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
RSA public key

\[ N = 9782D7123C330444C88E279BF321EE84AC39524F1D84026327B04F32E1E930FC81588010178DC75FCBF8258A068071317245D08817988813C4173495A922A41DA429A964F738020076EFFE7ED5811088873C6E58EEF1CDC900596681E2CEBE72368B51A821FC699E9C3FD66B377E2DF2485DC401DD99CC125890E5D969A6AC8B \]

\[ e = 10001 \]
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
LEARNING PHASE

Analysis of large number of RSA keys
22 software libraries and versions

16 types of smart cards

G&D
G&D Crypto Java Card
Infini
Infini Crypto Java Card
G&D
G&D Crypto Java Card
Infineon
Infineon Crypto Java Card
Gemalto
Gemalto Crypto Java Card
NXP
NXP Crypto Java Card
Oberthur
Oberthur Crypto Java Card
Feitian
Feitian Crypto Java Card

1 000 000 x
Gen_RSA_keypair()
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 $\varphi^{-1}/\psi^{-1}$ (and its impact on modulus $N$)

- For RSA512b, length of prime is 256bits $\Rightarrow \varphi^{-1}/\psi^{-1}$ 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
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

1. Harvest keys from known sources (learning set)
2. Apply mask to learning set
3. Count mask frequency
4. Group sources with very similar frequencies
5. Normalize mask vectors of groups

Identification of biased modulus bits (mask, 9bits)

Identification of biased modulus bits (mask, 9bits)

Classification matrix

<table>
<thead>
<tr>
<th>Mask value</th>
<th>Group I</th>
<th>Group II</th>
<th>...</th>
<th>Group XII</th>
<th>Group XIII</th>
</tr>
</thead>
<tbody>
<tr>
<td>000000000</td>
<td>0.124</td>
<td>0.347</td>
<td></td>
<td>0.105</td>
<td>0.012</td>
</tr>
<tr>
<td>000000001</td>
<td>0.004</td>
<td>0.038</td>
<td></td>
<td>0.236</td>
<td>0.454</td>
</tr>
<tr>
<td>000000011</td>
<td>0.046</td>
<td>0.002</td>
<td></td>
<td>0.447</td>
<td>0.112</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>111111110</td>
<td>0.394</td>
<td>0.044</td>
<td></td>
<td>0.320</td>
<td>0.002</td>
</tr>
<tr>
<td>111111111</td>
<td>0.046</td>
<td>0.347</td>
<td></td>
<td>0.015</td>
<td>0.312</td>
</tr>
</tbody>
</table>
Tree splits can be attributed to particular implementation choice(s)

38 different sources

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

Group separation threshold

- G&D SmartCafe 4.x
- G&D SmartCafe 6.0
- GNU Crypto 2.0.1
- NXP J2E145G
- NXP J2D081
- PGP SDK 4 FIPS
- OpenSSL 1.0.2g
- Oberthur Cosmo Dual 72K
- NXP J2COP 41 v2.2.1
- NXP J2A081
- NXP J2A080
- NXP J3A081
- Cryptix JCE 20050328
- mbedTLS 2.2.1
- FlexiProvider 1.7p7
- Bouncy Castle 1.53
- SunRsaSign OpenJDK 1.8
- Gemalto GXP E64
- Crypto++ 5.8.3
- 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
Identify origin library or device which generated given key

CLASSIFICATION PHASE
**Input key**

-----BEGIN CERTIFICATE-----
MIIG9zCCBd+gAwIBAgIIJOR2wFUwc20wDQYJKoZIhlvcNAQELBQAwgD8wDQYJKoZIhvcN
AQUAAAADgAwIBAgICDQcGA1UEBhMCVVMxEzARBgNVBAoTHkd2b2dsZSBJbmMxJ
TAjBgNVBAMTREdvb2dsZSBJbmRyeSBBdXRob3JpdHkgRzIwHhcNMTYwNzA2MDgx
NzQzWhcNMTYwOTI4MDgwMzAwk2zlQSqmqHS14N1RoQD9zPk/rEp4miQ9avgC6k7ibLukr4cG
myPc0kCQr8kNUBhH25DS6HpekTmO1s9q81KbtS2E7+4Q/57xgdghBLiaTEv7Q7+gskLQ/qJaTouwiDP
M6SHlVU6X2Ca1lNKg2wbx8h2Q63SD1ywFJ52HsNAClKp4ADvjjImYoWVitcLhpXg0cZ65sJs6Jk=
-----END CERTIFICATE-----

**Precomputed matrix**

<table>
<thead>
<tr>
<th>Mask value</th>
<th>Group I</th>
<th>Group II</th>
<th>...</th>
<th>Group XII</th>
<th>Group XIII</th>
</tr>
</thead>
<tbody>
<tr>
<td>000000000</td>
<td>0.124</td>
<td>0.347</td>
<td></td>
<td>0.105</td>
<td>0.012</td>
</tr>
<tr>
<td>000000001</td>
<td>0.004</td>
<td>0.038</td>
<td></td>
<td>0.236</td>
<td>0.454</td>
</tr>
<tr>
<td>000000011</td>
<td>0.046</td>
<td>0.002</td>
<td></td>
<td>0.447</td>
<td>0.112</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>111111110</td>
<td>0.394</td>
<td>0.044</td>
<td></td>
<td>0.320</td>
<td>0.002</td>
</tr>
<tr>
<td>111111111</td>
<td>0.046</td>
<td>0.347</td>
<td></td>
<td>0.015</td>
<td>0.312</td>
</tr>
</tbody>
</table>

**Classification**

- 44% OpenSSL’s group
- 11% PolarSSL’s group
- 9% PGP’s group
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-----
MIIGA0GCSqGSIb3DQEBAQUAA4GN\nGjKXplLC5NYj2q86TG
ImNItnuz7a$sX9P7i2TkIPNS3SLx1VFA3h=YeBDj0cI
H9WmH2MGHczJdh6ka18CRRwK9iLVy
H8pATK7UTWE98B9
PI8MTFtukW7V+ey=WIDAAB
-----END PUBLIC KEY-----

https url (certificate with RSA key generated)
https://f.muni.cz/

We think that your separate key(s) were generated by (sorted from the most probable)

1. Important: Classification of single key is less accurate

Key identification (first few characters of in ascii armor/web domain):
MIIGMA0GCSqGSIb3DQEBAQUAA4GN\nGjKXplLC5NYj2q86TG
ImNItnuz7a$sX9P7i2TkIPNS3SLx1VFA3h=YeBDj0cI
H9WmH2MGHczJdh6ka18CRRwK9iLVy
H8pATK7UTWE98B9
PI8MTFtukW7V+ey=WIDAAB
-----END PUBLIC KEY-----

Key identification (first few characters of in ascii armor/web domain):

<table>
<thead>
<tr>
<th>Group</th>
<th>Group I</th>
<th>Group II</th>
<th>Group III</th>
<th>Group IV</th>
<th>Group V</th>
<th>Group VI</th>
<th>Group VII</th>
<th>Group VIII</th>
<th>Group IX</th>
<th>Group X</th>
<th>Group XI</th>
<th>Group XII</th>
<th>Group XIII</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key length</td>
<td>1024</td>
<td>65537</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Exponent</td>
<td>65537</td>
<td>2048</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

Key identification (first few characters of in ascii armor/web domain):

<table>
<thead>
<tr>
<th>Group</th>
<th>Group I</th>
<th>Group II</th>
<th>Group III</th>
<th>Group IV</th>
<th>Group V</th>
<th>Group VI</th>
<th>Group VII</th>
<th>Group VIII</th>
<th>Group IX</th>
<th>Group X</th>
<th>Group XI</th>
<th>Group XII</th>
<th>Group XIII</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key length</td>
<td>2048</td>
<td>65537</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Exponent</td>
<td>65537</td>
<td>2048</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

1. 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.

<table>
<thead>
<tr>
<th>Group</th>
<th>Group I</th>
<th>Group II</th>
<th>Group III</th>
<th>Group IV</th>
<th>Group V</th>
<th>Group VI</th>
<th>Group VII</th>
<th>Group VIII</th>
<th>Group IX</th>
<th>Group X</th>
<th>Group XI</th>
<th>Group XII</th>
<th>Group XIII</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key length</td>
<td>96.35%</td>
<td>3.26%</td>
<td>0.36%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Please give us feedback: click on the source group by which your key(s) were generated and then submit feedback form.
Classification accuracy

Mask value

- OpenSSL 1.0.2g
- OpenJDK 1.8
- Gemalto GXP E64

Try online app at http://crcs.cz/rsapp/
Classification accuracy (test set, 10k keys/source)

<table>
<thead>
<tr>
<th># keys in batch</th>
<th>Top 1 match</th>
<th>1</th>
<th>2</th>
<th>5</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group I</td>
<td>95.39%</td>
<td>98.42%</td>
<td>99.38%</td>
<td>99.75%</td>
<td>100.00%</td>
</tr>
<tr>
<td>Group II</td>
<td>17.75%</td>
<td>32.50%</td>
<td>58.00%</td>
<td>69.50%</td>
<td>70.00%</td>
</tr>
<tr>
<td>Group III</td>
<td>45.36%</td>
<td>72.28%</td>
<td>93.17%</td>
<td>98.55%</td>
<td>100.00%</td>
</tr>
<tr>
<td>Group IV</td>
<td>90.14%</td>
<td>97.58%</td>
<td>99.80%</td>
<td>100.00%</td>
<td>100.00%</td>
</tr>
<tr>
<td>Group V</td>
<td>63.38%</td>
<td>81.04%</td>
<td>97.50%</td>
<td>99.60%</td>
<td>100.00%</td>
</tr>
<tr>
<td>Group VI</td>
<td>54.68%</td>
<td>69.22%</td>
<td>88.45%</td>
<td>94.60%</td>
<td>99.00%</td>
</tr>
<tr>
<td>Group VII</td>
<td>7.58%</td>
<td>31.69%</td>
<td>64.21%</td>
<td>82.35%</td>
<td>99.00%</td>
</tr>
<tr>
<td>Group VIII</td>
<td>15.65%</td>
<td>40.30%</td>
<td>68.46%</td>
<td>76.60%</td>
<td>99.00%</td>
</tr>
<tr>
<td>Group IX</td>
<td>22.22%</td>
<td>45.12%</td>
<td>76.35%</td>
<td>83.00%</td>
<td>99.00%</td>
</tr>
<tr>
<td>Group X</td>
<td>0.63%</td>
<td>6.33%</td>
<td>27.42%</td>
<td>42.74%</td>
<td>55.56%</td>
</tr>
<tr>
<td>Group XI</td>
<td>11.77%</td>
<td>28.40%</td>
<td>55.56%</td>
<td>65.28%</td>
<td>76.00%</td>
</tr>
<tr>
<td>Group XII</td>
<td>60.36%</td>
<td>79.56%</td>
<td>97.20%</td>
<td>99.40%</td>
<td>99.40%</td>
</tr>
<tr>
<td>Group XIII</td>
<td>39.56%</td>
<td>70.32%</td>
<td>96.20%</td>
<td>99.70%</td>
<td>99.70%</td>
</tr>
<tr>
<td>Average</td>
<td>40.34%</td>
<td>57.90%</td>
<td>78.59%</td>
<td>85.47%</td>
<td>99.00%</td>
</tr>
</tbody>
</table>

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
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
- Microsoft: 12%
- OpenSSL: 86%

Classification (10-99 keys with same subject and issue date)
- Microsoft: 10.18%
- OpenSSL: 82.84%
- Botan: 5.61%
- WolfSSL: 1.09%
- Nettle: 1.09%
- OpenJDK: 1.09%
- Cryptlib: 1.09%

[Image of pie charts and percentages]
Sanity check: keys which *cannot* be from OpenSSL

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

<table>
<thead>
<tr>
<th>Dataset</th>
<th>!OpenSSL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cert. Transparency</td>
<td>11.80%</td>
</tr>
<tr>
<td>PGP keyset</td>
<td>47.35%</td>
</tr>
<tr>
<td>TLS IPv4</td>
<td>18.91%</td>
</tr>
<tr>
<td>Let’s Encrypt</td>
<td>1.83%</td>
</tr>
</tbody>
</table>

OpenSSL rare in PGP
Leaves ~81% for OpenSSL
OpenSSL is default client
Evolution of TLS keys in time

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

![Graph showing evolution of TLS keys over time with different groups and dates.]

### Unique keys only – 2–9 keys batch

**Count:**
- 84
- 96
- 106
- 94
- 97
- 95
- 98
- 104
- 2056
- 114
- 109
- 111
- 138

**Date:**
- 2016/201
- 2015/03/01
- 2015/04/02
- 2015/06/16
- 2015/07/02
- 2015/08/02
- 2015/10/01
- 2015/11/06
- 2015/12/02
- 2016/04/04
- 2016/06/03

**Known Groups:**
- Group I
- Group II
- Group III
- Group IV
- Group V
- Group VI
- Group VII
- Group VIII
- Group IX
- Group X
- Group XI
- Group XII

**Unknown Groups:**
- Botan
- Microsoft
- OpenSSL
- Nettle

![Diagram showing the distribution of groups and dates.]

**Legend:**
- Green: Group I
- Purple: Group II
- Red: Group III
- Orange: Group IV
- Yellow: Group V
- Pink: Group VI
- Blue: Group VII
- Light Green: Group VIII
- Pink: Group IX
- Light Blue: Group X
- Yellow: Group XI
- Pink: Group XII
- Unknown: Blue

**OpenSSL:**
- Nettle
- Botan
Audit: What Amazon EC2 uses to generate RSA keys?

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

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.

<table>
<thead>
<tr>
<th>Group VI</th>
<th>Group XIII</th>
<th>Group XI</th>
<th>Group III</th>
<th>Group XII</th>
<th>Group VII</th>
<th>Group I</th>
<th>Group II</th>
<th>Group IV</th>
<th>Group V</th>
<th>Group VIII</th>
<th>Group IX</th>
<th>Group X</th>
</tr>
</thead>
<tbody>
<tr>
<td>98.86 %</td>
<td>0.80 %</td>
<td>0.23 %</td>
<td>0.10 %</td>
<td>0.00 %</td>
<td>0.00 %</td>
<td>not possible</td>
<td>not possible</td>
<td>not possible</td>
<td>not possible</td>
<td>not possible</td>
<td>not possible</td>
<td>not possible</td>
</tr>
</tbody>
</table>

Verisign CA

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 99% probability that correct source is identified within the first three most probable groups.

<table>
<thead>
<tr>
<th>Group XIII</th>
<th>Group XI</th>
<th>Group III</th>
<th>Group XII</th>
<th>Group VII</th>
<th>Group I</th>
<th>Group II</th>
<th>Group IV</th>
<th>Group V</th>
<th>Group VI</th>
<th>Group VIII</th>
<th>Group IX</th>
<th>Group X</th>
</tr>
</thead>
<tbody>
<tr>
<td>74.10 %</td>
<td>18.42 %</td>
<td>7.46 %</td>
<td>0.02 %</td>
<td>0.00 %</td>
<td>not possible</td>
<td>not possible</td>
<td>not possible</td>
<td>not possible</td>
<td>not possible</td>
<td>not possible</td>
<td>not possible</td>
<td>not possible</td>
</tr>
</tbody>
</table>
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

---

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

<table>
<thead>
<tr>
<th>Group VII</th>
<th>Group II</th>
<th>Group X</th>
<th>Group VIII</th>
<th>Group IX</th>
<th>Group XIX</th>
<th>NXP J2A080, NXP J2A081, NXP J3A081, NXP JCOP 41 v2.2.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>99.40 %</td>
<td>0.56 %</td>
<td>0.03 %</td>
<td>0.00 %</td>
<td>0.00 %</td>
<td>not possible</td>
<td>FIPS 140-2 level 3 crypto smartcards manufactured by NXP</td>
</tr>
</tbody>
</table>
Forensics: source library/device for factorizable keys

- 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...

Private keys are available – more accurate classification possible

Identification of responsible source allows to contact and eventually fix
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

Source profiles not equal, but similar enough => Accuracy significantly decreased
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
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

Library: mbedTLS 2.2.1

Library: Cryptix JCE 20050328
24 different software libraries

8 classification groups
Difference in libraries based on private keys and factorization

- 24 different software libraries
- 19 classification groups

Libraries:
- OpenSSL 1.0.2g
- GPG Libgcrypt 1.6.5
- PGP SDK 4
- Bouncy Castle 1.54
- cryptlib 3.4.3
- Crypto++ 5.6.3
- WolfSSL 3.9.0
- Botan 1.11.29
- LibTomCrypt 1.17
- Apple corecrypto 337 FIPS
- Apple corecrypto 337
- Apple Libgcrypt 1.6.5 FIPS
- OpenSSL FIPS 2.0.12
- Nettle 3.2
- Microsoft CryptoAPI
- Microsoft CNG
- Microsoft .NET
- GNU Crypto 2.0.1
- PGP SDK 4 FIPS
- Cryptix JCE 20050328
- FlexiProvider 1.7p7
- mbedTLS 2.2.1
- Bouncy Castle 1.53
- SunRsaSign OpenJDK 1.8
ADDING MORE SOURCES
Utimaco Se50 LAN HSM
Safenet Luna SA-1700 LAN

Library: mbedTLS 2.2.1
Library: Cryptix JCE 20050328
Library: SunRsaSign OpenJDK 1.8
Library: Bouncy Castle 1.53
Library: Botan 1.11.29
Library: cryptlib 3.4.3
Library: LibTomCrypt 1.17
Library: Nettle 3.2

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
What changed between BC 1.53 and 1.54?

```java
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
RandomData.generate()
Simple power analysis of RSA keypair generation

<table>
<thead>
<tr>
<th></th>
<th>Key generation</th>
<th>Decryption</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Fixed</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Variable</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Prime P generation start end
Prime Q generation start end
Incremental search for primes

How to find a prime number:

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…

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
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

BACKUP SLIDES
Infineon JTOP 80K M8.4

Infineon JTOP 80K – overlapping serial test 16-bit

Observed frequency of pattern as multiple of expected frequency:

- Pattern (1) 0.96
- Pattern (2) 1.02
- Pattern (3)
  - xy(x0 XOR 01), xy(x0 XOR 80)
- Pattern (4)
  - xyxy, xxxx

16-bit pattern (hexadecimal):

0000 2492 4924 6DB6 9248 B6DA DB6C FFFF
Oberthur Cosmo Dual 72K

![Graph showing the overlapping serial test 16-bit pattern distribution](image-url)