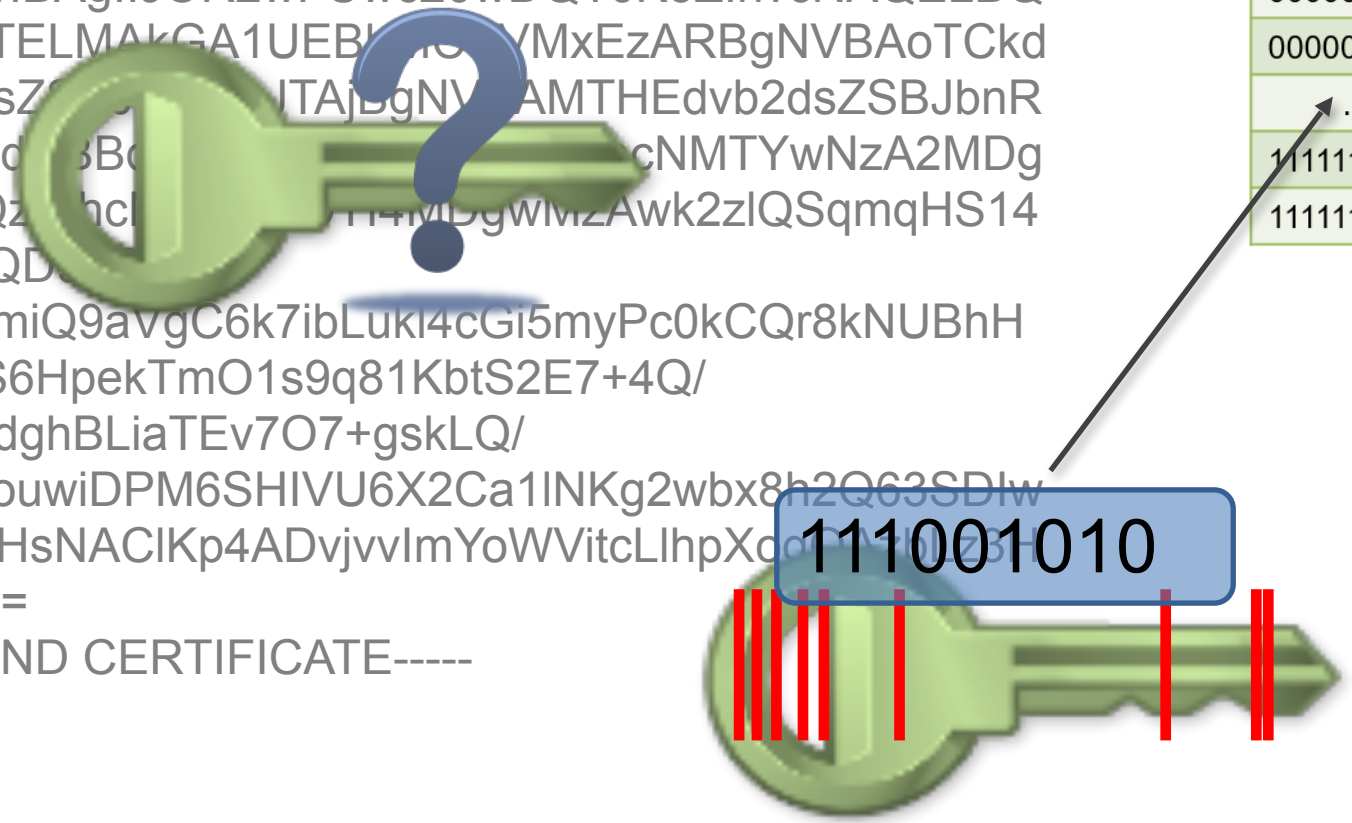


Identify origin library or device which generated given key

# CLASSIFICATION PHASE

# Input key

-----BEGIN CERTIFICATE-----
MIIG9zCCBd
+gAwIBAgIIJOR2wFUwc20wDQYJKoZIhvcNAQELBQ
AwSTELMAkGA1UEB
vb2dsZ
lcm5lc
xNzQz
NRoQD
rEp4miQ9avgC6k7ibLukl4cGi5myPc0kCQr8kNUBhH
25DS6HpekTmO1s9q81KbtS2E7+4Q/
57xgdghBLiaTEv7O7+gskLQ/
qJaTouwiDPM6SHIVU6X2Ca1lNKg2wbx8h2Q63SDlw
FJ52HsNAClKp4ADvjvvImYoWVitcLlhpXd
ls6Jk=
-----END CERTIFICATE-----




# Precomputed matrix

| Mask value | Group I | Group II | … | Group XII | Group XIII |
|---|---|---|---|---|---|
| 000000000 | 0.124 | 0.347 | | 0.105 | 0.012 |
| 000000001 | 0.004 | 0.038 | | 0.236 | 0.454 |
| 000000011 | 0.046 | 0.002 | | 0.447 | 0.112 |
| … | | | | | |
| 111111110 | 0.394 | 0.044 | | 0.320 | 0.002 |
| 111111111 | 0.046 | 0.347 | | 0.015 | 0.312 |

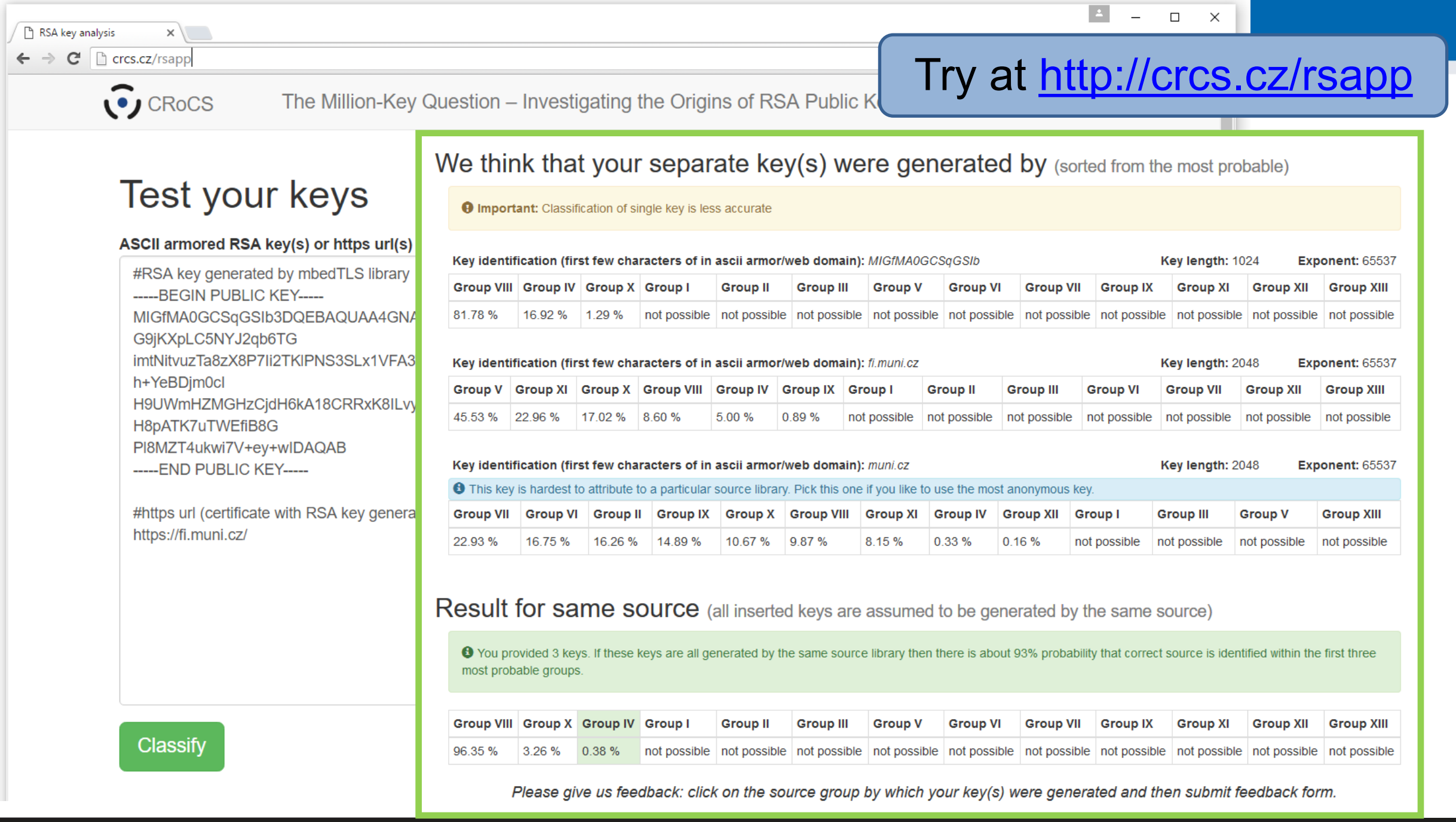
# Classification

44% OpenSSL's group

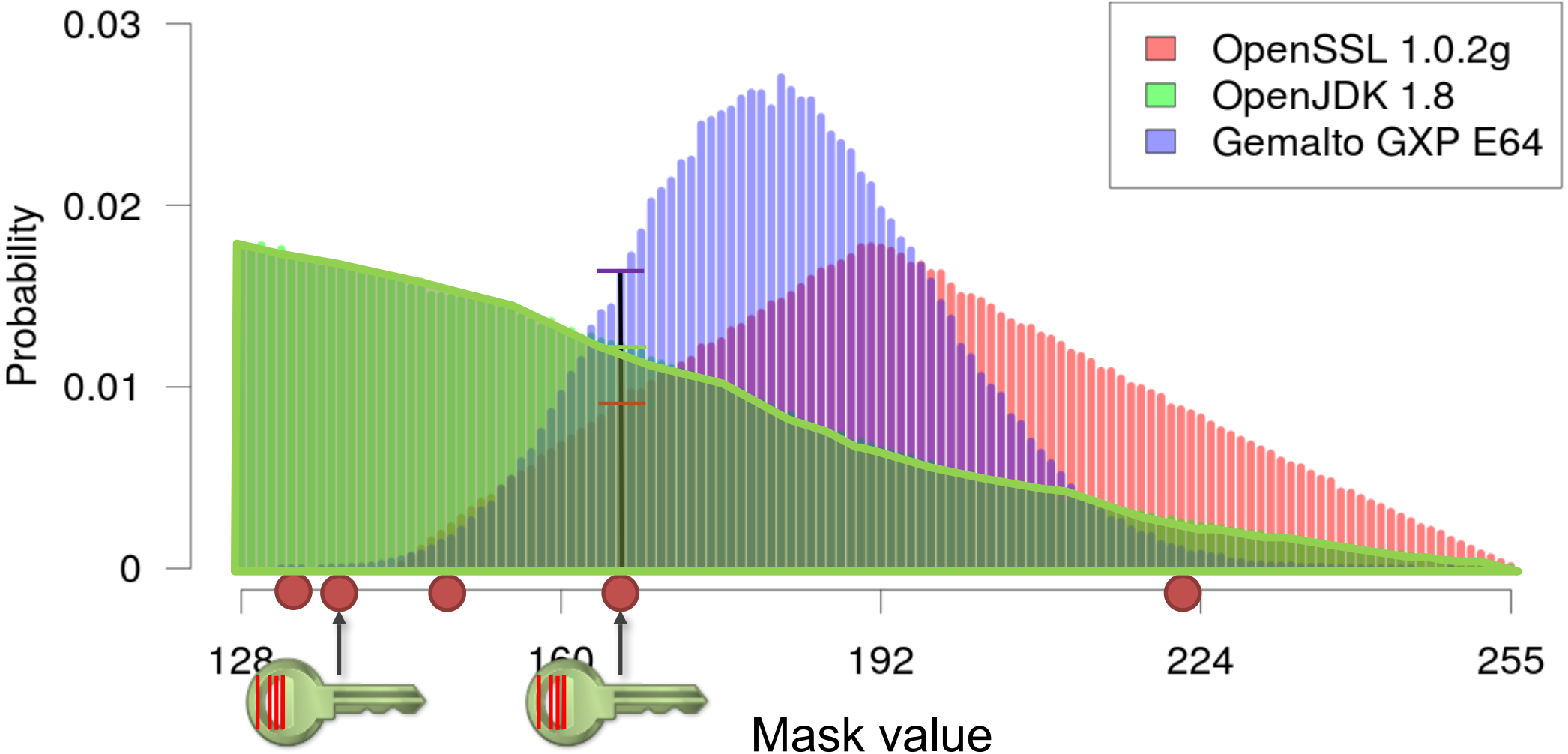
11% POLAR SSL's group

9% PGP's group

…


RSA key analysis
crcs.cz/rsapp
CRoCS
The Million-Key Question – Investigating the Origins of RSA Public K
Try at http://crcs.cz/rsapp
Test your keys
ASCII armored RSA key(s) or https url(s)
#RSA key generated by mbedTLS library
-----BEGIN PUBLIC KEY-----
MIGfMA0GCSqGSIb3DQEBAQUAA4GNA
G9jKXpLC5NYJ2qb6TG
imtNitvuzTa8zX8P7Ii2TKlPNS3SLx1VFA3
h+YeBDjm0cl
H9UWmHZMGHzCjdH6kA18CRRxK8ILvy
H8pATK7uTWEfiB8G
PI8MZT4ukwi7V+ey+wIDAQAB
-----END PUBLIC KEY-----
#https url (certificate with RSA key genera
https://fi.muni.cz/
Classify
We think that your separate key(s) were generated by (sorted from the most probable)
Important: Classification of single key is less accurate
Key identification (first few characters of in ascii armor/web domain): MIGfMA0GCSqGSIb
Key length: 1024
Exponent: 65537
Group VIII 81.78 %
Group IV 16.92 %
Group X 1.29 %
Group I not possible
Group II not possible
Group III not possible
Group V not possible
Group VI not possible
Group VII not possible
Group IX not possible
Group XI not possible
Group XII not possible
Group XIII not possible
Key identification (first few characters of in ascii armor/web domain): fi.muni.cz
Key length: 2048
Exponent: 65537
Group V 45.53 %
Group XI 22.96 %
Group X 17.02 %
Group VIII 8.60 %
Group IV 5.00 %
Group IX 0.89 %
Group I not possible
Group II not possible
Group III not possible
Group VI not possible
Group VII not possible
Group XII not possible
Group XIII not possible
Key identification (first few characters of in ascii armor/web domain): muni.cz
Key length: 2048
Exponent: 65537
This key is hardest to attribute to a particular source library. Pick this one if you like to use the most anonymous key.
Group VII 22.93 %
Group VI 16.75 %
Group II 16.26 %
Group IX 14.89 %
Group X 10.67 %
Group VIII 9.87 %
Group XI 8.15 %
Group IV 0.33 %
Group XII 0.16 %
Group I not possible
Group III not possible
Group V not possible
Group XIII not possible
Result for same source (all inserted keys are assumed to be generated by the same source)
You provided 3 keys. If these keys are all generated by the same source library then there is about 93% probability that correct source is identified within the first three most probable groups.
Group VIII 96.35 %
Group X 3.26 %
Group IV 0.38 %
Group I not possible
Group II not possible
Group III not possible
Group V not possible
Group VI not possible
Group VII not possible
Group IX not possible
Group XI not possible
Group XII not possible
Group XIII not possible
Please give us feedback: click on the source group by which your key(s) were generated and then submit feedback form.

# Classification accuracy



# Classification accuracy (test set, 10k keys/source)

| # keys in batch | Top 1 match 1 | 2 | 5 | 10 |
|---|---|---|---|---|
| Group I | 95.39% | 98.42% | 99.38% | 99.75% |
| Group II | 17.75% | 32.50% | 58.00% | 69.50% |
| Group III | 45.36% | 72.28% | 93.17% | 98.55% |
| Group IV | 90.14% | 97.58% | 99.80% | 100.00% |
| Group V | 63.38% | 81.04% | 97.50% | 99.60% |
| Group VI | 54.68% | 69.22% | 88.45% | 94.60% |
| Group VII | 7.58% | 31.69% | 64.21% | 82.35% |
| Group VIII | 15.65% | 40.30% | 68.46% | 76.60% |
| Group IX | 22.22% | 45.12% | 76.35% | 83.00% |
| Group X | 0.63% | 6.33% | 27.42% | 42.74% |
| Group XI | 11.77% | 28.40% | 55.56% | 65.28% |
| Group XII | 60.36% | 79.56% | 97.20% | 99.40% |
| Group XIII | 39.56% | 70.32% | 96.20% | 99.70% |
| **Average** | **40.34%** | **57.90%** | **78.59%** | **85.47%** |

1 key

Top 1: avg. **40.34%**, min. 0.63%, max. 95.36%

Top 3: avg. **73.09%**, min. 39.32%, max. 98.41%

5 keys

Top 1: avg. **78.59%**, min. 27.42%, max. 99.38%

Top 3: avg. **97.48%**, min. 91.45%, max. 100.00%

10 keys

Top 1: avg. **85.47%**, min. 42.74%, max. 100.00%

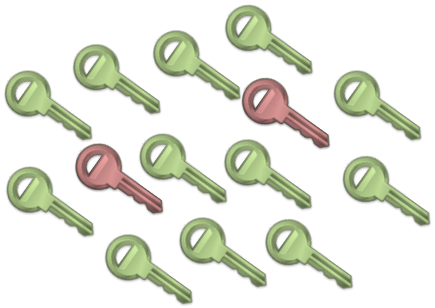
Top 3: avg. **99.27%**, min. 95.00%, max. 100.00%

How we can use classification in real world?

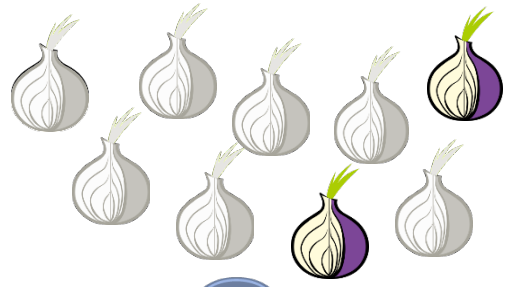
# APPLICATION OF CLASSIFICATION

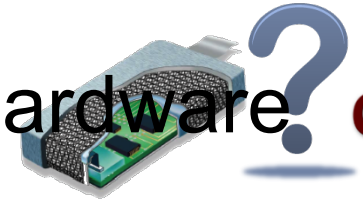
# Impact (of the possibility) of public key classification

- Information leakage vulnerability

- Statistics: current usage trends (TLS/SSH…)
- Quick search for other keys from vulnerable library
- Forensics: source lib/device of weak keys
- De-anonymization: linking Tor hidden services
- Audit: identify source libs in large organization
- Audit: verify Crypto-as-a-Service use of secure hardware



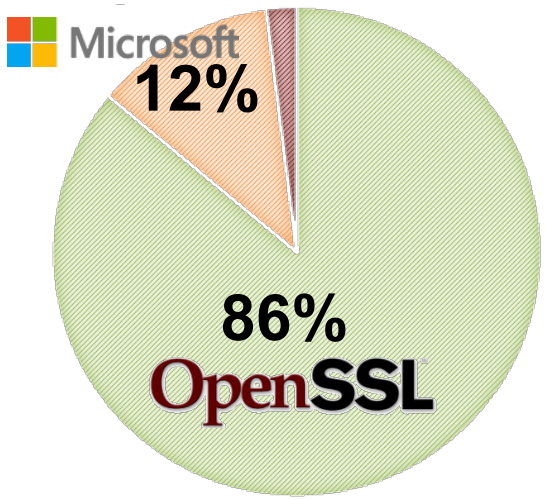


OpenSSL

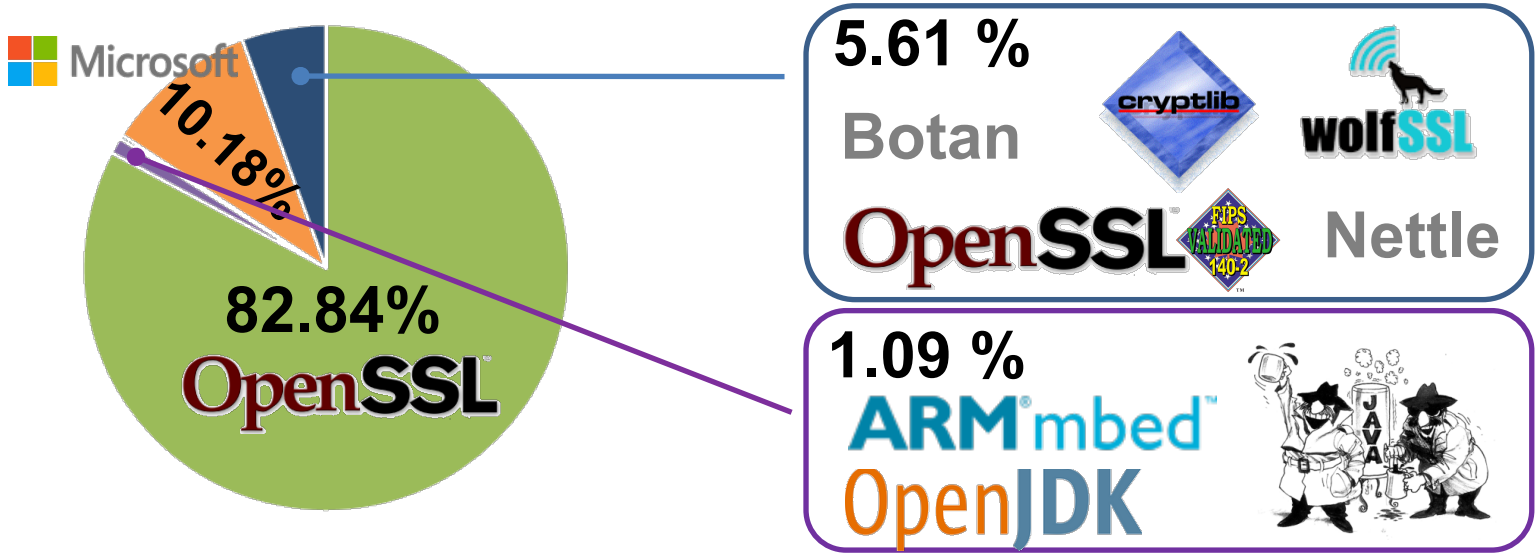
# Sanity check with real world keys: IPv4 TLS dataset

- Datasets: IPv4 TLS scan(10M), PGP(1.4M), Cert. Transparency(13M)…
  - Problem: keys in these datasets are not annotated with source library
- Web servers market share => OpenSSL (~86%), Microsoft (~12%)

**Expected**



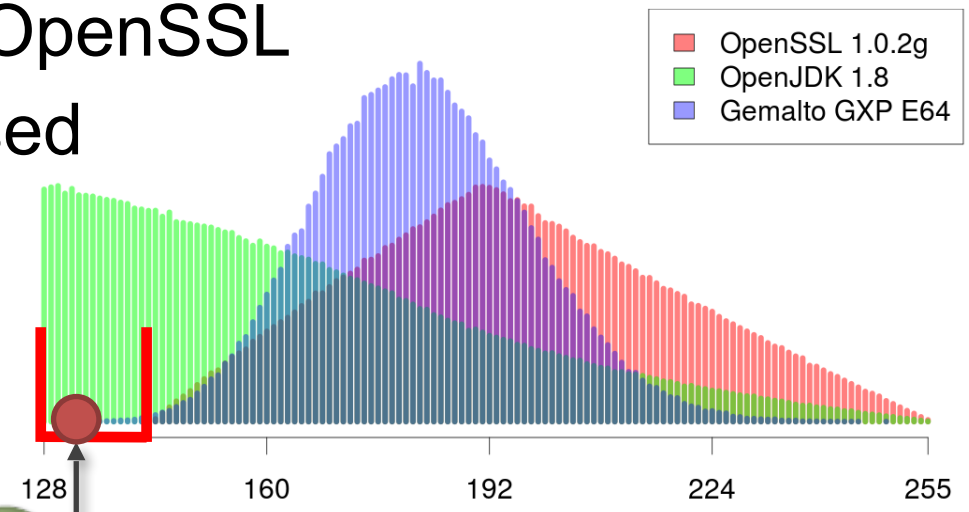

**Classified (10-99 keys with same subject and issue date)**



# Sanity check: keys which *cannot* be from OpenSSL

- Keys with mask value never generated by OpenSSL
- Advantage: all keys from dataset can be used

| Dataset | !OpenSSL |
|---|---|
| Cert. Transparency [16] | 11.80% |
| PGP keyset [54] | 47.35% |
| TLS IPv4 [15] | 18.91% |
| Let's Encrypt [15] | 1.83% |



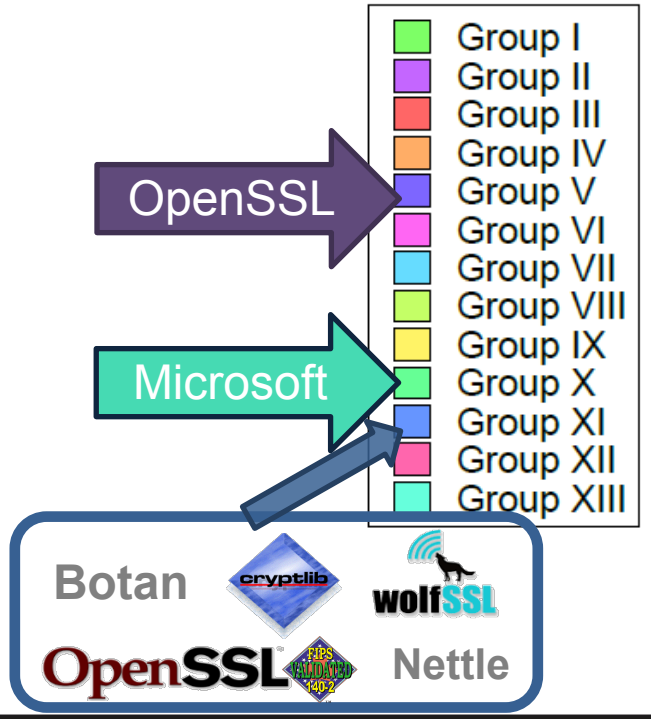

OpenSSL rare in PGP

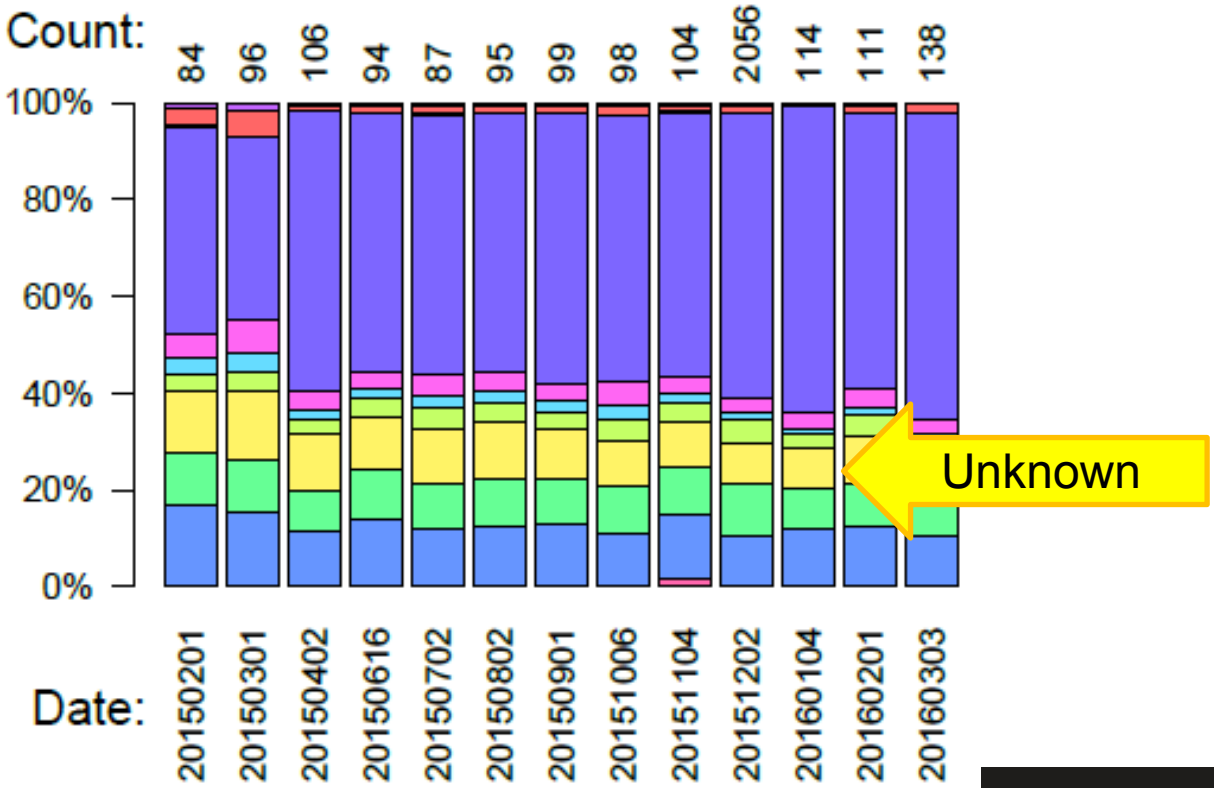
Leaves ~81 % for OpenSSL

OpenSSL is default client

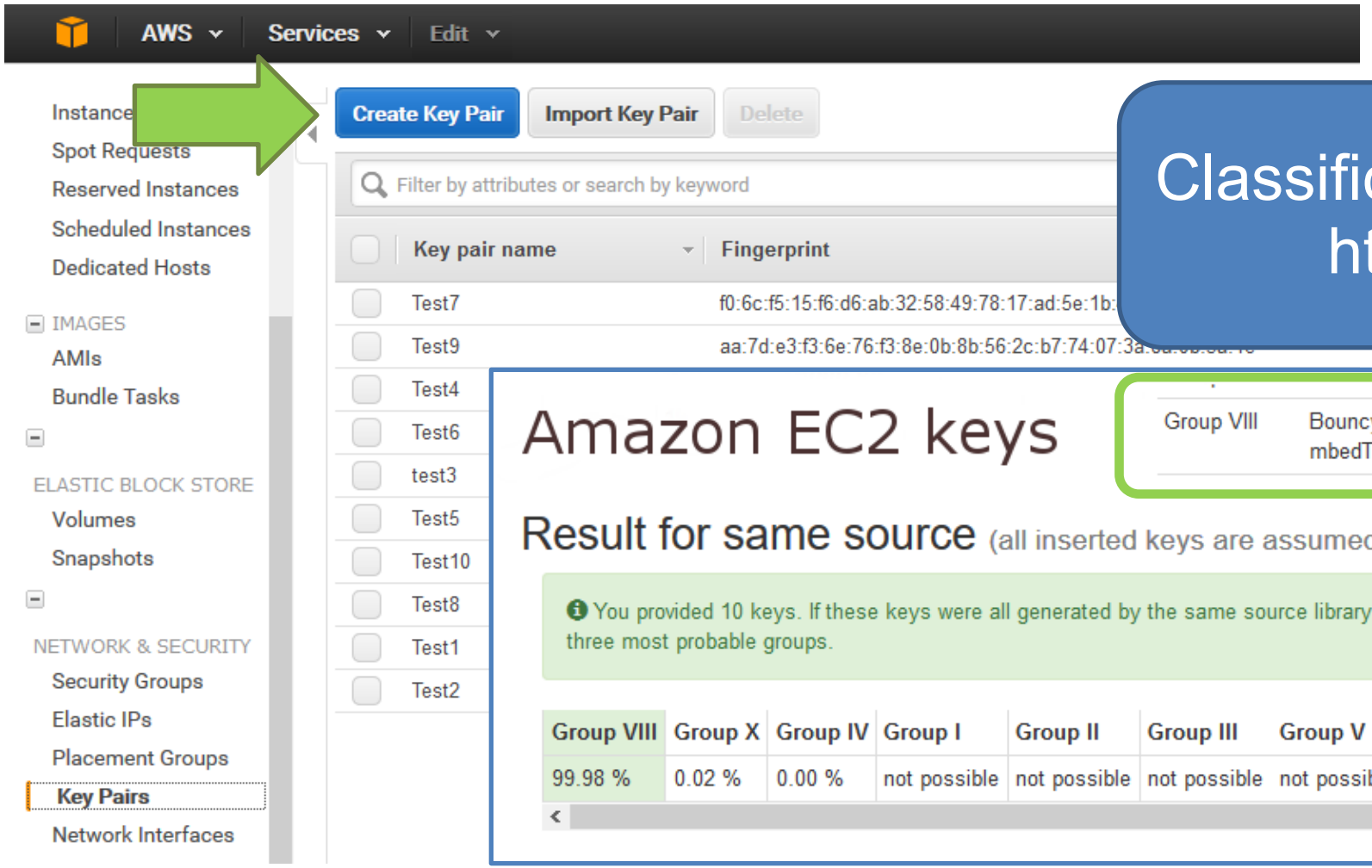
# Evolution of TLS keys in time

- https://scans.io publishes periodic scan results every month
- Changes in key dataset can be tracked



# Audit: What Amazon EC2 uses to generate RSA keys?




Classification of public keys via http://crcs.cz/rsapp

## Amazon EC2 keys

| | |
|---|---|
| Group VIII | Bouncy Castle 1.53, ~~Cryptix JCE 20050328, FlexiProvider 1.7p7~~, mbedTLS 2.2.1, SunRsaSign (OpenJDK 1.8) |

### Result for same source (all inserted keys are assumed to be generated by the same source)

You provided 10 keys. If these keys were all generated by the same source library then there is a three most probable groups.

| Group VIII | Group X | Group IV | Group I | Group II | Group III | Group V | Group VI |
|---|---|---|---|---|---|---|---|
| 99.98 % | 0.02 % | 0.00 % | not possible | not possible | not possible | not possible | not possibl |

More specific if private key is also inspected

# CloudFlare proxy HTTPS

## Result for same source (all inserted keys are assumed to be generated by the same source)

You provided 7 keys. If these keys were all generated by the same source library then there is about 97% probability that correct source is identified within the first three most probable groups.

OpenSSL 1.0.2g

| Group VI | Group XIII | Group XI | Group III | Group XII | Group VII | Group I | Group II | Group IV | Group V | Group VIII | Group IX | Group X |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 98.86 % | 0.80 % | 0.23 % | 0.10 % | 0.00 % | 0.00 % | not possible | not possible | not possible | not possible | not possible | not possible | not possible |

# Verisign CA

## Result for same source (all inserted keys are assumed to be generated by the same source)

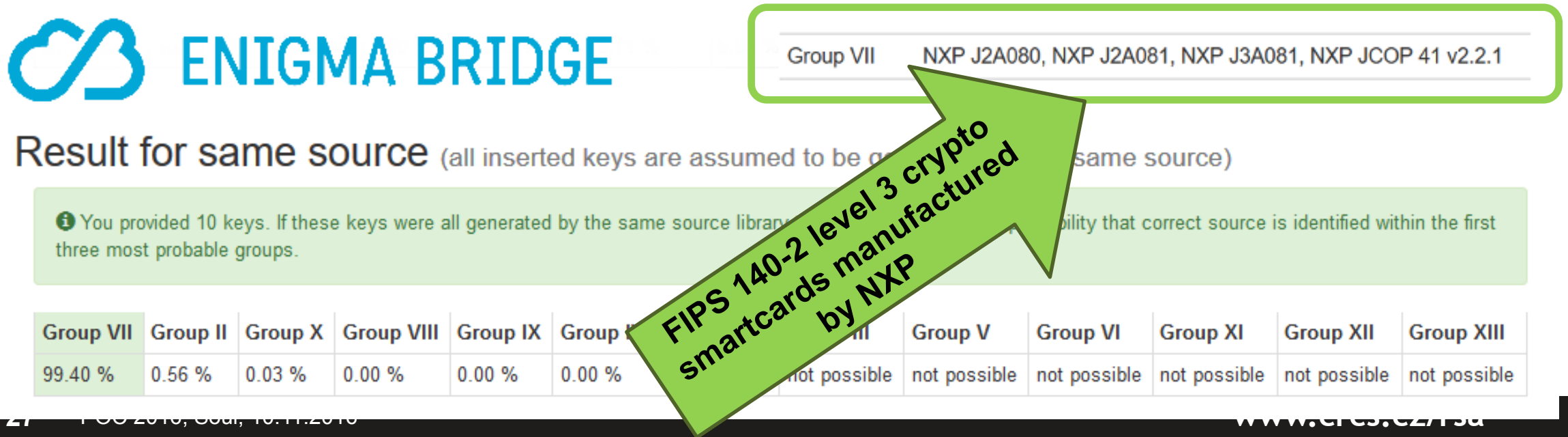
keys. If these keys were all generated by the same source library then there is about 99% probability that correct source is identified within the first groups.

Botan 1.11.29, Feitian JavaCOS A22, Feitian JavaCOS A40, GNU Libgcrypt 1.6.5, GNU Libgcrypt 1.6.5 FIPS, Gemalto GCX 72K, LibTomCrypt 1.17, Nettle 3.2, Oberthur Cosmo 64, OpenSSL FIPS 2.0.12, PGPSDK 4, SafeNet Luna SA-1700 LAN, WolfSSL 3.9.0, cryptlib 3.4.3

| Group XIII | Group XI | Group III | Group XII | Group VII | Group I | Group II | Group IV | Group V | Group VI | Group VIII | Group IX | Group X |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 74.10 % | 18.42 % | 7.46 % | 0.02 % | 0.00 % | not possible | not possible | not possible | not possible | not possible | not possible | not possible | not possible |

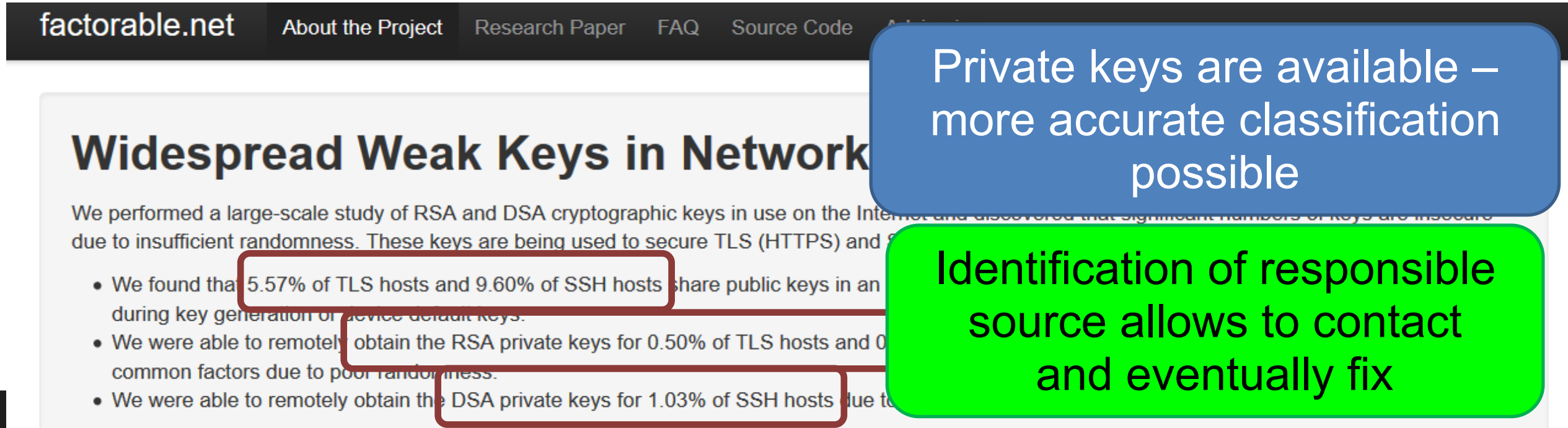
# Audit: Secure hardware behind Crypto-as-a-Service?

- **EnigmaBridge.com** claims key operations in FIPS140-2 certified hardware
- 10 public keys extracted from Enigma Bridge platform via JSON API
  - Private key not extractable



# Forensics: source library/device for factorizable keys

- Internet wide scans, https://www.scans.io/, https://www.censys.io/
- Attempts to factorize fraction of keys from these large scans
  - Shared prime between two or more keys (GCD attack), insufficient entropy during device start, repeated randomness in DSA signatures…



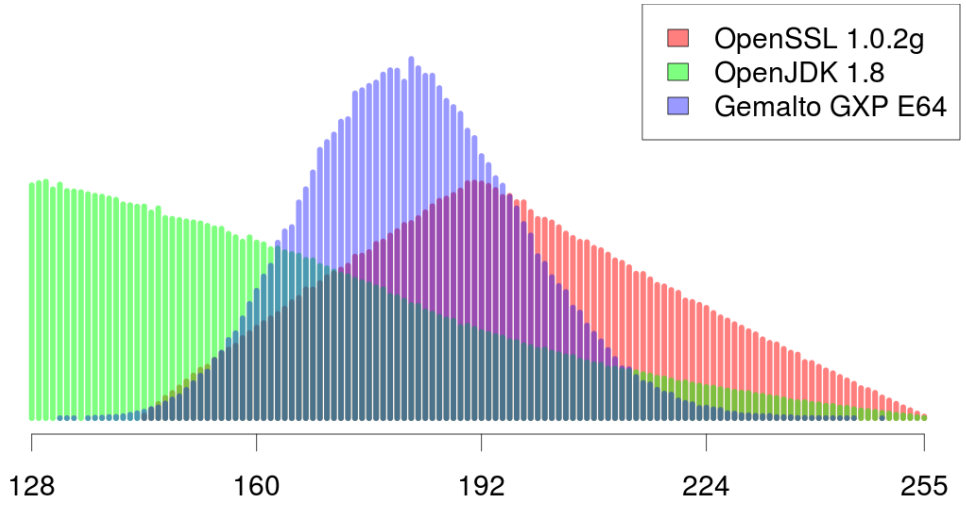
How to defend against possibility of classification?

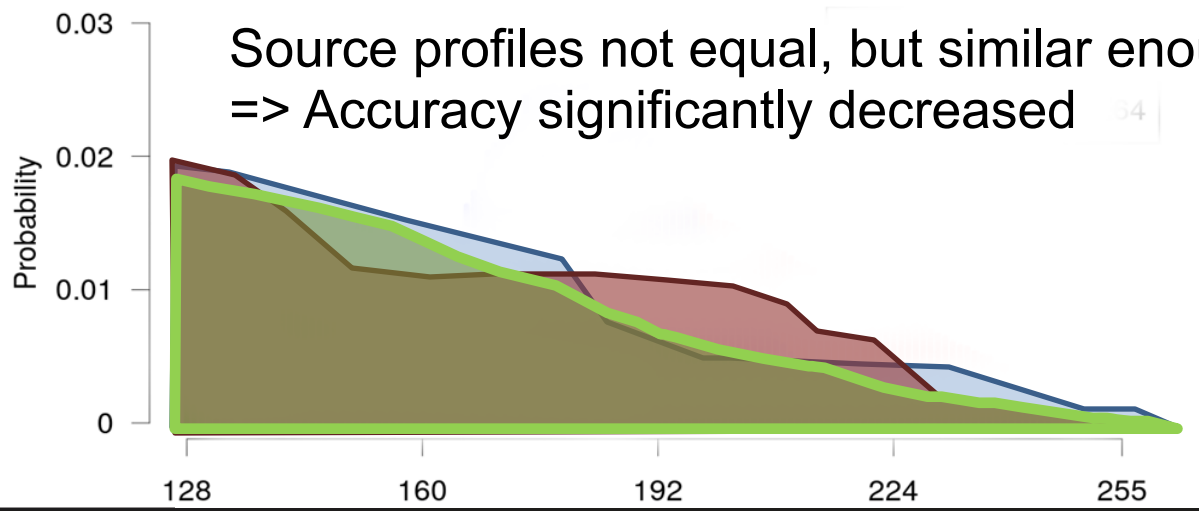
# MITIGATION

# How to defend against public key classification?

## 1. Developers of libraries

- Unify RSA key generation
  - Unlikely to happen soon, changes in critical part of code, legacy binaries…
- Plan to make minimal code changes to libs to decrease accuracy
  - Then Pull requests to upstream



# How to defend against public key classification?

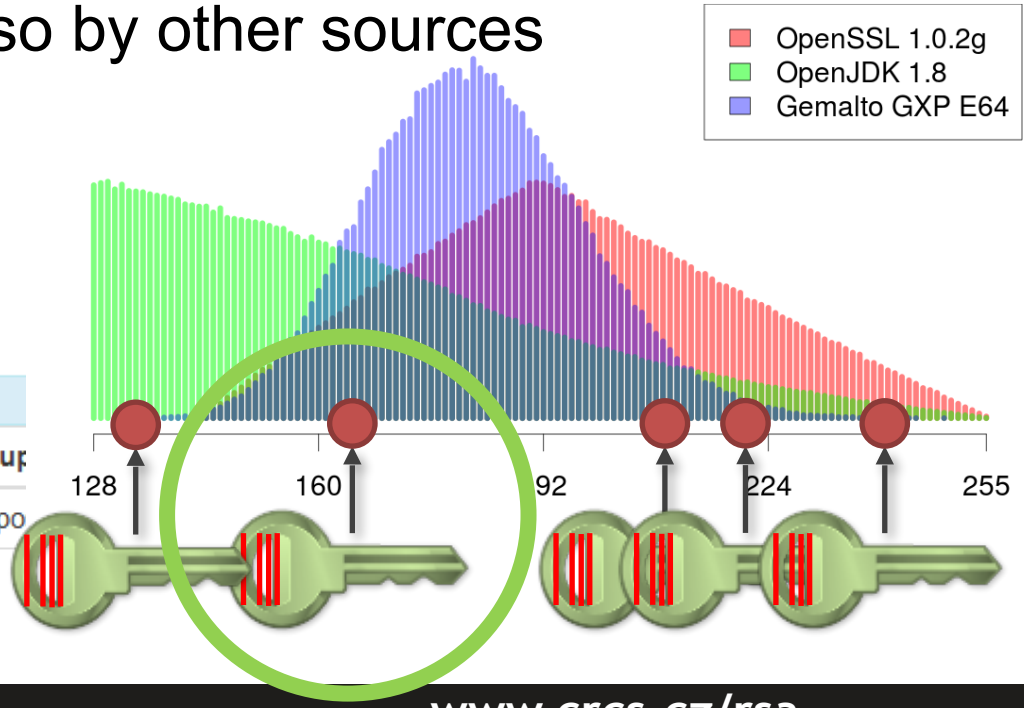
## 2. Users of libraries

- Select one from multiple generated keys
  - Generate multiple keys, pick least “specific” one
  - Key with high probability to be generated also by other sources
  - Only about 5 keys required on average
  - http://crcs.cz/rsapp

**Key identification (first few characters of in ascii armor/web domain):** *muni.cz*

This key is hardest to attribute to a particular source library. Pick this one if you like to use the most anonymous key.

| Group VII | Group VI | Group II | Group IX | Group X | Group VIII | Group XI | Group IV | Group XII | Group |
|---|---|---|---|---|---|---|---|---|---|
| 22.93 % | 16.75 % | 16.26 % | 14.89 % | 10.67 % | 9.87 % | 8.15 % | 0.33 % | 0.16 % | not po |

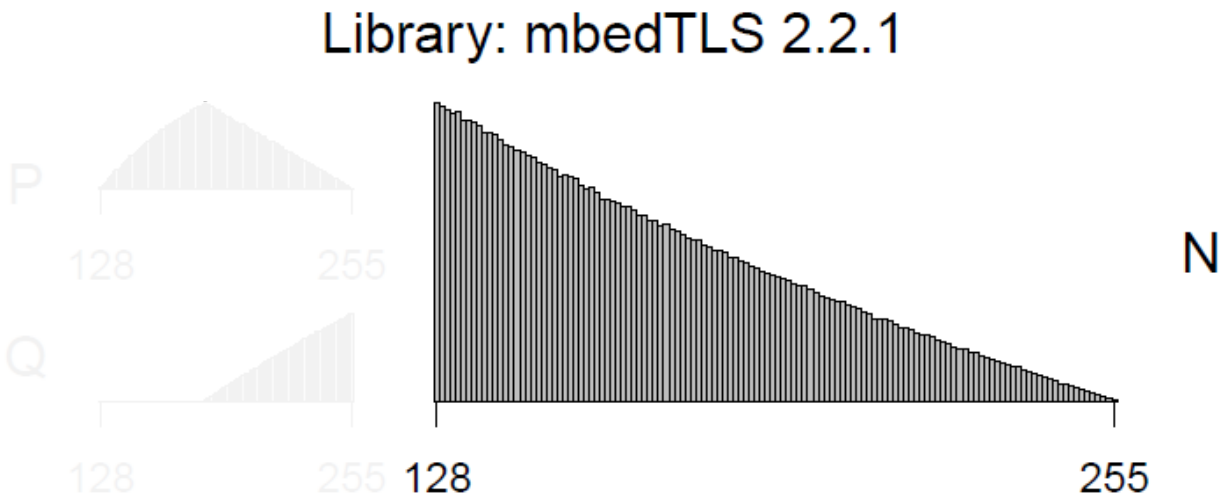


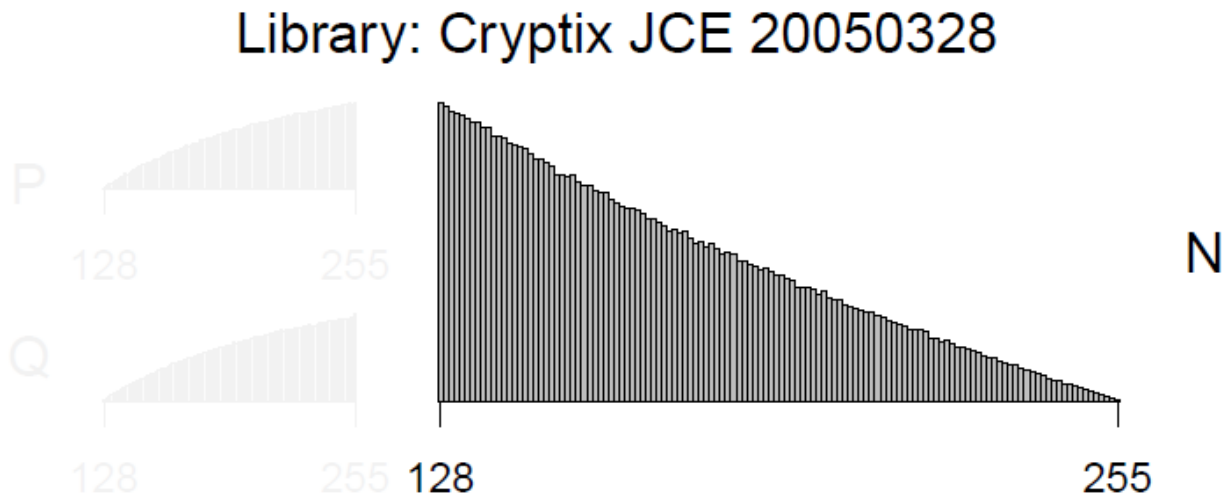
# Limitations of the current work

1. Lower accuracy with single key only (40% on avg.)
2. Can't distinguish all libraries mutually (groups)
   - Better results if private key is available
3. Some sources missing (HSMs…)
   - Will be misclassified at the moment
   - Adding more sources, please contribute!
4. Can't distinguish versions of libs
   - Until key generation algorithm changes

# WHAT IF PRIVATE KEYS ARE AVAILABLE?

# More information available in private keys



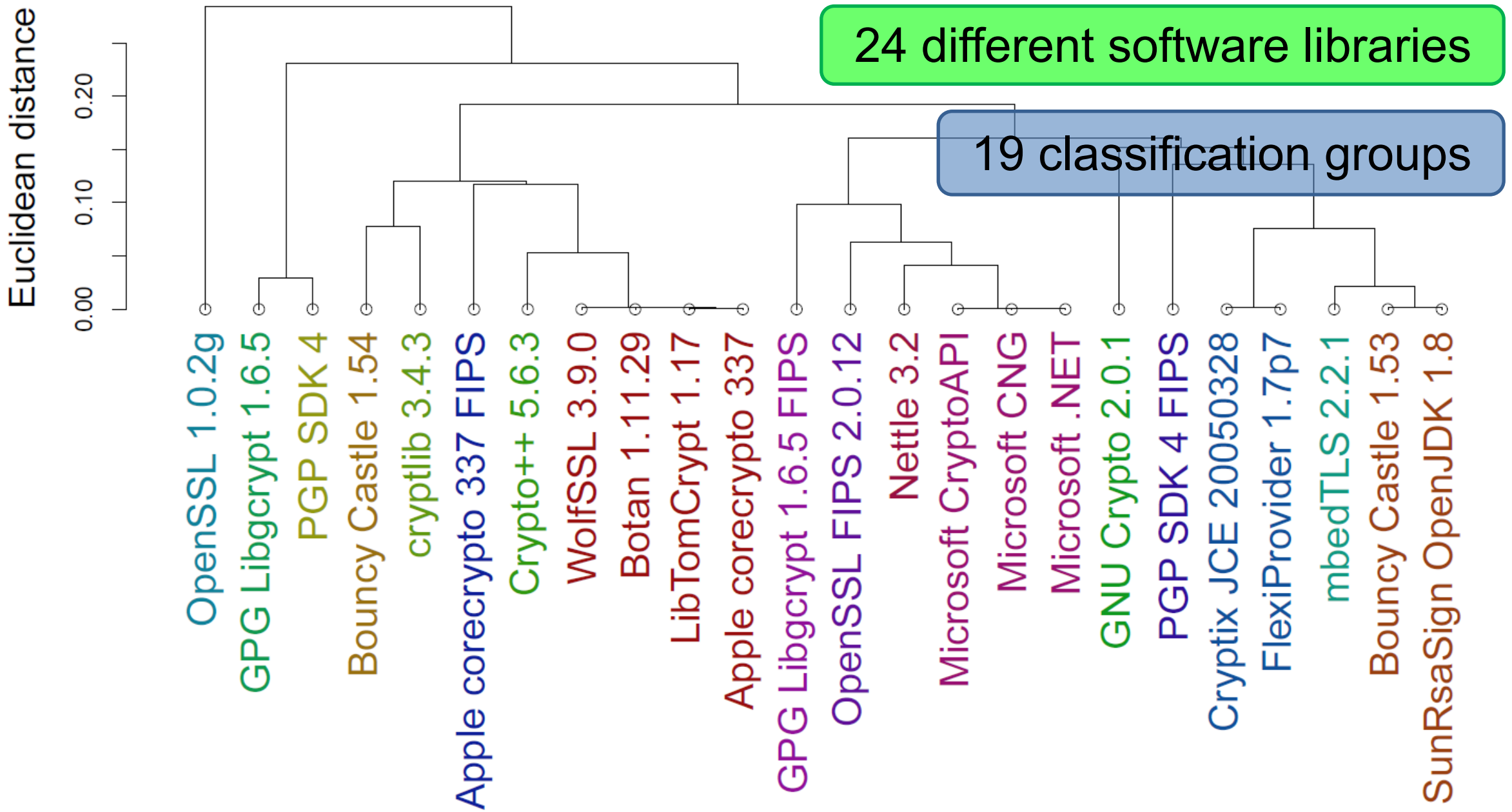
# Difference in libraries based on public keys




24 different software libraries

8 classification groups

# Difference in libraries based on private keys and factorization

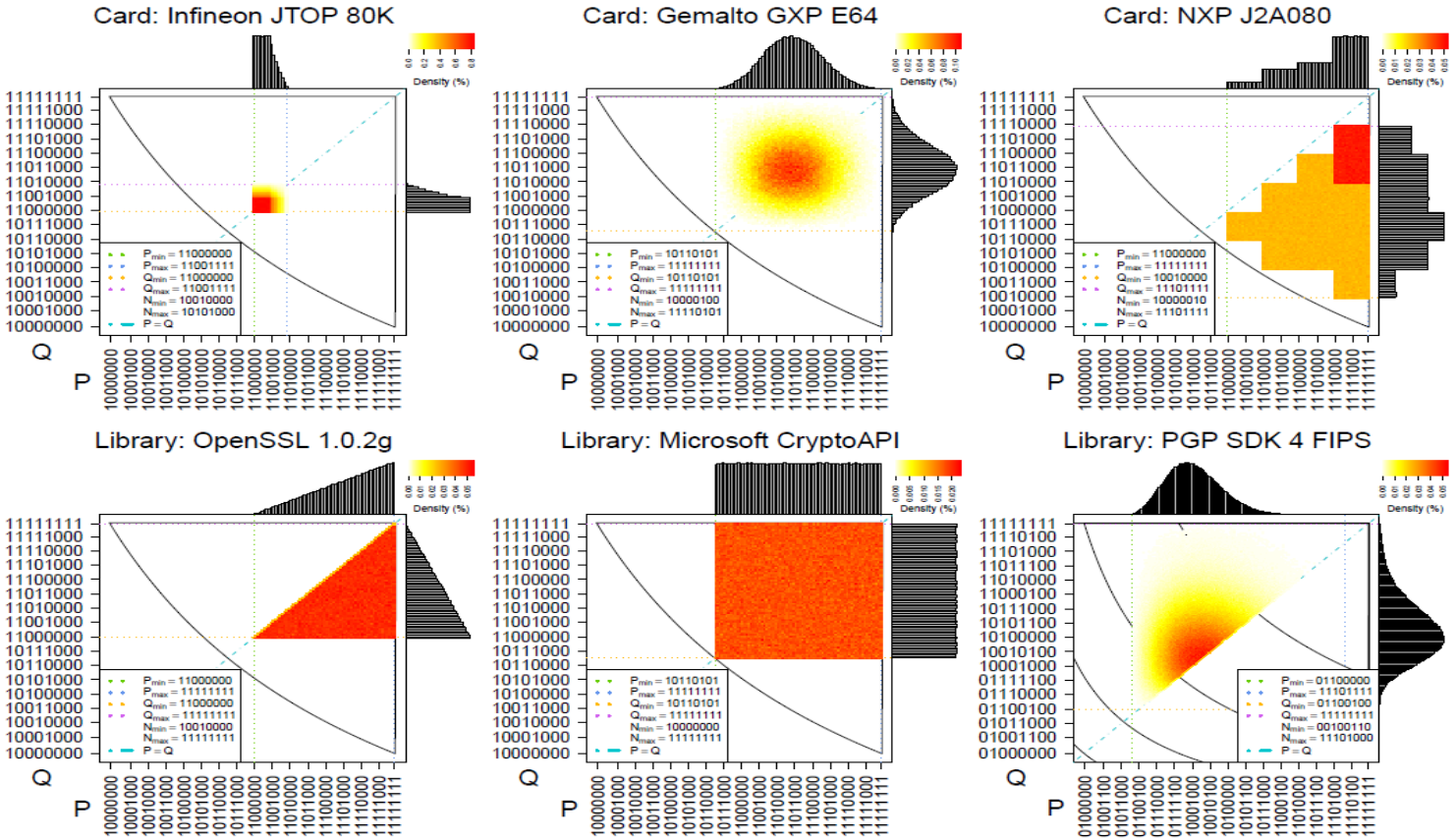


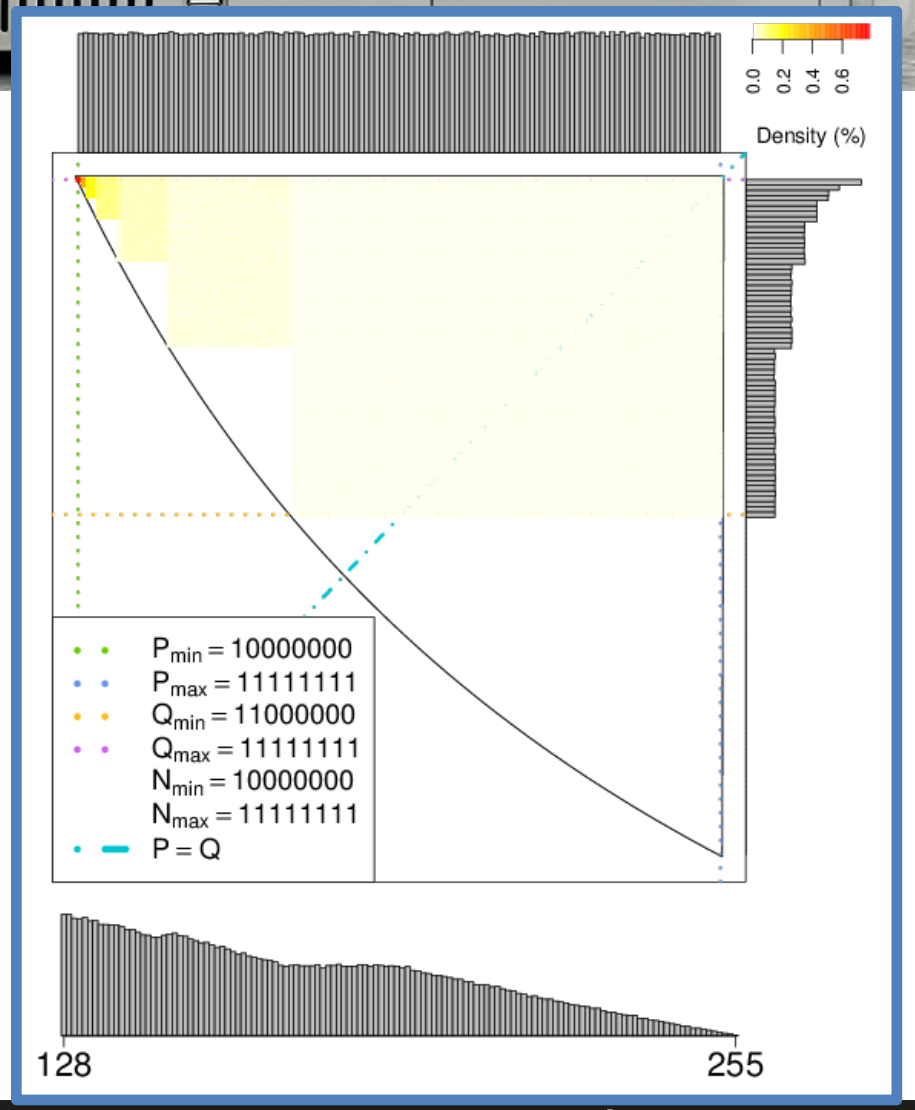

24 different software libraries

19 classification groups

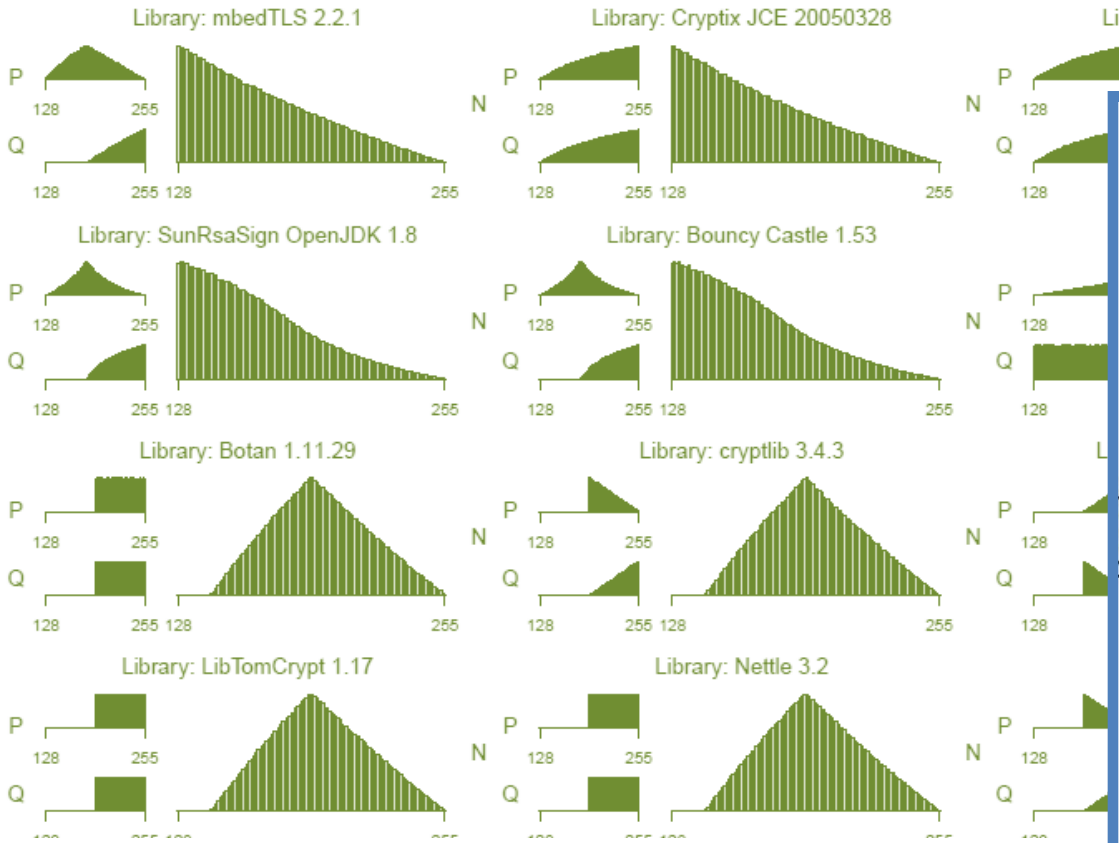
# ADDING MORE SOURCES

# Utimaco Se50 LAN HSM

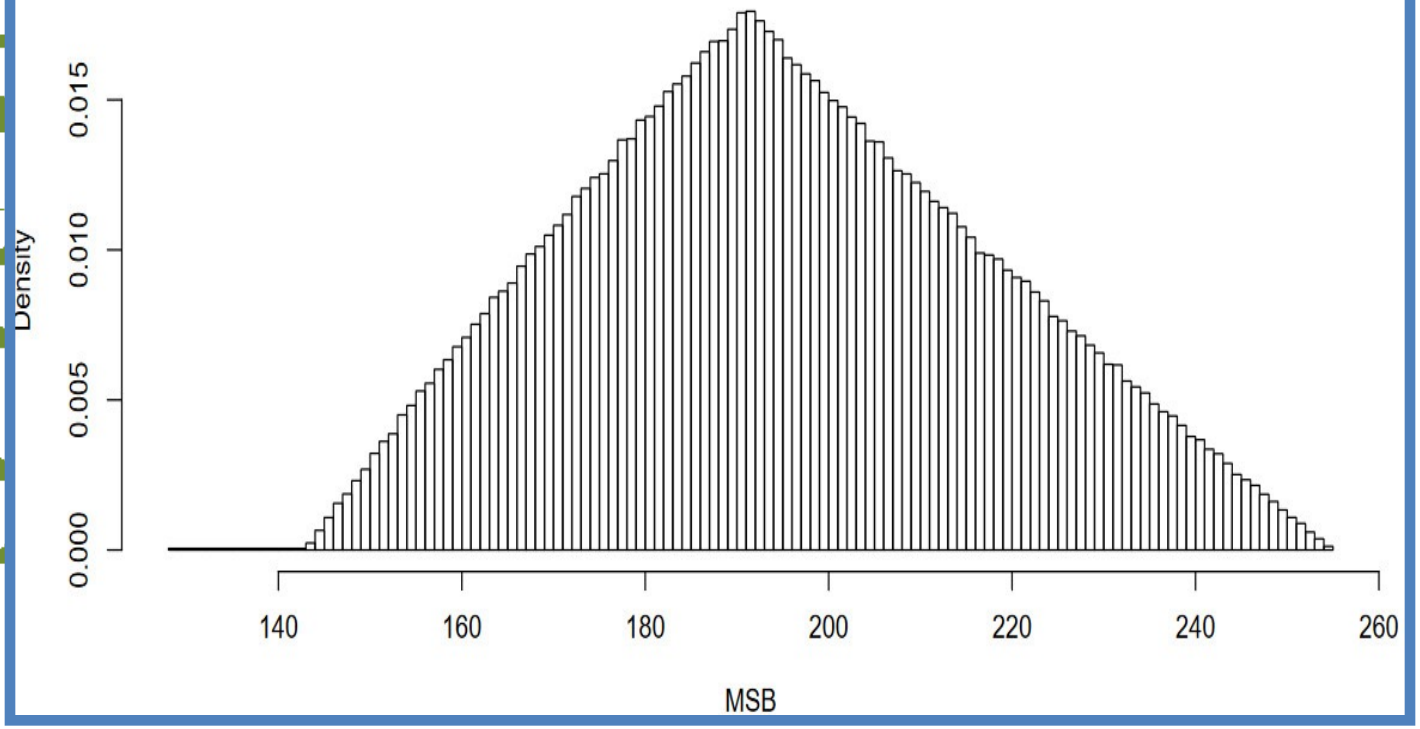



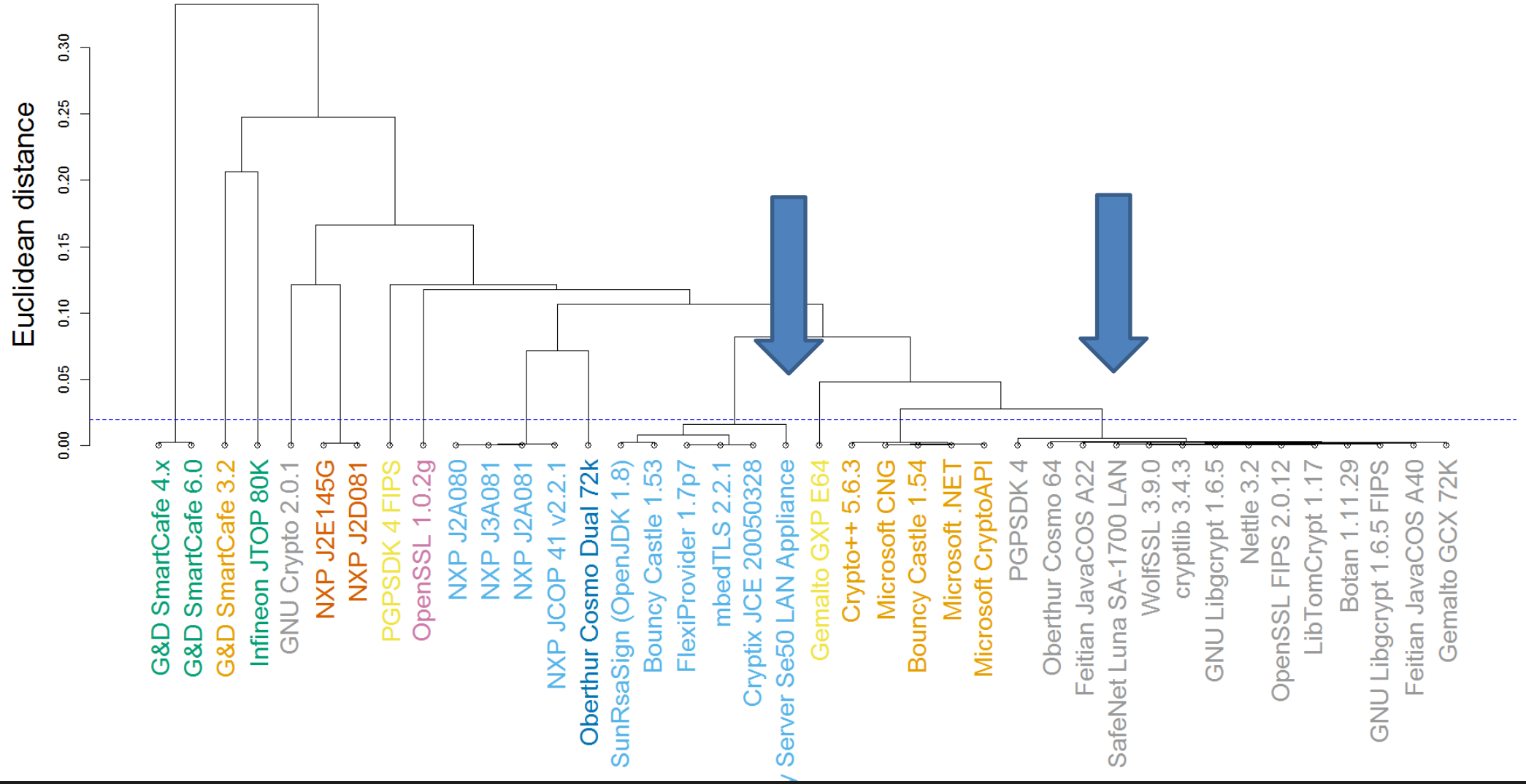
# Safenet Luna SA-1700 LAN


MSB of modulus – private keys not available

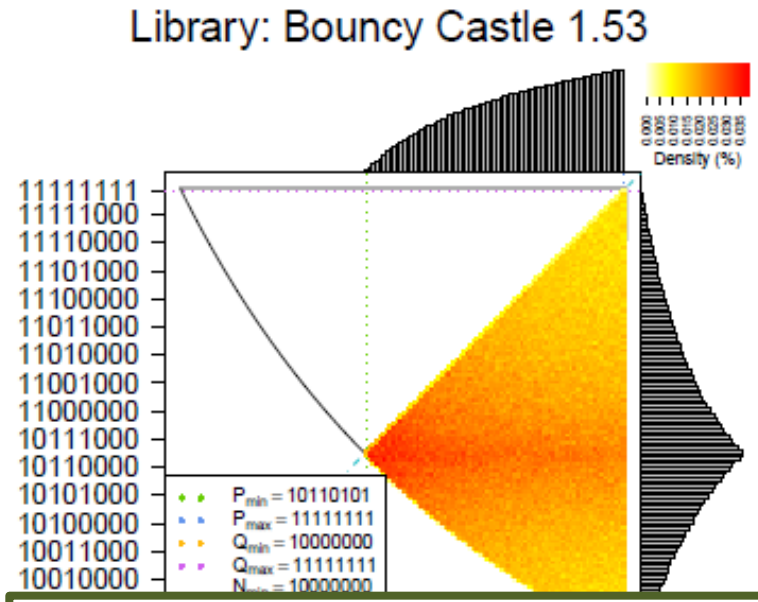


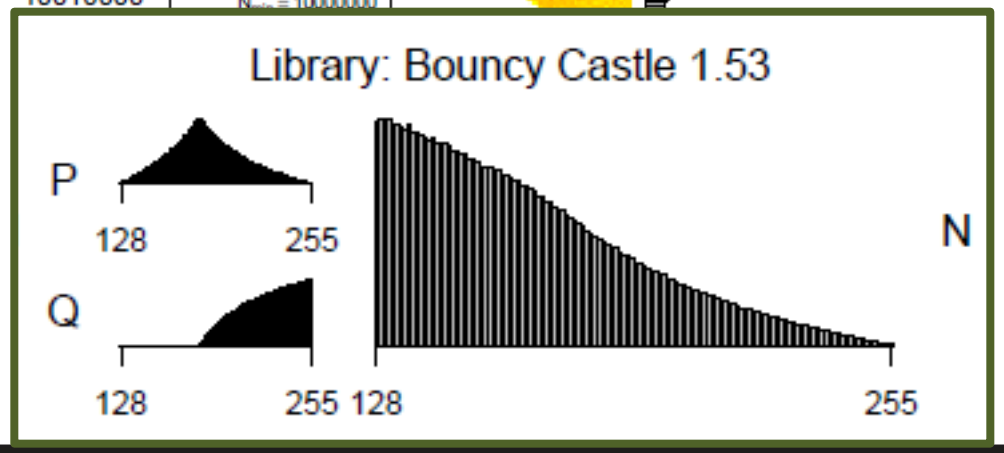
# Updated classification table with HSMs (public key)

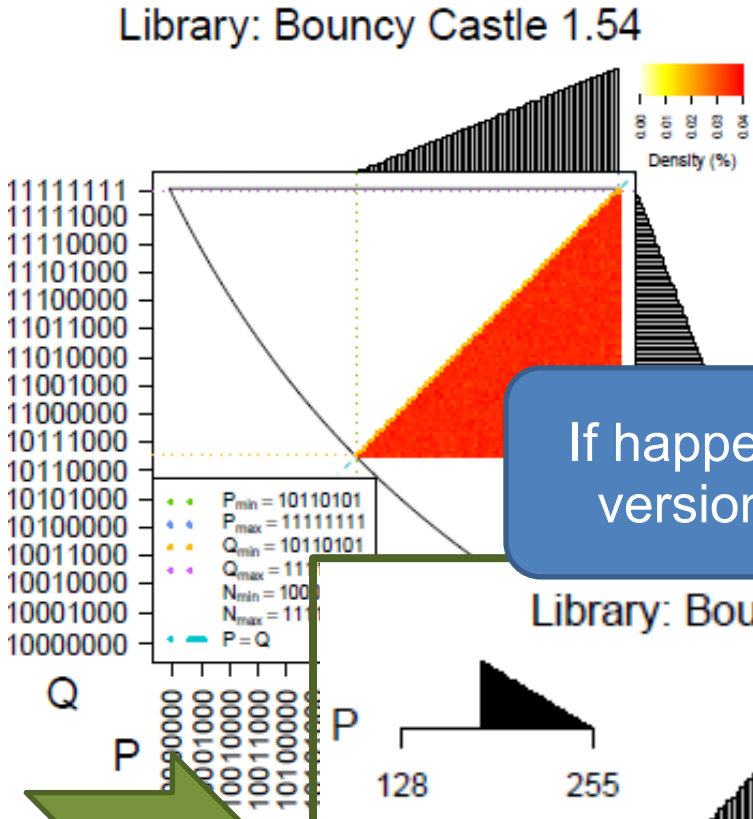


# DETECTION OF LIBS VERSION RANGE

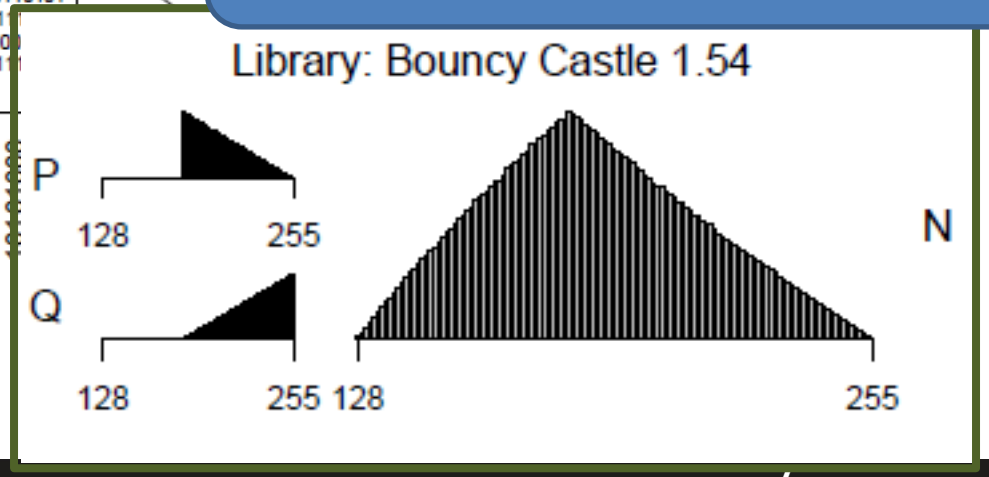
# Occasional change with library/device revision




If happens, different ranges of versions can be recognized



https://github.com/bcgit/bc-java/blob/r1rv54/core/src/main/java/org/bouncycastle/crypto/generators/RSAKeyPairGenerator.java

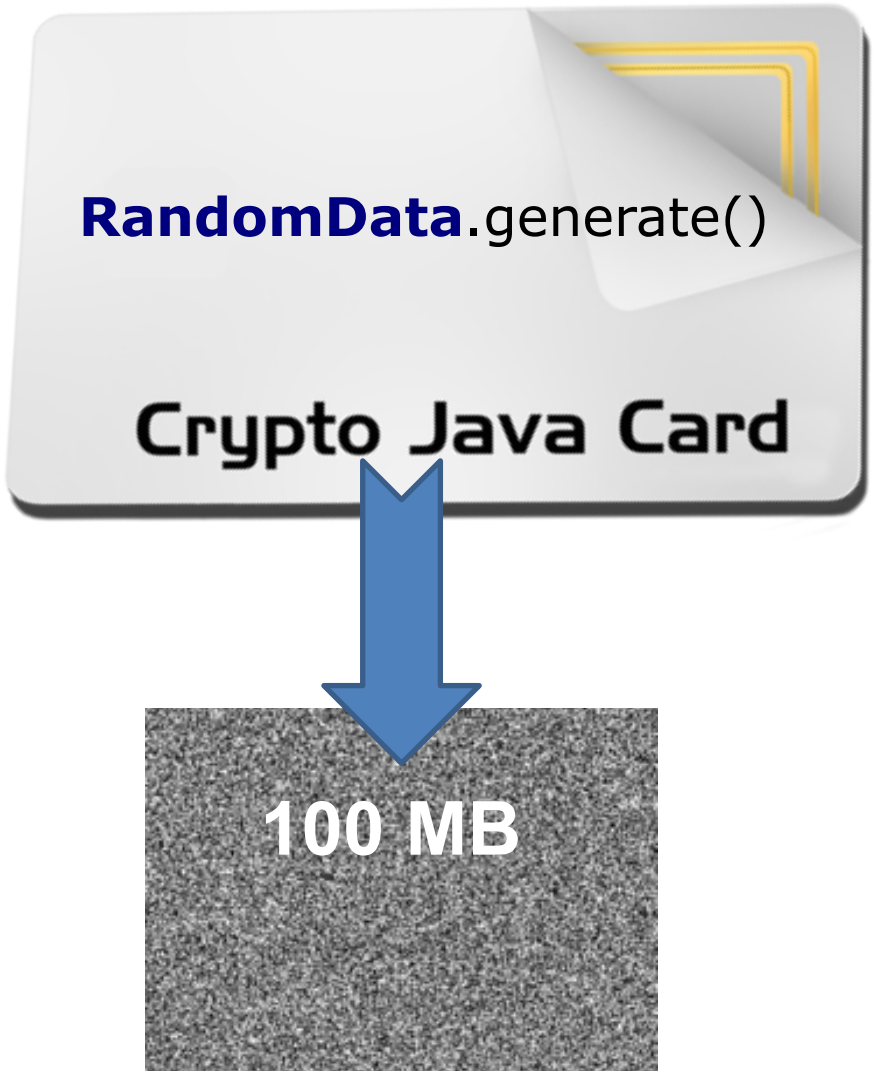
# What changed between BC 1.53 and 1.54?

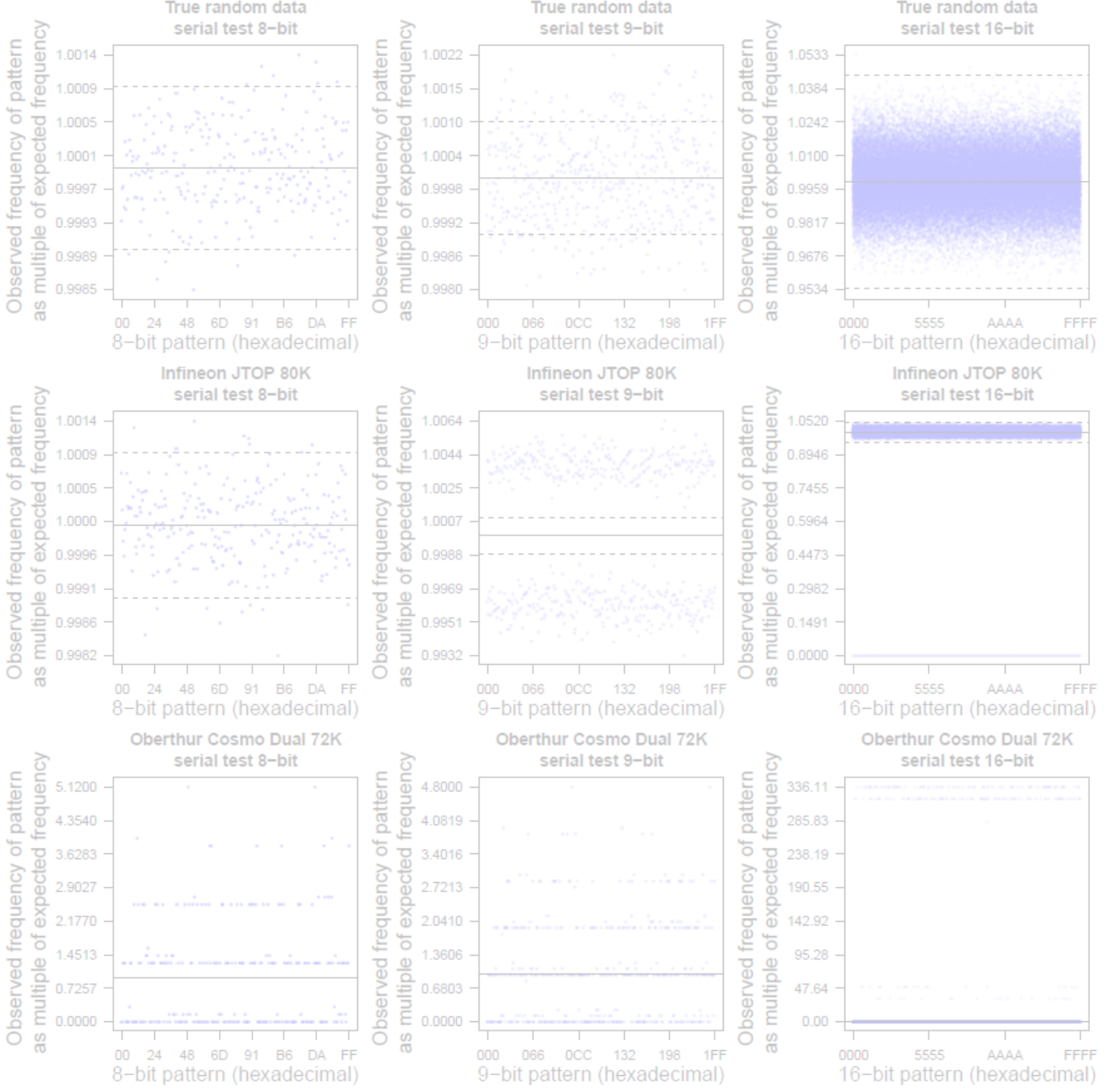
```
protected BigInteger chooseRandomPrime(int bitlength, BigInteger e, BigInteger sqrdBound) {
    for (int i = 0; i != 5 * bitlength; i++)  {
        BigInteger p = new BigInteger(bitlength, 1, param.getRandom());

        if (p.mod(e).equals(ONE)) {
            continue;
        }
        if (p.multiply(p).compareTo(sqrdBound) < 0) {
            continue;
        }
        if (!isProbablePrime(p)) {
            continue;
        }
        if (!e.gcd(p.subtract(ONE)).equals(ONE)) {
            continue;
        }
        return p;
    }
    throw new IllegalStateException("unable to generate prime number for RSA key");
}
```

Added in BC 1.54

How are RSA keys generated on cryptographic smartcards

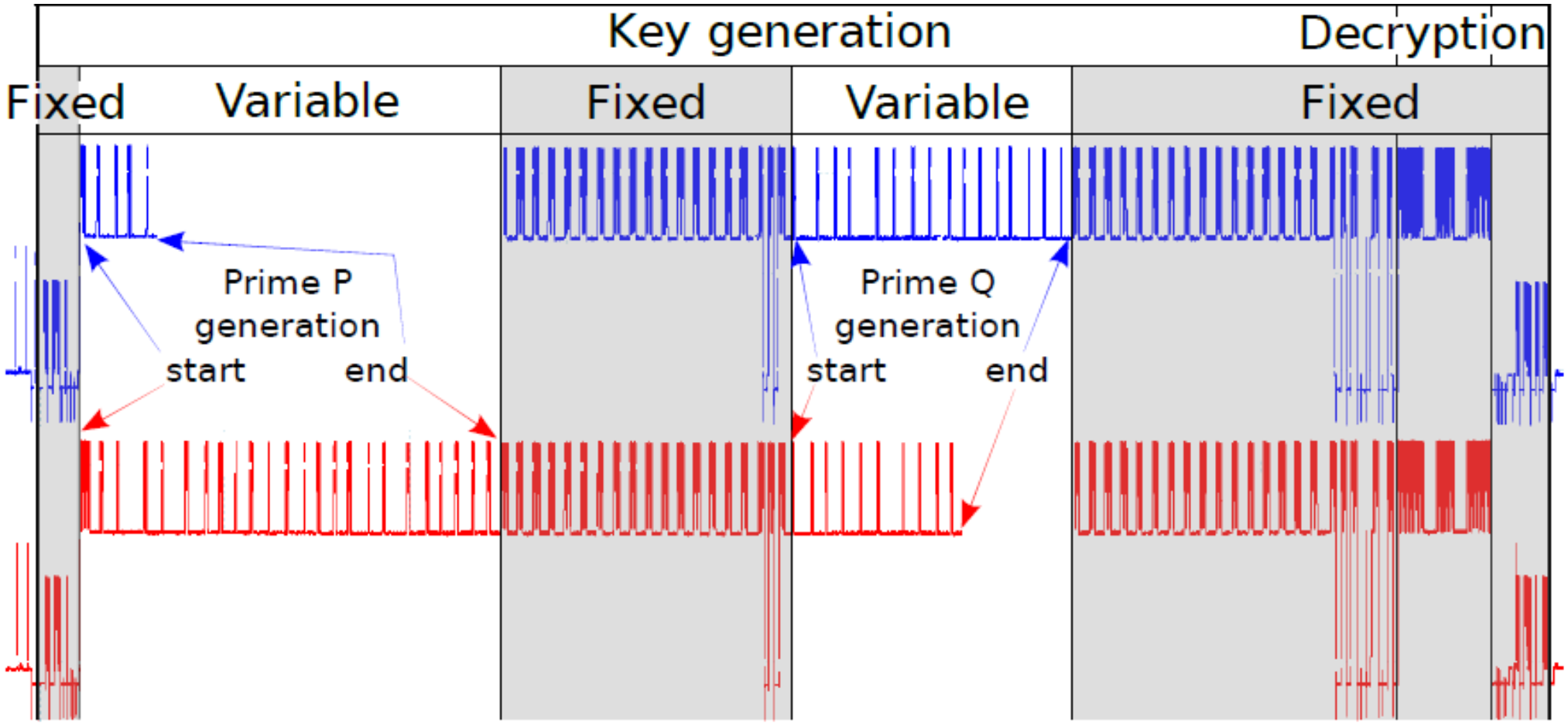
# RSA ON SMARTCARDS




## Serial RNG test

# Simple power analysis of RSA keypair generation



# Incremental search for primes

What if random number generation in step 2 fails?

Factorizable keys observed in 0.05% cases for Oberthur Cosmo Dual 72K
0x800000000…00005f is first prime starting from 0x800…000

1. Zeroize buffer
2. Generate random number into buffer => candidate prime
3. Manipulate few bits to ensure proper length and odd number
4. Test if candidate value is prime
5. If YES, jump to step 7
6. If NO, increment candidate value by 2 and jump to step 4
7. Continue with second prime and RSA keypair generation…

# CONCLUSIONS

# Conclusions

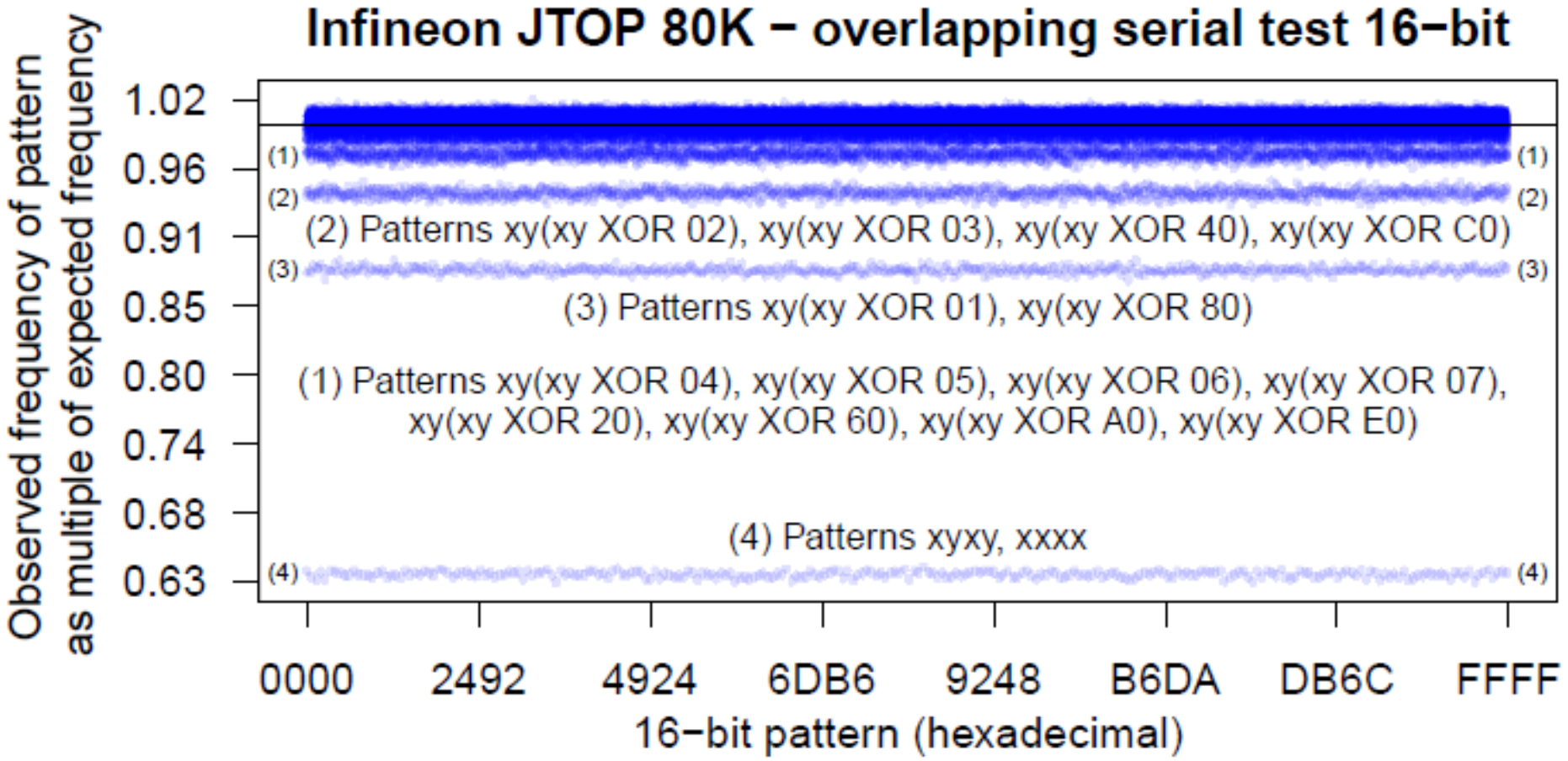
- RSA keypair generation observably bias public keys
  - Different libraries use different implementation choices
- Source library can be probabilistically estimated from RSA public key
  - Accuracy more than 85 % with 10 keys (>99 % within top three matches)
  - For some sources, even a single key is enough
- Information disclosure vulnerability
  - Forensics, de-anonymization, vulnerability scans, compliancy testing…

Questions

*Get tech. report and datasets at http://crcs.cz/rsa, try classification at http://crcs.cz/rsapp*

# BACKUP SLIDES

# Infineon JTOP 80K M8.4



# Oberthur Cosmo Dual 72K