Random Data and Key Generation Evaluation of Some Commercial Tokens and Smart Cards

Ahmad Boorghany, Siavash Bayat Sarmadi, Parnian Yousefi, Pouneh Gorji, Rasool Jalili

Data & Network Security Lab (DNSL)
Computer Engineering Dept., Sharif Univ. of Technology

ISCISC’14
September 3, 2014
Outline

- Background
  - Randomness Failures in Cryptography
  - Common Prime Attack on RSA Keys
- Our Experiments
  - Idea
  - Methodology and Tools
- Evaluation Results
  - Randomness Evaluation
  - RSA Key Evaluation
- Conclusion and Future Works
Background
Randomness in Cryptography

- **Crucial** for CPA security [GM84]
  - CPA is a **weak security notion** (respecting CCA or CCA2)

- **Some stories:**
  - Debian’s Openssl Bug [YRS+09]
    - RNG output domain $< 65536$
    - For **two years**: 2006~2008
    - 2012: still **57000 vulnerable** HTTPS/SSH servers on the Internet [HDWH12]
  - Android’s RNG Bug [MMS13]
    - Successful thefts from **Bitcoin users** [But13]
Common Prime Attack on RSA Keys

- RSA Cryptosystem:
  - Depends on the factoring problem
  - \( p \) and \( q \) are large random primes
  - 512 bits each in RSA-1024

- Common Prime Factor?
  - If the RNG is good, probability < \( 2^{-500} \)
  - If \( N_1 = p \times q_1 \) and \( N_2 = p \times q_2 \):
    - \( p = \text{GCD}(N_1, N_2) \) → Done efficiently
    - \( q_1 = N_1/p \), \( q_2 = N_2/p \)
Common Prime Attack on RSA Keys

- Heninger et al. [HDWH12] in USENIX Sec 2012
- Crawled the Internet looking for common factors
  - Live hosts: 23,044,976
  - Vulnerable ones: 66,540 (≈ 3 in 1000)

- Almost all failures: on embedded/constrained devices
  - Lack of good entropy sources
Common Prime Attack on RSA Keys

- Bernstein et al. [BCC+13] in Asiacrypt 2013
- Tested Taiwanese DB of certificates
  - Personal smart cards
  - More than 3,000,000 RSA public keys
Common Prime Attack on RSA Keys

- 105 moduli factored easily by pair-wise GCD
- The most popular modulus (46 occurrences):

```plaintext
c00000000000000000000000000000000
000000000000000000000000000000000
000000000000000000000000000000000
000000000000000000000000000002f9
```

- Why? Maybe *randomness failures*.
Our Experiments
The Idea

- Evaluate hardware security modules in the market
  - Tokens
  - Smart Cards

- So, what to do?
  - Generate RSA Keys, and compute pair-wise GCDs
  - Generate random streams, and evaluate them in advance
How to talk to these devices?

- PKCS#11
  - C_GenerateRandom
  - C_GenerateKeyPair

- Java Card:
  - Command 1
  - Import JavaCard. ...
  - public class TestCard {
    ...

Methodology

- Targeted Tokens and Smart Cards:
  - Token 1: PKCS#11
  - Token 2: PKCS#11
  - Token 3: PKCS#11
  - Token 4: PKCS#11
  - Token 5: PKCS#11
  - Smart Card 1: PKCS#11
  - Smart Card 2: Java Card
  - Smart Card 3: Java Card

- Sorry, but no names 😞
Methodology

- For each hardware:
  - 10,000,000-bit stream generated
  - Its randomness evaluated using NIST’s Statistical Test Suit (STS)
  - 161 instances from 15 distinct tests
    - Frequency Test
    - Runs Test
    - Serial Test
    - Overlapping/Non-overlapping Template Test
    - etc.
Methodology

- For each hardware:
  - 200 RSA key-pairs generated
    - 1024-bit and 2048-bit
  - Pair-wise GCDs computed:
    - With each other
    - With the database of MOCCA
      - 25000 certificates
    - With the database of Heninger et al.’s crawling
      - Using factorable.net
Evaluation Results
Randomness Evaluation

- Simple frequency diagram

- Token 1
- Token 2
- Token 3
- Token 4
- Token 5
- Smart Card 1
- Smart Card 2
- Smart Card 3
- Blum-Blum-Shub
Randomness Evaluation – STS Results

Background

Our Experiments

Eval. Results

Token 1

0.95 Nonoverlap. Templ.

Token 2

Token 3

0.95 Nonoverlap. Templ.

Token 4

Token 5

0.95 Serial

Smart Card 1

0.27 Frequency

0.27 Cumul. Sums 1

0.85 Runs

0.29 Cumul. Sums 2

Smart Card 2

0.94 Nonoverlap. Templ.

0.95 Nonoverlap. Templ.

0.95 Nonoverlap. Templ.

Smart Card 3
### RSA Key Evaluation

<table>
<thead>
<tr>
<th>Hardware Module</th>
<th>Type</th>
<th>1024-bit Generation Time (s)</th>
<th>2048-bit Generation Time (s)</th>
<th># of Factored Moduli</th>
</tr>
</thead>
<tbody>
<tr>
<td>Token 1</td>
<td>PKCS#11</td>
<td>9.7</td>
<td>42.0</td>
<td>0</td>
</tr>
<tr>
<td>Token 2</td>
<td>PKCS#11</td>
<td>19.3</td>
<td>153</td>
<td>0</td>
</tr>
<tr>
<td>Token 3</td>
<td>PKCS#11</td>
<td>3.9</td>
<td>15.2</td>
<td>0</td>
</tr>
<tr>
<td>Token 4</td>
<td>PKCS#11</td>
<td>14.8</td>
<td>148</td>
<td>0</td>
</tr>
<tr>
<td>Token 5</td>
<td>PKCS#11</td>
<td>19.0</td>
<td>103</td>
<td>11</td>
</tr>
<tr>
<td>Smart Card 1</td>
<td>PKCS#11</td>
<td>18.1</td>
<td>57.7</td>
<td>0</td>
</tr>
<tr>
<td>Smart Card 2</td>
<td>Java Card</td>
<td>1.98</td>
<td>12.7</td>
<td>0</td>
</tr>
<tr>
<td>Smart Card 3</td>
<td>Java Card</td>
<td>2.0</td>
<td>13.4</td>
<td>0</td>
</tr>
</tbody>
</table>

- **Token 5**: very small prime factors: 3, 5, 7, ...
Conclusion and Future Works

- Evaluation is a must!
- Better evaluation methods required
- Note: only simple vulnerabilities can be found by statistical testing
- Other schemes: ECDSA, etc.
Thanks for your attention

Questions?
References


References
