Generating Random and Pseudorandom Sequences in Mobile Devices

Jan Krhovjak, Vashek Matyas, and Jiri Zizkovsky

Faculty of Informatics, Masaryk University, Brno, Czech Republic
{xkrhovj,matyas,xzizkovs}@fi.muni.cz

Abstract. In our paper we study practical aspects of random and pseudorandom number generation in mobile environments. We examine and analyze several sources of randomness available in current mobile phones and other mobile devices at the application level. We identify good physical sources of randomness that are capable of generating data with high entropy in reasonable time and we investigate some relevant aspects (such as security, energy requirements, performance) of integrating selected pseudorandom number generators in the Symbian OS environment. The main contribution of this paper is the identification and analysis of randomness sources in mobile devices and a practical proposal for their post-processing, including a prototype implementation.

Keywords: mobile device, random sequence, source of randomness.

1 Introduction

Unpredictable cryptographic keys, padding values or per-message secrets are critical to securing communication by modern cryptographic techniques. Their generation typically requires an unpredictable physical source of random numbers and secure mechanism for their digital postprocessing.

Most common generation techniques involve truly random and pseudorandom number generators. The former are typically based on a nondeterministic physical phenomena (e.g., radioactive decay or thermal noise), while the latter are only deterministic algorithms where all randomness of the output is dependent on the randomness of one or several inputs (often called seed). Generation of pseudorandom data is typically (in most environments) faster and truly random data is used in this process only as the initial input.

Our paper deals with issues related to the generation of truly random and pseudorandom data (i.e., bits, numbers, and sequences) in mobile computing environments. Mobile devices are different from general purpose computers, and are now commonly used for security-critical applications like mobile banking, secure voice and data communication, etc. However, current mobile platforms do not provide a suitable built-in entropy source for seeding a pseudorandom generator. We consider the camera and the microphone noise as the most promising candidates for good sources of randomness. They provide a considerable amount

A.U. Schmidt and S. Lian (Eds.): MobiSec 2009, LNICST 17, pp. 122–133, 2009.

of randomness (entropy) in a given time period and can be easily (with minimal postprocessing) used as a reliable true random number generator (TRNG).

In order to use random data for cryptographical purposes we need to obtain a sequence of unpredictable random bits (i.e., with a sufficient amount of entropy) distributed according to the uniform distribution. Since almost all external physical sources of randomness can be observed or influenced (gamma rays, temperature, etc.) resulting in non-random or even constant values, we postprocess resulting data with a pseudorandom number generator (PRNG)[1].

We implemented ANSI X9.31 (former X9.17) and Fortuna PRNGs for mobile devices with Symbian OS 9.x. Both these PRNGs are based on classical cryptographic primitives (e.g., AES) and use available randomness sources during the whole generation process. This property implies robust and secure design with capability of recovering from internal state compromise. Both our implementations are a proof of concept that demonstrates the feasibility of generating high-quality pseudorandom data in mobile phones at the application level.

A detailed discussion regarding identification, testing, evaluation of physical sources of randomness, and entropy estimation can be found in [6]. Some issues described in this paper were discussed at the conceptual level also in [7].

2 Requirements on Random Data

Let us begin with a basic description of requirements on random data for cryptographic purposes. We can distinguish between qualitative and quantitative requirements for random data. The former cover good statistical properties of generated random data and unpredictability of such data, while the latter deal with measuring of randomness and also cover demands of used cryptographic techniques and the performance issues of cryptographic generators.

2.1 Qualitative Requirements

Random data generated directly from a randomness source often contains statistical defects and dependencies causing parts of the random data to be easily predictable. These statistical defects are typically inducted by hardware generators during the sampling of the analogue randomness source or by influencing the sampled physical randomness source (e.g., by an active adversary).

The first step towards unpredictability lies in ensuring (at least to certain level) good statistical properties of generated random data. This can be partially solved by using digital postprocessing and/or statistical testing. Digital postprocessing is the deterministic procedure capable to reduce statistical relations and dependencies (including bias and correlation of adjacent bits). Statistical testing allows to (manually) detect some design flaws of a generator or to (automatically) avoid breaking or influencing the generator during its lifecycle.

[1] Note that an ideal PRNG for cryptographic purposes produces sequences that is unpredictable and computationally indistinguishable from the truly random data.

Typical techniques of digital postprocessing involve use of deterministic pseudorandom number generators or randomness extractors. The former serve only for spreading simple statistical defects into a longer sequence of bits. The latter allow to condense available input randomness to the most compact form that has uniformly distributed bits without statistical defects. Pseudorandom number generators as well as randomness extractors can be based on cryptographic primitives (e.g., hash functions) or simple mathematical functions. However, none of deterministic digital postprocessing methods can be used for improving initial randomness from the physical source.

The advantage of randomness extractors lies in good theoretical and mathematical background – more sophisticated randomness extractors can even provide some provable guarantees of the quality (resulting distribution) and quantity (in terms of extracted so-called min-entropy) of its output. We can distinguish deterministic or non-deterministic randomness extractors. The former work only on limited classes of randomness sources. The latter do not have this restriction, but they need an additional truly random input. Unfortunately, probability distributions of randomness sources must be in both cases precisely defined, otherwise it is not possible to construct appropriate and well working randomness extractor. In addition to that, the usage of an randomness extractor makes the generation of random numbers even slower. More details can be found in [11].

There are several commonly used statistical test suites or batteries (e.g., CRYPT-X, DIEHARD, and NIST) for verification of the statistical quality of random data by detecting deviations from true randomness (for details see [10]). However, no finite set of statistical tests can be viewed as complete and the results of statistical testing must be always interpreted with some care and caution to avoid wrong conclusions about a specific randomness source or generator.

2.2 Quantitative Requirements

A precise comparison of several sources of randomness requires also some measure of randomness. Basic measure is in information theory often called uncertainty or entropy and referred as Shannon entropy or alternatively information entropy. The well-known Shannon formula computes the entropy according to all observed probabilities of values in the probability distribution. This results only in average case entropy that is inappropriate for cryptography purposes. To (partially) cope with this situation the worst case min-entropy measure is often used. Min-entropy formula computes the entropy according to the most probable value in probability distribution.

Unfortunately, both entropy formulas have one serious drawback: they work only for exactly defined (and fixed) randomness distribution. In our case we cannot make any assumptions about distributions dynamically formed by used sources of randomness (e.g., they can be under ongoing attack) and this can always imply biased results in terms of entropy. This is a fundamental problem that can be seen also in the theory of randomness extractors. To partially cope with this problem, we perform all our experiments and entropy estimations in the worst conditions – i.e., under simulated attack on physical randomness source.

3 Randomness in Mobile Devices

Mobile devices are considerably different from general purpose computers and this also influences the process of generating random and pseudorandom data. The possibility to change the environment where a mobile device operates is definitely a great advantage given by the mobility nature of the device. The existence of several embedded input devices such as microphone, radio receiver, video camera, or touchable display is another advantage[2].

On the other hand, mobility and small physical size of devices bring also a higher possibility of theft or (temporary) loss with a potential compromise of the generator state. The important assumption of secure generator design is thus the impossibility of deducing the (previous/future) inner state of the generator and fast recovery of its entropy level (after a time-limited compromise).

Well-designed and robust generator must always have a sufficient amount of entropy – shortly after turning the device on, after letting the device out of sight, and after an intensive generation of random data. This non-trivial task may require employment of energetically costly sources of randomness (e.g., video camera) and/or the user contribution. Utilization of these sources may also be required to assure higher security – e.g., for mobile banking purposes.

3.1 Sources of Randomness in Mobile Devices

This part of the paper deals with practical experiments performed on two smartphones Nokia N73 with the Symbian OS and two similar PDA phones E-Ten X500 and E-Ten M700 with the Windows Mobile OS. The goal of our experiments was to assess the quality of selected sources of randomness in these mobile devices and to estimate the amount of randomness (entropy) in these sources. We used two identical Nokia N73 devices since we wanted to verify the correctness of our results or to detect unexpected behavior of the smartphone – in a case when one device suffers, e.g., by some manufacturing defect. Some issues described in this section were discussed and described in more detail in [6].

Due to the API restrictions in the Symbian OS, we were forced to drop sources of randomness like the battery level, signal strength or GPS position as measurements over these sources do not provide output (at the API level) with a sufficient precision (e.g., battery and signal values are available in the form of an integer between 0 and 10) or frequency (e.g., external GPS provides only one measurement per second).

On the contrary, microphone and digital camera perform a high-rate sampling of physical sources, yielding high volumes of data. Since we can never guarantee the quality of a physical source, our analysis is concentrated on the microphone and the camera noise that arises, e.g., in the CCD/CMOS chip or A/D converter, and is always present in the output data.

[2] We are aware that in general purpose computers use of audio/video data for generating random numbers is not a novel idea (see, e.g., [2]), but mobile devices with embedded cameras and microphones have a clear advantage with respect to fitness and practical applications of such techniques.

Microphone input: We wanted to evaluate all microphones in the worst possible conditions, therefore the first idea of our test settings involved recording of: some music sample, audio feedback, noise in an extremely quiet room. After several basic experiments with the built-in notebook microphone we saw that an audio feedback (i.e., sound loop between an audio input and an audio output) brings even more entropy than the music sample recording. Since the recorded noise is also present in music or other audio samples, the worst conditions for the microphone input arise from recording of noise in an extremely quiet environment (which was in our case a closed quiet room in the night).

We analyzed several microphones that have obviously different characteristic, e.g., due to different solidity of membrane or other manufacturing differences. 204 800 captured noise samples[3] were used to form a histogram of values. Furthermore, we used min-entropy (worst case) formula for quantitative analysis and for estimation of entropy in the generated data. Our min-entropy estimations, with the assumption of independency within the samples, are: 0.5 bits of entropy per sample for Nokia N73 hands-free microphone; 0.016 bits of entropy per sample for E-Ten M700 embedded microphone; 0.023 bits of entropy per sample for the E-Ten X500 embedded microphone.

As the next step we tested statistical quality and independency of captured noise samples. We used the Fast Fourier Transform for analyzing the basic frequency components present in the noise – ideal noise is expected to have all frequencies uniformly present. Several harmonic frequencies (narrow peaks in spectrogram) were revealed in spectrum of E-Ten devices and the best result (i.e., smoothest frequency spectrum) belongs to the Nokia N73 hands-free microphone with only few harmonic frequencies (see Figure 1).

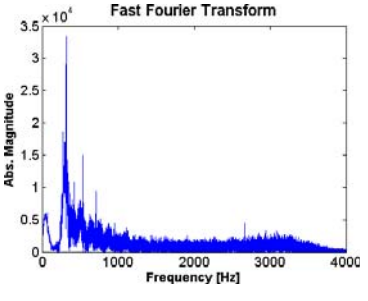

Fig. 1. Microphone frequency spectrum

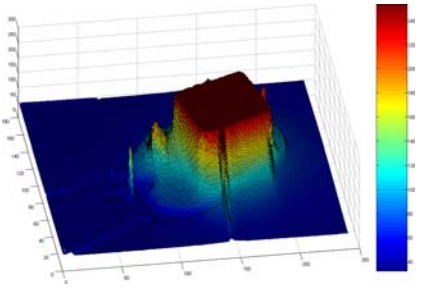

Fig. 2. Overexposure by a halogen lamp

We performed also several correlation tests that found a correlation in the recorded samples of noise. This correlation decreased as we took only every second/third value from our samples. Sequence created from every fourth value was without any statistically significant correlations. We therefore recommended

[3] We set 16-bit pulse coded modulation (a signed PCM) at the frequency 8000 Hz for sampling a sound wave.

to lower the estimated entropy at least 5 times with respect to the amount calculated for each sample value.

Camera input: Digital cameras for mobile devices can be based on several different silicon optical sensors, typically CCD or CMOS. All of them use an array of semiconductor photo-sensors to transfer an accumulated electric charge to a voltage. Low-quality sensors are typically used in mobile devices, and these sensors have a higher noise level than sensors used in standard digital cameras.

The worst conditions for these sensors involve recording of a static image (white or black background). The input source is often available even when the camera cover is closed or covered. This is both convenient and useful – it serves as a defense against an active attacker that illuminates the sensors and forces them to produce biased values. Illumination of the Nokia N73 CCD sensor by a halogen lamp is depicted in Figure 2 – in this case the central area of a sensor is forced to produce maximum values (225) and all noise is effectively removed.

Several components of the noise arising from optical sensors can be distinguished (shot noise, read-out noise, etc.), but we are interested in the overall noise. It is a well-known fact that the predominant component of this noise is the thermal noise and its actual level may depend on physical conditions – namely the temperature (for details see [1]). Therefore, we performed all practical experiments with the camera set at two temperature levels: 5 °C and 45 °C. We detected an inside decrease of the noise towards lower temperature, but the noise is still significantly present to provide enough entropy.

We captured all the test data from the viewfinder[4] rather than from a high-resolution picture. We are aware that the viewfinder image is downsampled from a full resolution of the optical sensor, but our primary intent was to overcome other software post-processing techniques (noise reduction, compression, etc.). However, performed experiments confirmed the presence of low level post-processing (details are not provided by the manufactures, similarly as for the downsampling algorithm) as, for example, correcting of light intensity towards the border of lens or automatic ISO level correction. All these proprietary techniques make the analysis and entropy estimation much harder.

At least 2000 noise samples (at temperatures 5–8 °C) have been always used to create a histogram of values for a particular color pixel. Our analysis confirmed that all three tested devices have clearly different camera chips and noise patterns. Our min-entropy estimates, with the assumption of independency within the pixels and frames, are: 3.2/3.3/3.9 bits of entropy per R/G/B color pixel for Nokia N73; 3.0/2.9/3.7 bits of entropy per R/G/B color pixel for E-Ten M700; 1.3/2.2/0.8 bits of entropy per R/G/B color pixel for E-Ten X500.

For statistical testing of the quality of noise samples we used the NIST test battery, auto- and cross-correlation functions in Matlab, and Fast Fourier Transformation. Our experiments revealed no statistically significant dependencies (significance level was set to 0.01) between neighboring pixels, rows, or even successive pixels in frames.

[4] The used resolution is at least 180×240 pixels and recording speed is within 10–15 frames per second.

Full description of all our experiments with Nokia N73 and E-TEN X500/M700 devices and other related technical details can be found in [6].

3.2 Secure Pseudorandom Numbers

The advantage of pseudorandom number generators lies in secure masking (smoothing) of statistical deviations caused by temporarily corrupted or influenced sources of randomness. If these generators behave correctly (in a secure computing environment), the resulting sequences have their statistical quality assured and the only critical point remains resistance to cryptanalysis.

The majority of multi-user and mobile computing environments cannot be considered as a secure computing environment (due to malicious admins/users, malware, possibility of device temporary loss). A careful generator design must thus take into account both forward and backward security after a compromise of its internal secret state. Forward/backward security means that an attacker with the knowledge of internal state of a generator is not able to learn anything about previous/future states and outputs of the generator. Forward security can be guaranteed by using one-way functions and backward security requires periodical refreshing of the internal state with additional truly random data.

The consequence of periodical refreshing is utilizing all available random samples – with the possibility of their gathering and accumulation (so-called pooling) even in the time when no generation is required. Such hybrid generators behave deterministically only between two successive reseedings.

We selected ANSI X9.31 and Fortuna pseudorandom number generators for our reference implementation – their basic properties are described below.

ANSI X9.31 PRNG: The ANSI X9.31 (former X9.17) PRNG uses in our setting only one cryptographic primitive, a block cipher. The NIST specification [8] suggests two implementations using the algorithm 3DES or AES. The main difference between these two block ciphers is the supported length of a block (3DES uses 64-bit blocks, AES operates on 128-bit blocks).

Let K be a 128-bit AES secret key, which is reserved only for the generation of pseudorandom numbers. E_K denote AES encryption (ECB) under the key K, V is a 128-bit seed value, and DT is a 128-bit date/time vector. I is an intermediate value, and the symbol $\oplus$ is the exclusive-or (XOR) operator.

Then a pseudorandom output R is in each iteration generated as follows:

$$I = E_K(DT), \tag{1}$$

$$R = E_K(I \oplus V), \tag{2}$$

$$V = E_K(R \oplus I). \tag{3}$$

The generator has no embedded pooling mechanism and the entropy gathering is performed on the fly. The critical part of generator's inner state is the secret key K that is expected to be stored securely – otherwise the forward security can be no longer guaranteed. Another part of inner state, the value R, is produced directly as an output[5]. Moreover, the DT vector is based solely on low-entropy

[5] The generator was originally designed for generation of single DES keys, therefore the output was expected to be secret.

date/time – this implies slow recovery after a state compromise and therefore weak backward security.

Fortuna PRNG: Fortuna [3] currently represents one of the most sophisticated generator designs with an improved and automated mechanism of pooling and extremely simplified (almost removed) process of entropy estimation.

Fortuna PRNG consists of three components. The first part is the generator itself, it takes a fixed-size seed and outputs an arbitrary amount (often restricted to 1 MB per single request) of pseudorandom data. The generator is the most primitive component of the Fortuna – a block cipher in the counter mode with some refinements. It is recommended to use AES with a 256-bit key and a 128-bit counter. The key and the counter form a secret internal state of the generator.

After every request another 256 bits of pseudorandom data are generated for a new encryption key. Periodical rekeying ensures not only backward security, but also forward security of the generator as it is infeasible to get previous output data after the key change even when the attacker knows the secret internal state of the generator. There is no reset of the counter and this property prevents short cycles that could occur due to repeating key values.

The second component is called accumulator and its main purpose is to collect and pool entropy from various sources of randomness. There are 32 pools, which are filled with data from one or several randomness sources in a cyclical fashion. This design ensures that random events from a single source are distributed evenly over the pools and an attacker controlling only some randomness sources cannot control all these pools. The accumulator is also responsible for periodical reseeding of the generator. Reseeds are numbered (1, 2, 3, etc.) and the pool P_i is used if and only if 2^i is a divisor of the reseed counter. Thus P_0 is used every reseed, P_1 every second reseed, P_2 every fourth reseed, etc. The only entropy estimate that must be done sets the minimal pool size necessary for reseeding – but it can be quite optimistic, e.g., 64 bytes for required 128 bits of entropy.

The seed file is last component of the Fortuna PRNG that serves as the storage of high-entropy data. It preserves the PRNG internal state and ensures that even after rebooting the generator can produce good pseudorandom data. Quite a simple concept in theory, but practical implementation is very difficult and depends extremely on the environment and platform (for details see [3]).

4 Integration into Symbian OS

In this section we summarize details of our implementation and integration of ANSI X9.31 and Fortuna PRNGs into the Symbian OS 9.x. We used the smartphone Nokia N73 for our practical experiments and performance tests.

4.1 ANSI X9.31 PRNG

We implemented the ANSI X9.31 PRNG as a simple GUI application[6] that is based on the AES encryption function and the SHA-1 hash function. SHA-1 is

[6] The source code is available at: http://sourceforge.net/projects/ansix931prng/.

an internal Symbian library function and the AES is implemented as a reference code provided by the IAIK Krypto Group AES Lounge ported to the Symbian OS by Philipp Henkel [5]. The generator itself is implemented in one class.

There are object attributes that preserve the generator secret state and that are necessary for the generator initialization – the most important is a disposable 320-bit entropy pool. It is implemented as two SHA-1 contexts and the initial entropy comes from 5 samples from the camera viewfinder, 20 audio samples from the microphone and the timing of each keystroke. Construction and initialization of the camera and microphone objects run asynchronously and the speed of data acquisition from these sources is 1495 KB/s. Since the estimated amount of entropy from these samples radically exceeds 320-bit, we can expect that the pool contains 320-bit of entropy. The pool is used only for following operations: first 128 bits are used as AES key material, next 128 bits fill in the seed and remaining 64 bits initialize first half of the date/time vector. As the generator is not reseeded during its runtime, it never recovers from a compromised state.

We improved the original X9.31 generator by initializing the first half of date/time vector by truly random data that remains fixed during whole generation process, while the second half is updated in every iteration by 64-bit of date/time. This is obviously a tradeoff between security and energy efficiency – better backward security can be achieved by using purely truly random data instead of low-entropy date/time, but the continuously running camera and microphone will drain the battery very soon. Another disadvantage is that the running camera is not accessible for other applications.

This implementation can be used in applications that require some random data at the start, but we do not recommend its continuous usage. The generator has in fact poor forward and backward security. In case of forward security, it is very easy to get previously generated random bits, if an attacker gets the secret state of the generator. We also implemented another version of ANSI X9.31 based solely on a one-way hash function (in our case SHA-256), which provides a better forward security for the mobile environment, as there are no problems with key management and secure key storage [4].

Backward security depends crucially on the entropy of the date/time vector. However, after a state compromise the length of DT is in fact reduced to 64 bits and we must expect that the attacker can predict a significant amount of DT vector bits. An optimistic assumption is that the attacker knows the time of the data request with a minute precision. Theoretical precision of the time in Symbian is in order of microseconds, but the analysis has proved that the time value is always divisible by 125. It reduces the number of possibilities to 480 000, and so approximately 18.87 bits of entropy. If the attacker can predict the time with second precision, then he has only 8000 possibilities, i.e., 12.96 bits of entropy. Cryptanalytic attacks on ANSI X9.17/X9.31 are discussed in [9].

The actual speed of our implementation is only 2.44 KB/s with the reference non-optimized AES implementation of encryption speed 75 KB/s. The average speed of implementation based on SHA-256 is 3.9 KB/s. The above informal security analysis and slow performance in mobile phones clearly suggests that practical usage of the ANSI X9.31 PRNG is not advisable.

4.2 Fortuna PRNG

In this subsection we describe the implementation of Fortuna PRNG on Nokia Series60 mobile devices with the Symbian OS 9.x[7]. As compatibility is not as straightforward as claimed by the vendors, it is worth noting we used the Nokia N73 smartphone, as there can be some differences when using other models.

Fortuna PRNG is designed to run continually, gathering entropy and servicing user requests for pseudorandom data. Therefore we decided to implement Fortuna by means of the Symbian OS *Client-Server Framework*. The server component is the application without a graphical user interface, implementing whole functionality of the Fortuna PRNG. The client component is the interface to the Fortuna PRNG, implemented as a dynamic-link library. This library with four exported functions allows to start Fortuna server, create new session with the running server, and perform several operations with an established sessions (e.g., requests the Fortuna server for pseudorandom data). The Fortuna server is a crucial part of the Fortuna PRNG, implementing all the functionality. It is a common Symbian OS application without a GUI, running on the background as a separate process, collecting entropy and servicing user requests. The generator component class consists of 256 bits long AES key, 128 bits long counter and two objects representing the cipher context and its key. The accumulator component of the Fortuna PRNG consists of the array of 32 objects – each object represents one pool containing entropy from random events. While the pools are parsable strings of unbounded length, it is more practical to implement the pools just as hash contexts.

The seed file component does not have its own implementing class. It tries to open the seed file in the `/Private` folder of the application. This folder is accessible only to this application and to processes with the capability *AllFiles*. When the seed file exists and contains at least 64 bytes of data, the generator is seeded with this data and the seed file is updated with the newly generated 64 bytes of pseudorandom data. If the seed file is not found (or is not long enough), the generator will not be seeded and user data requests will have to wait until the pools accumulate enough entropy to initialize the generator properly. The Fortuna class contains two object attributes dedicated to reseeding. A new 64-bit counter incremented at each reseeding and the time when the last reseeding was done. At least 100 ms delay is required between two successive reseedings.

We implemented 3 randomness sources: the keyboard (`CFKeyRandSrc`), the microphone (`CFAudioRandSrc`) and the camera (`CFCamRandSrc`). The first source gathers entropy from keyboard events. The randomness source starts waiting for the key events and when the key is pressed, the window server[8] generates the key event and sends it to our application. The key event data itself does not contain much entropy, therefore it is not used at all. Only the time (12.96 bits of entropy) of key event arising is sent to the appropriate pool. The second randomness source gathers entropy from the microphone device. According to our entropy estimation 1 KB sample contains 51 bits of entropy. The period and

[7] The source code is available at: http://sourceforge.net/projects/fortunaprng/.

[8] A server which manages screen, keyboard and pointer on behalf of client applications.

the exact amount of data taken should be precisely tuned to fulfill the goals of the target application. In our case the audio sample is taken every minute.

The last implemented randomness source gathers entropy from the camera device. As one frame from the viewfinder yields almost 100 KB, the data is spread over all pools. Strictly speaking, 3180 bytes of data is added to each pool. According to our estimations, this represents 11 080 bits of entropy for each pool. Although in theory it is possible to capture the viewfinder frames continually, it would limit the device user considerably. The camera has a significant power consumption and moreover it would be reserved for the Fortuna server only. Therefore, we capture only one viewfinder frame every minute.

We performed five battery life tests[9] with continuous utilization of different sources of randomness. Keyboard events have been monitored at all times, but no keys were pressed during measurements. The lowest speed of our implementation in the continuous capturing mode is 13.85 KB/s. The detailed results are summarized in Table 1 – the first column presents the average time of battery life and the second column then standard deviation of our measurements. Our

Table 1. Fortuna energy requirements – battery life

Sources of randomness (continuous sampling)	Avg. time [hh:mm]	Std. deviation [mm:ss]
Keyboard, microphone	17:27	8:34
Keyboard, camera	3:48	2:24
Keyb., cam., micr.	3:24	3:36

current Fortuna implementation captures both audio and video periodically just once a minute. In this case the battery is exhausted approximately after 3 days, 18 hours and 13 minutes. This is a non-trivial battery stress in comparison to 10 days, 11 hours and 1 minute of producing pure pseudorandom data without any entropy pooling. Energy requirements of our current reference Fortuna implementation are 2.78 times higher then energy requirements of implementation without any entropy pooling. For non-critical applications we thus recommend to capture data only once per 5 minutes – the battery is then exhausted approximately after 7 days, 11 hours and 34 minutes.

5 Conclusions and Future Work

In this paper we show a practical approach to random data generation for the mobile environment, providing a clear path to developers of many security-critical applications for the mobile environment. We investigated several possible ways of generating both random and pseudorandom data in mobile devices with Symbian OS. We start with the identification of available sources of randomness and assessment of their quality and performance. Our analysis showed that mobile devices have several good sources of randomness – we confirmed that at least the

[9] All measurements were performed with a Li-Pol battery type BP-6M (970 mAh).

microphone and camera noise contain a sufficient amount of entropy and thus can be reliably used as a good sources of truly random data.

Our work then lead to the investigation of the truly random data postprocessing with the use of pseudorandom number generators that are able to utilize all accessible sources of randomness in the generation process. We selected ANSI X9.31 and Fortuna pseudorandom number generators that we then successfully implemented and integrated into the Symbian-based smartphone Nokia N73.

We also want to point out that on top of the truly random data sources that we examined further, other sources like battery and signal level (and in some specific situations also GPS position tracking) are worth investigating. This can be done, for example, on the completely open-source Linux-based OpenMoko cellphone or at a higher granularity by using external devices (such as a cellular modem or GPS unit). Work with Symbian OS phones, however, can also be addressed by teams that have a lower-level (API) access to the devices than that available to us and other ordinary developers.

Acknowledgment. We acknowledge the support of the research project of Czech Science Foundation No. 102/06/0711 and of the FIDIS Network of Excellence.

References

1. Kennedy, J.: Digital camera fundamentals. Andor Technology, `http://www.andor.com/pdfs/Digital%20Camera%20Fundamentals.pdf`
2. Eastlake, D., Cybercash, S.C., Schiller, J.: RFC1750: Randomness Recommendations for Security. MIT Press, Cambridge (1994)
3. Ferguson, N., Schneier, B.: Practical Cryptography. John Wiley & Sons, Chichester (2003)
4. Gutmann, P.: Software generation of practically strong random numbers. In: Proc. of the 7th USENIX Security Symposium, pp. 243–257. USENIX Association (1998)
5. Henkel, P.: Port of Rijndael Block Cipher to Symbian OSs (2005), `http://www.newlc.com/AES-Encryption.html`
6. Krhovjak, J., Svenda, P., Matyas, V.: The sources of randomness in mobile devices. In: Proc. of the 12th Nordic Workshop on Secure IT System, pp. 73–84. Reykjavik University (2007)
7. Krhovjak, J., Svenda, P., Matyas, V., Smolik, L.: The sources of randomness in smartphones with Symbian OS. In: Security and Protection of Information 2007, pp. 87–98. University of Defence (2007)
8. Keller, S.S.: NIST-recommended random number generator based on ANSI X9.31 Appendix A.2.4 using the 3-key triple DES and AES algorithms. NIST (2005)
9. Kelsey, J., Schneier, B., Wagner, D., Hall, C.: Cryptanalytic attacks on pseudorandom number generators. In: Vaudenay, S. (ed.) FSE 1998. LNCS, vol. 1372, pp. 168–188. Springer, Heidelberg (1998)
10. Rukhin, A., Soto, J., Nechvatal, J.: A statistical test suite for random and pseudorandom number generators for cryptographic applications. In: NIST Special Publication 800-22 (2001)
11. Shaltiel, R.: Recent developements in explicit constructions of extractors. In: Bulletin of the EATCS, pp. 67–95 (2002)