

---

# Generating Random and Pseudorandom Sequences in Mobile Devices

---

**Jan Krhovjak, Vashek Matyas, Jiri Zizkovsky**

Faculty of Informatics  
Masaryk University, Brno  
Czech Republic

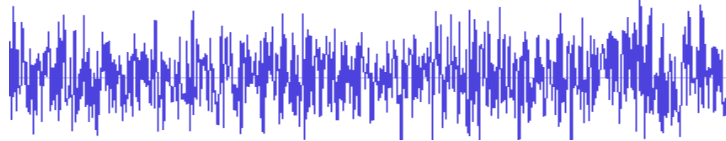
# Motivation and common problems

- Random data – bits, numbers, sequences
  - Uniformly distributed and unpredictable random variable
- Random data in cryptography – why so important?
  - Cryptographic keys, padding values, nonces, etc.
  - Statistical quality and unpredictability is critical
- Famous failures of random data generators...
  - Early version of Netscape SSL (1996)
  - Sun's MIDP ref. implementation (2005)
  - OpenSSL FIPS Object module (2007)
  - Java 2 ME (2007)
  - Debian OpenSSL (2008)
    - Certification authorities – hundreds of thousands of revoked keys

# Basics of random data generation

- Generating truly random data

- Based on nondeterministic physical phenomena
  - Radioactive decay, thermal (white) noise, etc.



- In deterministic environments hard and slow

- Generating pseudorandom data

- Based on deterministic algorithms
  - Short input (seed) – truly random data
  - Output – pseudorandom data, computationally indistinguishable from truly random data

- Mobile environments – open issues...

# More sophisticated approach

- Mobile environments are different & not secure
  - Temporal theft => inner state compromise
  - Forward/backward security must be assured
    - All previous/future states and outputs well secured
    - One-way functions & periodical refreshing
- Hybrid semi-deterministic approach
  - Usage of available truly random data
    - Periodically improves inner state or inner pool
  - Deterministic only between two reseedsings



# Qualitative requirements on random data

- Statistical properties & unpredictability
  - Stat. defects and correlations => predictability
    - Typically induced by hardware (during the sampling)
  - Ensuring good stat. properties is critical
- Digital post-processing
  - Reduce bias and dependencies of adjacent bits
  - Pseudorandom number generators
    - Spreading simple stat. defects into longer sequence
  - Randomness extractors
    - Condense input randomness to most compact form
- Statistical testing
  - Manual testing => detection of design flaws
  - On-the-fly testing => detect breaking or influencing

# Selected mobile devices & initial thoughts

- Programmable mobile devices
  - Smartphone Nokia N73 (Symbian OS, JavaME)
  - PDA phones E-TEN M700 & X500 (WM5/WM6)



- Some low-level sources have no sufficient precision (API restrictions) or have a slow refresh frequency
  - Battery level and signal strength (only ten-value scale)
  - GPS position (only one measurement per second)
- The most promising sources of randomness are microphone and digital camera noise

# Quantitative requirements on random data

- Basic measure for randomness is called *uncertainty* or *entropy* (Shannon or average-case)

$$H_1(X) = - \sum_{x \in X} P_X(x) \log P_X(x)$$

- Sample  $x$  is drawn from random distribution  $X$  with probability  $P_X(x)$
- Logarithm base corresponds to units (2 => bits)

- Better measure is *min-entropy* (worst-case)

$$H_\infty(X) = \min_{x \in X} (-\log P_X(x)) = -\log(\max_{x \in X} P_X(x))$$

- Always “less than” or “equal to” Shannon entropy

- How many random bits are extractable per time unit?

# Entropy estimation – attacker-aware methodology

- Obvious problem
  - Sampled data can be (and typically are) non-uniform
    - Attacker can entirely change statistical distribution of sampled data
  - This typically implies unrealistic (min-)entropy estimation
- Ideal solution
  - Remove all stat. defects & dependences
  - Entropy estimation and extraction (or vice versa)
- My *attacker-aware* methodology
  - Data captured with as little noise as possible
  - Computing of histograms, probabilities, entropy estimation
  - Statistical testing (NIST, auto- and cross- correlation)
    - Success: no impact to entropy estimation
    - Failure: simple “entropy extraction” and/or new lower estimation

# Analysis of microphone & camera input

- Microphone voice input

- Quiet room



- Nokia N73

- Min-entropy: **1.9** bits;
- Shannon ent.: **2.9** bits
- Upper bound of ent.: **4.2** bits

- E-TEN M700

- Min-entropy: **9.1** bits
- Shannon ent.: **10.1** bits
- Upper bound of ent.: **11.8** bits

- E-TEN X500

- Min-entropy: **8.5** bits
- Shannon ent.: **9.4** bits
- Upper bound of ent.: **10.6** bits

- Autocorrelation

- Taking only every 2<sup>nd</sup>, 3<sup>rd</sup>, 4<sup>th</sup> value decreased correlation
- Lower entropy – at least 5x

- Digital camera input

- View finding
- Closed camera cover
- Temperatures 5–45 °C
- Entropy estimation for single R/G/B color pixel follows



- Nokia N73

- Min-entropy: **3.2, 3.3, 3.9** bits
- Shannon entropy: **3.9, 4.0, 3.8** bits

- E-TEN M700

- Min-entropy: **3.0, 3.0, 3.7** bits
- Shannon entropy: **4.5, 4.3, 5.4** bits

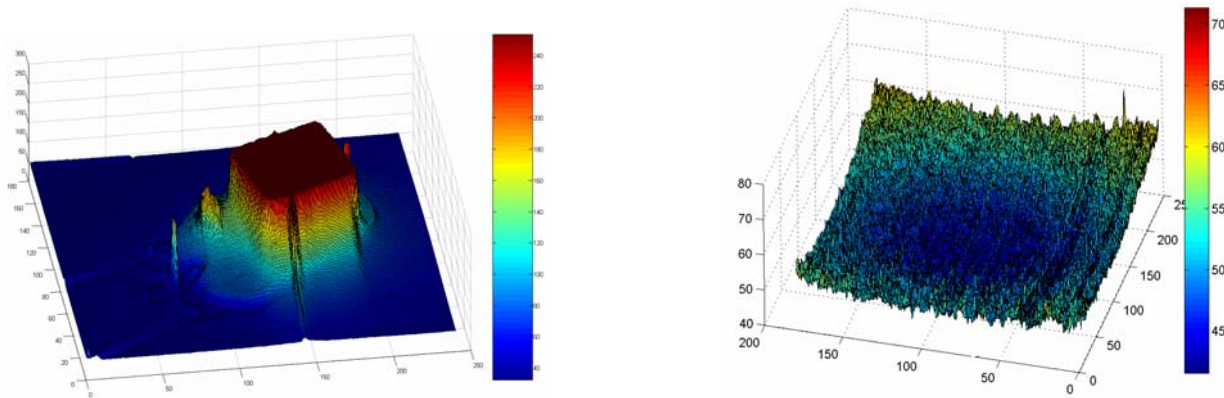
- E-TEN X500

- Min-entropy: **1.3, 2.2, 0.8** bits
- Shannon entropy: **3.1, 3.1, 2.5** bits

- Testing independency of pixels in image (& between images)

# Randomness sources – overall results

- Microphone & camera input have great potential
  - Big throughput and inherently present internal noise
  - Estimated entropy for **microphone** input [bits per second]
    - 16bit PCM samples at 8000 Hz, min-entropy between 1.9 and 9.1  
=> **thousands** bits **per one** second
  - Estimated entropy for **camera** input [bits per second per picture]
    - 3 different 8bit colors, min-entropy between 0.8 and 3.9, resolutions 240×180 or 240×320 => more than **100 000s** bits **in one** picture





# Design of prototype

- Target platform selection (Symbian OS 9.x)
  - Nokia N73 used for practical experiments and performance tests
- Speed of basic crypto. primitives on Nokia N73
  - SHA-1: **1799** KB/s; SHA-256: **2667** KB/s
  - Non-optimized AES: **75** KB/s; Native AES: **345** KB/s
- Integration of two pseudorandom generators
  - Hybrid generators ANSI X9.17/31 and Fortuna
    - GUI application vs. client-server approach
  - Random data from microphone, camera, keyboard
    - Estimated entropy in data always radically exceeds amount of required entropy (or pool sizes)

# ANSI X9.31 (former X9.17) PRNG

- GUI application – no embedded continuous pooling
- Disposable 320bit “pool” is implemented as two SHA-1 contexts and is used only during initialization
  - 5 samples from the camera viewfinder, 20 audio samples from the microphone, and the timing of each keystroke
  - 128 bits is used as key  $K$ , 128 bits as seed  $V$ , 64 bits fill the second part of DT vector
- Generator itself uses a 128bit block cipher AES
  - Basic loop ( $E()$  is AES encryption) is:
    - $I = E_K(DT)$  [encrypted date/time vector]
    - $R = E_K(I \oplus V)$  [128bit output]
    - $V = E_K(R \oplus I)$  [updated seed]
  - Improved version uses SHA-256
- Good for appl. that require random data only at the start

# Fortuna PRNG

- Client-Server Framework
- Generator: AES-256 in CTR mode (128bit counter)
  - Periodical re-keying (forward/backward security)
- Pooling: based on SHA-2
  - 32 pools for collecting entropy (cyclically)
    - Thousands bytes of data is always added to each pool
  - P0 is used every reseed, P1 is used every second reseed, P2 is used every fourth reseed, etc.
- Seed file for storage of high-entropy data
  - Generator will have entropy even after rebooting
  - /Private folder of application, 64 bytes of data

# Performance & power consumption

- Performance (N73)
  - ANSI X9.31: 326 KB/s
  - Fortuna: 848 KB/s
- Compromise of security and usability (N73)
  - Battery discharging (Fortuna)
    - Continuous sampling => a few hours (~3 h)
    - One sample per 1 minute => almost 4 days
    - One sample per 5 minutes => almost 7.5 days
    - No sampling => almost 10.5 days
  - High energy requirements during continuous sampling
    - Digital camera is reserved only for Fortuna

# Conclusion

- Identification and analysis of available **randomness sources** in **mobile devices**
  - Microphone & camera input have great potential
- Analysis and secure integration of selected **digital post-processing** methods
  - Integration issues of randomness extractors and pseudorandom data generators
  - Implementation of ANSI X9.17/31 and Fortuna PRNG
- A **practical approach** to random data **generation** for the mobile environment
  - Clear path to application developers of many security-critical applications for the mobile environment