Faculty of Informatics
Masaryk University

Cryptographic random and pseudorandom data generators

Dissertation thesis

Jan Krhovják

Brno, 2009

I would like to dedicate this thesis to my loving parents...

Acknowledgements

First, I would like to thank my supervisor Vashek Matyas for giving me the chance to be a part of his research security group and for his care and support. He always motivated me even when things did not go very well.

I also would like to express my gratitude to Luděk Smolík for his fruitful discussion about the physical sources of randomness. Discussions and ideas given by Dan Cvrček also greatly contributed to this thesis.

I am thankful to all my colleagues from the laboratory and especially to Petr Švenda and Marek Kumpošt. We spent a really great time working together and discussing our projects, but also a lot of fun and great experience.

I am also grateful to Andreas Pfitzmann for giving me the opportunity to spent a very productive time with him and his research group in Dresden. This was a really nice experience that also greatly contributed to this thesis.

I also like to thank all anonymous reviewers for their constructive comments and suggestions which also helped to improve our research.

Finally, I would like to express my gratitude to my family. They were a constant source of support, understanding, care and encouragement during my whole academic life.

I am generally grateful for all the support I have received whilst researching and writing up this dissertation.

I acknowledge the support of the research project of Czech Science Foundation No. 102/06/0711.

Jan Krhovják, Brno, January 2009

Abstract

This dissertation thesis deals with cryptographic random and pseudorandom data generators in mobile computing environments (such as mobile phones, personal digital assistants, cryptographic smartcards). These mobile devices are typically bounded by the amount of energy, performance, memory or even silicon area. This lack of resources leads to very limited computing environments with: a) limited number of sources of randomness for reliable true random data generator; b) limited number of pseudorandom data generators (or other methods of digital post-processing) suitable for secure and efficient implementation (in this mobile environment).

In the first chapter we explain the basics of random and pseudorandom data generation for cryptography purposes and also all necessary terminology. The second chapter is organized as a survey of basic requirements on random and pseudorandom data, fundamental results in the field and description of several experiments with cryptographic smartcards. In the last part of this chapter we also discuss a novel idea of distributed random data generation. The third chapter is dedicated to identification and analysis of available sources of randomness in mobile computing environments. The fourth chapter focuses on the secure and efficient digital post-processing of truly random data with the use of randomness extractors or pseudorandom data generators. The fifth chapter presents technical details regarding the integration of our design prototype into selected mobile devices (smartphone Nokia N73 with Symbian OS).

The main contribution of this dissertation is the identification and analysis of available sources of randomness in mobile devices, secure integration of selected digital post-processing methods (resulting in a design prototype) and the analysis of approaches to distributed random data generation.

Table of Contents

Chapter 1

Introduction

In today's world of modern computers and ubiquitous networks most security solutions rely on the use of cryptography. Generation high-quality randomness is a vital (and maybe most difficult) cornerstone of many cryptographic operations and the importance of a careful design of cryptographic (pseudo)random data generators cannot be underestimated.

Following the second Kerckhoffs' principle – the security of a cryptosystem shall not be based on keeping the algorithm secret but solely on keeping the key secret – the quality and unpredictability of secret data (e.g., cryptographic keys, padding values or per-message secrets) is critical to securing communication by modern cryptographic techniques. Generation of such data for cryptographic purposes typically requires an unpredictable physical source of random data, secure mechanisms for its digital post-processing and/or secure and robust methods for gathering random data from one or several distributed systems.

Most common generation techniques involve truly random and pseudorandom data generators. The former are typically based on nondeterministic physical phenomena (e.g., radioactive decay or thermal noise), while the latter are typically deterministic algorithms where all randomness of the output is dependent on the randomness of the one or several inputs (often called seed). The process of generation truly random data in the deterministic environment of computer systems or even single chip devices is extremely hard and slow, i.e., only a small amount of good quality random data can be generated in a reasonable time. Therefore we often restrict ourselves to the use of deterministically generated pseudorandom data instead of truly random data.

Generation of pseudorandom data is in most computational environments faster and truly random data is in this process used only as an initial input. Since the whole generating process is deterministic, the randomness of the output is fully dependent on the randomness of the input. There exist many classes of pseudorandom data generators (designed, e.g., for simulation purposes or randomized algorithms), but the goal of a pseudorandom data generator in cryptography is to produce pseudorandom data that is unpredictable and computationally indistinguishable from truly random data.

Verification of the statistical quality of (pseudo)random data by detecting deviations from true randomness (known a priori) is based on statistical testing. These tests may be useful as the first step in determining whether or not a generator is suitable for a particular cryptographic application. However, no set of statistical tests can surely point out a generator as appropriate for usage in a particular application, i.e., statistical testing cannot serve as a substitute for cryptanalysis. In addition, the results of statistical testing must be interpreted with some care and caution to avoid incorrect conclusions about a specific generator.

1.1 Basic terminology

The term *random* or *randomness* covers also unpredictability (as is required in cryptography). The measure for randomness is called *uncertainty* or *entropy* [Sha48] and is typically defined [ECS05, MvOV01] as $H(X) = -\Sigma p_i log_2 p_i$, where i varies from 1 to the number of possible values of random variable X and p_i is the probability of the value numbered i.

A *random bit generator* (RBG) is defined [MvOV01] as a device or algorithm which outputs a sequence of statistically independent and unbiased binary digits. The output of the RBG is called a *random bit sequence*. A *pseudorandom bit generator* (PRBG) is defined [MvOV01] as a deterministic algorithm which, given a truly random binary sequence of length k, outputs a binary sequence of length $l \gg k$ that "appears" to be random. The input to the PRBG is called the *seed*, while the output of the PRBG is called a *pseudorandom bit sequence*.

There is no significant difference between *truly random data* such as truly random bits, truly random bit sequences, truly random numbers, or truly random number sequences. Similarly there is no significant difference between *pseudorandom data* such as pseudorandom bits, pseudorandom bit sequences, pseudorandom

numbers, or pseudorandom number sequences. A (pseudo)random bit sequence can be constructed by concatenation of (pseudo)random bits. A (pseudo)random number in interval $\langle 0, n\rangle$ can be converted from (pseudo)random bit sequence of length $\lfloor log_2 n \rfloor + 1$ (if n is exceeded, new sequence should always be used). A (pseudo)random number sequence can be constructed by concatenation of (pseudo)random numbers. Consequently, there is no significant difference between appropriate generators of these types of (pseudo)random data.

The terms *random challenge* and *nonce* (contraction of "number used once") have in the context of cryptographic protocols the same meaning as random data.

1.2 Structure of the thesis

This dissertation thesis[1] deals mainly with issues related to the generation of truly random and pseudorandom data (i.e., bits, numbers and sequences) in mobile computing environments (such as mobile phones, personal digital assistants or cryptographic smartcards). These mobile devices are typically bounded by the amount of energy, performance, memory or even silicon area and this lack of resources leads to very limited computing environments. On the contrary, the mobile devices are now commonly used for a lot of security-critical applications like mobile banking, secure data and voice communication, etc. Good sources of randomness and suitable methods of their digital post-processing are thus the most critical requirements for a reliable random data generation.

Chapter 2 presents the expected application requirements in terms of amount and speed of random data generation in such environments followed by a detailed description of fundamental results in the research area of random number generators. Special attention belongs to experiments with hardware generators of selected cryptographic smartcards and hardware security modules. An analysis of approaches to distributed random or pseudorandom data generation in hostile environment is also discussed in this chapter.

Chapter 3 is focused on identification and analysis of available sources of randomness in selected mobile devices (Nokia N73 with the Symbian OS and E-Ten X500 and M700 with MS Windows Mobile OS). Several experiments are performed with available mobile devices in different external conditions (such as temperature, ambient light, acoustic noise). The analysis of experiments results

[1] The research was supported by the grant of the Czech Science Foundation No. 102/06/0711.

followed by statistical testing of randomness show that at least the microphone and camera noise contain a sufficient amount of entropy and thus can be reliably used as a good sources of truly random data.

Chapter 4 deals with methods of digital post-processing of truly random data with the use of randomness extractors or pseudorandom data generators. Both could be used to post-process the data captured directly from a physical randomness source and to ensure the uniformity of output and overcom certain correlations or statistical dependencies caused by hardware sources of truly random data.

Chapter 5 is then dedicated to a detailed technical description and integration of design prototype into selected mobile device. Symbian-based smartphone Nokia N73 is used as our target platform for integration of ANSI X9.31 an Fortuna pseudorandom data generators. The performance analysis and power consumption of these generators are also discussed.

Note that the reader of this theses should be familiar with the basics of cryptography and discrete mathematics that can be found, e.g., in [MvOV01].

1.3 Literature review

The work presented in this dissertation is based predominantly on the following publications:

- [Krh06a]: *Analysis, demands, and properties of pseudorandom number generators.* Main content: Demands of common cryptographic schemes on random data and description of properties of several pseudorandom data generators.
- [KŠMS07]: *The sources of randomness in smartphones with Symbian OS.* Main content: The identification and analysis of sources of randomness in smartphones with Symbian OS.
- [KŠM07]: *The sources of randomness in mobile devices.* Main content: The identification and analysis of sources of randomness in mobile devices.
- [KSM08]: *Generating random numbers in hostile environments.* Main content: The proposal of a distributed generation of random data.

- [BKMŠ09]: *Towards true random number generation in mobile environments*. Main content: Security issues related to a digital post-processing by randomness extractors.

- [KMŽ09]: *Generating random and pseudorandom sequences in mobile devices*. Main content: A study of qualitative and quantitative requirements on random and pseudorandom data followed by a description of secure integration (and implementation) issues of selected pseudorandom data generators.

- [Krh06b]: *Cryptographic random and pseudorandom number generators*. Main content: A detailed description of fundamental results in the field of random data generators.

- [CKM+04, CHK+05, BCK+06, KKK+07, KKL+08]: Smartcards, final report for the Czech National Security Authority (2004–0008). Main content: Smartcard security research.

Chapter 2

Random data in cryptography

This chapter provides a basic analysis and description of requirements on random data that is inspired by [Krh06a, KMŽ09]. This is followed by a summary of the current state of the research area that is based on [CKM+04, Krh06b]. After a detailed description of fundamental results in the field we focus mainly on the mobile computing environments (involving mobile phones, personal digital assistants, cryptographic smartcards, etc.).

Description of several experiments with cryptographic smartcards and statistical testing of sequences generated by such smartcards are adopted from [CHK+05, BCK+06, KKK+07]. The last part of this chapter is focused on basic security aspects of distributed random data generation in potentially hostile environments and is based on [KSM08].

2.1 Requirements on random data

Let us begin with a basic description of requirements on random data for cryptographic purposes – for details see [Krh06a, KMŽ09]. 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.

2.1.1 Qualitative requirements

Rigorous qualitative analysis of randomness and unpredictability is probably the most challenging problem for many researchers in this field. Random data generated directly from a randomness source (the most common techniques are described in section 2.2.1) often contains some statistical defects or correlations causing parts of the random data to be easily predictable. These statistical defects are typically inducted by hardware random data 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 a certain level) good statistical properties of generated random data. This can be partially solved by using digital post-processing and/or statistical testing. Digital post-processing is a 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.

Typical techniques of digital post-processing involve use of deterministic pseudorandom data generators (the most important of them are described in section 2.2.2) or randomness extractors (discussed in section 2.2.3). 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 data generators as well as randomness extractors can be based on cryptographic primitives[1] (e.g., hash functions) or simple mathematical functions (consider, e.g., von Neumann corrector [Neu51]). However, none of deterministic digital post-processing 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.

There are several commonly used statistical test suites or batteries (for example CRYPT-X, DIEHARD, and NIST) for manual verification of the statistical

[1] Randomness extractors need not use any cryptography at all, although there may be efficiency gains in using cryptographic components for this task [BH05].

quality of random data by detecting deviations from true randomness (for details see section 2.3). Reduced batteries of automatic tests are often implemented in hardware security modules, smartcards or other security-critical devices – they can be used occasionally after every restart of the device or continuously during generation. However, no finite set of statistical tests can be viewed as complete (there are infinite statistical defects). 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.

2.1.2 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 [Sha48]. 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 – this is a common approach especially in the theory of randomness extractors. These two entropy measures are special cases of generalized Rényi entropy [Rén60].

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 lead to biased results in terms of entropy. This is a fundamental problem that can be also seen in the theory of randomness extractors. To partially cope with this problem, we perform all our experiments and entropy estimations (described in chapter 3) in worsened conditions – i.e., under ongoing attack(s) on the physical randomness source.

The introduction of a measure of randomness allows for assessing and comparing sources of randomness according to the amount of produced entropy. It can be expressed as amount of entropy per sample (of the same size) or as amount of entropy in a given time period. The performance of available randomness source can be a crucial factor for securing the outgoing communication.

The demands of common cryptographic schemes on random or pseudorandom data generation (detailed description can be found in the section 2.1.3) are between hundreds and thousands bits of entropy. The larger amounts of random data are typically required for asymmetric cryptography (e.g., private/public keys or domain parameters), while the smaller amounts are typically required for symmetric cryptography (e.g., secret keys or initialization vectors). Of course, some schemes or applications require only a one-time generation of static random data (e.g., cryptographic keys), while others require more frequent generation of dynamic random data (e.g., initialization vectors). It is also not surprising that these requirements are higher if random data is intended for long-term usage and smaller if random data is intended for short-term usage.

The requirements on random data have always an upper bound given by the speed of the transmission technology in question (Bluetooth, WiFi, EDGE, WiMAX, etc.). They are also dependent on the application type that can be categorized as interactive (e.g., transfer of voice or video), semi-interactive (e.g., web-based services), and non-interactive (e.g., one-time file transfer or sending an e-mail message). The data can be encrypted and transferred immediately after creating/filling an outgoing packet (to prevent unwanted delays), or after reaching other pre-specified thresholds. A new initialization vector (and often also padding) is required for each encryption. Splitting data to several independent pieces thus implies higher requirements on initialization vectors (and padding). The consequence is higher requirements on the volume of random or pseudorandom data.

2.1.3 Demands of common cryptographic schemes

In this section we focus on the few (from many) existing cryptographic schemes and we provide a basic comparison of their demands on (pseudo)random data. Overall demands are given by the length of the supported key (including frequency of re-keying) and by the amount of additional (pseudo)random data necessary to perform each encryption/sign operation.

We begin with the cryptographic keys and initialization vectors, because they are the most critical for security of many cryptographic primitives. Then we approach to various padding schemes, cryptographic protocols, and other mechanisms that also require (pseudo)random data.

2.1.3.1 Cryptographic keys and initialization vectors

Symmetric cryptosystems have typically strictly defined structure and supported length of keys and initialization vectors that cannot be changed without redesign the usage model (e.g., from DES to 3DES) or the cipher itself (e.g., from CAST-128 to CAST-256). The length of secret keys begins for a vast majority of commonly used block ciphers, which are currently considered as secure (e.g., 3DES, AES, IDEA, and RC6), at 112 bits and typically does not go beyond 448 bits. Ciphers that support only keys less than 80 bits long (e.g., DES) are considered obsolete and insecure. On the contrary, keys between 256 and 448 bits in size (supported, e.g., by Blowfish and MARS) are used in practice very rarely and longer keys (supported, e.g., by RC5 and RC6) even not at all.

With respect to the used mode of operation of block cipher, the additional (pseudo)random data – the initialization vector (IV) – can be also required. The cipher then produces different output (ciphertext) for the same input (plaintext) and secret key without a complex process of re-keying. It can be viewed as randomization of the encryption process that is performed (in the CBC mode) by XORing plaintext IV to the first input block or (in the CFB and the OFB mode) by XORing encrypted IV to the first input block. The length of IV for block ciphers is thus always the same as length of the block itself (i.e., typically 64, 128, or 256 bits).

Requirements described above are very similar for commonly used stream ciphers (e.g., A5, E0, HC, and Py). Supported key-lengths typically do not go beyond 256 bits and almost all stream ciphers also require initialization vectors – here mostly comparable to the length of the used key. Sometimes a design of stream ciphers (e.g., RC4) does not provide dedicated input for IV and incorporates IV as a part of a key (e.g., 24bit IV in WEP or 48bit IV in WPA; “WEP key” or “WPA key” is thus concatenated key and IV, no longer than 256 bits).

In both block and stream ciphers, initialization vectors need not be always secret, but they must be always different (i.e., freshly generated) – at least for the encryption process with the same secret key. The need of (pseudo)random data is thus strongly dependent on the frequency of the performed encryptions (and this is dependent on the application itself).

Cryptographic keys (or more precisely key pairs – private key and public key) used in a vast majority of asymmetric cryptosystems are typically several times larger than the keys used in symmetric cryptosystems. The reason is that asymmetric

cryptosystems depend on the intractability of certain mathematical problems – defined mostly over finite fields and using modular arithmetic – and their solution is not as time consuming as an exhaustive search of the key space. Lengths of keys (that are currently considered as secure) begin in commonly used asymmetric cryptosystems at 1024 bits and typically do not go beyond 8192 bits. The only exceptions are cryptosystems working over elliptic curve group in which modular arithmetic is replaced by operations defined over elliptic curves. This mathematical model has a higher complexity than modular arithmetic and key-lengths can be thus shorter – typically between 160 bits and 512 bits.

Asymmetric cryptosystems are (in comparison with symmetric) easily parameterizable and a variable key-length is typically restricted only by the particular implementation. On the contrary, not all generated random data can directly serve as a key. Due to the underlying mathematical problems, the key must typically have specific algebraic properties – e.g., its number representation must be a prime number. Moreover, each public key must be related to the certain secret key. After generating (pseudo)random data (that represents some number) it is typically necessary to perform an additional amount of computation to check the properties of such generated number, or possibly, to find the nearest number with required properties.

Three most common asymmetric cryptosystems are RSA, DSA, and ECDSA. The RSA public key consists of modulus n and exponent e, while the RSA private key consists of (the same) modulus n and exponent d. Modulus n is a product of two (pseudo)randomly generated primes p and q, exponent e is an integer such that $1 < e < \phi(n)$, and exponent d is an integer that depends on both n and e – concretely $de \equiv 1 \ (mod \ \phi(n))$. The size of RSA key pair is typically bit-length of its modulus $n = pq$ and should be equal or larger than 1024 bits – this implies that a bit-length of values p, q should be at least 512 bits. Moreover, exponent e can be also generated (pseudo)randomly and overall demands on (pseudo)random data are thus bounded by n (if e is chosen) and $2n$ (if n-bit e is generated).

DSA makes use of several public parameters: a prime modulus p ($2^{1023} < p < 2^{1024}$), a prime divisor of $p-1$ denoted q ($2^{159} < q < 2^{160}$), and a generator of the subgroup of order $q \ mod \ p$ denoted g ($1 < g < p$). The DSA private key x is a (pseudo)randomly generated integer with $0 < x < q$ and public key is an integer $y = g^x \ mod \ p$. The size of DSA key is a bit-length of its prime modulus p. An additional secret (pseudo)random integer k ($0 < k < q$) – the per-message secret – must be computed prior to the generation of each signature. Larger values for p (1024, 2048, and 3072 bits) and q (160, 224, and 256 bits) are proposed

in [NIS06]. This leads to 6400 bits[2] for public parameters, 256 bits for private key, and 256 bits for each per-message secret. Currently, a longer key than for RSA is often recommended.

Similarly, ECDSA [JMV01] uses also public parameters – so-called domain parameters – that consists of a suitably chosen elliptic curve E defined over a particular field F_q of order q (either an odd prime or a power of 2), characteristic p, and base point $G \in E(F_q)$. These parameters can be specific to a single user or shared by a group of users. To avoid some common attacks it is necessary that the number of points on E must be divisible by a sufficiently large prime n (typically $n > 2^{160}$ and $n > 4\sqrt{q}$). The ECDSA private key d is a (pseudo)randomly generated integer from 1 to $n - 1$, and public key Q is a multiply of private key and base point, i.e., $Q = dG$. The per-message secret – integer k, where $0 < k < n$ – must be also computed prior to the generation of each signature. Demands on (pseudo)random data (without domain parameters) are thus at least 160 bits for private key and 160 bits for each per-message secret.

The fundamental question dealing with the impact of length of cryptographic keys to their security is often discussed. [Gir08] provides key length recommendations of the European Network of Excellence for Cryptology (ECRYPT), the National Institute of Standards and Technology (NIST), the Central Information Systems Security Division (DCSSI) and the National Security Agency (NSA). The most actual results from NIST are described in the table 2.1.

Lifetimes	Bits of security	AES key	DSA/RSA key	ECDSA key
Through 2010	80	–	1024	160
Through 2020	112	–	2048	224
Beyond 2030	128	128	3072	256
	192	192	7680	384
	256	256	15360	512

Table 2.1: Supported key lengths of commonly used cryptosystems.

Random or pseudorandom data can be used also to pad messages before their encryption/sign (i.e., padding) or to combine with a password (i.e., salting) to prevent dictionary attacks. In the next part we summarize demands of selected schemes closely.

[2]This is theoretical upper bound – the values p and g are not statistically independent.

2.1.3.2 Padding schemes and salting

Many cryptographic primitives use padding to extend messages to the required length of a block or integer multiple of a block. It can be used in symmetric cryptography by various block ciphers or cryptographic hash functions and in asymmetric cryptography by various encryption or digital signature schemes. The only exceptions are symmetric stream ciphers that do not require padding at all, because they do not need to divide the message into fixed-length blocks[3].

Block ciphers (in the CBC mode) and cryptographic hash functions typically use some form of structured deterministic padding appended (in)to the last block of the message. The former can fill out last block, for example, by n bytes of the value n (according to RFC 1423 [Bal93]); and the latter, for example, by bit 1 followed by zero bits and by total length of the message – as in MD5 (for details see RFC 1321 [Riv92]). However, correct interpretation and removal of the pad requires padding even if the original message filled up the last block completely. In this case, a new block for padding must be always added; otherwise the end of the input plaintext might be misinterpreted as padding. A poorly designed padding scheme can also (surprisingly) result to various side-channel attacks [Vau02, KR02].

The situation is more complicated in the case of asymmetric cryptosystems. Good padding scheme must ensure that a message block does not fall into the range of insecure plaintexts (for RSA, e.g., 0, 1, $n-1$ and others in dependence on public key) that can be easily decrypted (or even never encrypted). Moreover, a randomized padding should be used together with all deterministic cryptosystems (e.g., RSA) to effectively prevent dictionary attacks and ideally also chosen-ciphertext attacks. In consequence of these requirements, padding schemes are closely related to a particular cryptosystem.

The most common padding schemes are optimal asymmetric encryption padding (OAEP) [BR94], probabilistic signature scheme (PSS) [BR94], full-domain hash (FDH) [Cor00], provably secure elliptic curve encryption scheme (PSEC) [OP00], and their various improvements and modifications [Sho01]. OAEP and PSS are randomized padding schemes whereas FDH is a deterministic one.

For a better illustration of demands on (pseudo)random data we focus only on randomized padding schemes adapted for the RSA algorithm. Two such encryp-

[3]However, sometimes a padding of short messages is necessary to obscure the length of communication (or even traffic analysis).

tion schemes (RSAES-OAEP and RSAES-PKCS1-v1_5) and signature schemes (RSASSA-PSS and RSASSA-PKCS1-v1_5) are specified in PKCS#1 [RSA02].

RSAES-OAEP is parameterized by the choice of a mask generation function (MGF) and hash function – both fixed for a given RSA key. The length of this key (or more precisely modulus) is k bytes. The MGF input is a random seed and a length of the required output. The length of the hash function output is $hLen$ bytes. Each message block – of length $mLen$ bytes – is prior to the encryption formatted to a data block (DB). This data block DB is filled by $k - mLen - 2hLen - 2$ zero bytes and masked by XORing together with data generated by MGF. The input to MGF is a random seed S of length $hLen$ bytes. Masked DB is then used as a seed to MGF and newly generated data is used for masking the previous seed S. Both, masked seed S and masked DB are then used to form encoded message suitable for RSA encryption.

RSAES-PKCS1-v1_5 forms this encoded message directly. Its structure is very similar to DB, but instead of zero bytes uses between 8 and $k - mLen - 3$ bytes of (pseudo)random data. RSASSA-PSS uses a randomized EMSA-PSS encoding and requires typically $hLen$ bytes of a (pseudo)random salt. However, randomness is not critical to its security and usage of fixed values or sequence numbers results in a scheme similar to FDH. RSASSA-PKCS1-v1_5 uses deterministic EMSA-PKCS1-v1_5 encoding without requirements on (pseudo)random data.

Salt is a (pseudo)random data used commonly in the password-based cryptography. Instead of using a password directly as a low-entropy secret key, a combination of (typically non-secret) salt and secret password can be used to prevent brute force attacks (including a dictionary attack). This combination is performed by a key derivation function whose output is often stored as the encrypted version of the password.

PKCS#5 [RSA99] defines two such key derivation functions – PBKDF1 and PBDF2. The former is based on iterative hashing of password concatenated with salt and requires 64 bits of salt. The latter is based on recursively called pseudorandom function and requires only 8 bits of salt. UNIX function “crypt” also uses up to 128 bits of salt (according to particular implementation).

Other password-based techniques typically require incorporation to the cryptographic protocol design and can be thus very heterogeneous. They will be briefly discussed below.

2.1.3.3 Cryptographic protocols

Authentication protocols typically use nonces as random challenges to prevent replay and reflection attacks. Key establishment protocols need random or pseudorandom data to generate shared session keys. Authenticated key establishment protocols mostly combine these techniques (and, of course, also their requirements).

Common authentication protocols (e.g., those based on ISO/IEC 9798 – described in [MvOV01]) require at most four nonces (two generated by each side) per one protocol run. These random challenges are typically 64 (or better 128) bits long. Some of these protocols use timestamps or sequence numbers instead of nonces.

Key establishment protocols can be furthermore divided to key distribution and key agreement protocols. In the first case, only one party of the protocol is involved in the key generation and requirements on (pseudo)random data are thus dependent on the type of the generated key. In the second case, both (or more) parties are involved in the key generation process (issues regarding such multiparty generation of random data are discussed in section 2.4). The overwhelming majority of key agreement protocols are based on Diffie-Hellman exponential key exchange [DH76] where the required modulus must have size at least 1024 bits and the key(s) at least 160 bits [Gir08].

The problem of such protocols is that they are often vulnerable to the man-in-the-middle attack, which can be prevented only by authenticated versions of a protocol. However, cryptographic protocols involving authentication typically require pre-distribution of secret initial keying material. This drawback is removed in password-based protocols, where the password can be used (in certain circumstances) instead of secret key without risk of off-line dictionary attacks. The most famous of them are EKE [BM92], SPEKE [Jab96], and SRP [Wu97].

2.2 Generation of random data

After the analysis and description of requirements on random data we focus on the methods and principles of their generation. It includes a basic summary of techniques of generating truly random data and methods of their digital post-processing by pseudorandom data generators or randomness extractors. This section is based on [CKM+04, Krh06b].

2.2.1 True randomness

A high-quality source of randomness must be used to design a high-quality true random data generator for cryptographic purposes. In a typical environment of general purpose computer systems, some good sources of randomness may exist – almost any user input – the exact timing of keystrokes and the exact movements of mouse are well known. Some other possible sources are for example microphone (if unplugged then A/D convertor yields electronic noise [Ell95]), video camera (focused ideally on some kind of chaotic source as lava lamp [Lav00]), or fluctuations in hard disk access time [DIF94].

Other sources of randomness are process statistics, network statistics, I/O completion statistics, etc. However, they could be in a certain circumstances subjects of influence (i.e., could be predictable) and are thus less appropriate for cryptographic purposes. Almost worthless sources of randomness are then for example system date and time or process runtime. A cautionary example is the true random data generator in the Netscape implementation of secure sockets layer (SSL) protocol that used as randomness sources only the time of day, the process ID, and the parent process ID. Thus, an adversary who can estimate these three values can simply apply the well-known MD5 hash algorithm to compute the seed for pseudorandom data generator [GW96].

Since the amount of entropy extracted from these (or similar) randomness sources is variable (each single source is different), we should be very careful in estimating how much entropy a particular piece of data contains. More detailed description of the internal (computer) hardware randomness sources and requirements can be found in [Ell95, ECS05].

It is generally recommended to use an additional hardware to yield random sequences with sufficient entropy. In modern computers it can be realized by built-in true random data generator, such as the Intel solution [JK99], or specially designed add-on card with true random data generator, such as Quantis quantum random data generator [Qua04].

Beyond the difficulty of collecting truly random data from various randomness sources, several problems related to their practical use were identified [FS03]. Firstly, it is the problem of insufficient amount of truly random data (some applications cannot wait), which can be effectively solved by using pseudorandom data (see section 2.2.2). Secondly, it is the problem of breaking or influencing the true random data generator, which can be partially solved by using digital post-

processing (see section 2.2.2 and 2.2.3) and statistical testing (see section 2.3). Thirdly, it is the problem of judging and estimating the entropy from a particular physical event (no satisfactory solution exists).

As we describe below, all problems relevant to random sources and randomness are more serious in considerably different mobile computing environments – this area is currently most technically challenging. As far as we know, in the mobile computing environments some kind of hardware true random data generator located inside the integrated chip is typically used for generation of truly random data. However, the access to this generator is typically restricted only to applications that run on the subscriber identity module (SIM) card and user applications that run on the mobile phone often use weak sources of randomness, e.g., previously mentioned date and time [SMH05, KMH07].

2.2.1.1 True random data generators in mobile environments

Unfortunately, very few papers related to true random data generators designed for single chip devices (as smartcards) or other cryptographic hardware have been published – design of these generators is mostly kept secret due to the classified nature of most research in this field. Generally, four basic (noise-based) techniques of generating truly random data in a single integrated chip can be distinguished [BBL04]: direct amplification of a noise source [BB99, HCD97], jittered oscillator sampling [BGL+03, PC96, JK99], discrete-time chaotic maps [FCRV03, SK01], or metastable circuits [EHK+03, KC02, WF01].

Even if the hardware true random data generator is well-designed, the produced bit streams usually show a certain level of correlation due to physical limitations (bandwidth limitation, fabrication tolerances, and temperature drifts), implementation issues, deterministic disturbances and external attacks aimed at manipulation (see [BBL04]). A typical solution of these problems is to combine more different generation methods (by applying the XOR operation) or to process the sequences with a carefully designed decorrelating algorithm [PC00]. However, the drawback of such post-processing lies in decreasing the generator speed.

For example, the true random data generator implemented in Infineon cryptographic smartcards (sets 66 and 88) is quite likely based on two analogue oscillators. This type of true random data generator exploits the frequency instability of free running oscillators – a well-known phenomenon used to generate truly random data since 1955. The implementation published in [FMC84] is based on

two oscillators (the first is generating low frequency and the second is generating higher frequency) whose outputs are digitally mixed in such way that the low frequency signal is used to clock the high frequency signal. To improve the statistical characteristic of output (and thus decreasing the speed) a parity filter and a scramble circuit can be used.

The main implementation problem of hardware true random data generators for single chip devices is that the usually common sources of randomness are only hardly accessible inside the digital integrated circuit. Common solutions of this problem often include use of analogue randomness sources inside the digital circuit, even at the cost of high energy and silicon area consumption. Moreover, these analogue circuits should be influenced by periodical signals in close proximity to the generator. The low-cost and fully digital alternative to the true random data generator based on metastability of bistable flip-flop is presented in [EHK+03]. A new type of inverter-based ring oscillator with the ability to be set in metastable mode is described in [VHKK08]

Gemplus developers designed the special integrated chip Randaes that contains (among others) three experimental hardware true random data generators based on direct amplification of a noise source and jittered oscillator sampling. All of these generators are using the same adaptive digital decorrelator that improves statistical properties of generated sequences and guarantees their correctness without subsequent statistical analysis. The detailed description can be found in [BTSL01].

Fully digital high-speed true random data generator based (again) on jittered oscillator sampling and suitable for integration to the cryptographic smartcards is proposed in [BGL+03]. An enhancement of the oscillator-based architecture, where a compensation loop is added to maximize the statistical quality of the output, can be found in [BBL04]. And finally, the design of stateless and testable true random data generators is proposed in [BL05] – the stateless property allows to shift the verification of a minimum entropy limit after the post-processing.

The classified design and implementation details of true random data generators for cryptographic hardware (as for example smartcards or hardware security modules) could clearly result in simple design flaws. Black-box testing of an unknown generator is a difficult task that can involve also external influencing of such generator (e.g., by temperature or particle radiation). We deal with the security and influencing of true random data generators in such devices extensively in [CKM+04, CHK+05, BCK+06, KKK+07]. The generated sequences are

a subject of statistical testing (for details see section 2.3) that provides the only possibility how to asses the quality of an unknown generator.

The situation is more complicated for other mobile computing environments (especially mobile phones or personal digital assistants) that were not designed to perform cryptographic operations and thus they have no dedicated true random data generator. Identification of suitable sources of randomness in such mobile computing environments, evaluation of their acquisition speed, statistical testing of their quality, and basic estimation of the amount of available entropy in a given time period is the primary content of chapter 3.

2.2.2 Pseudorandomness

The problem of insufficient amount of truly random data can be effectively solved by using pseudorandom data (computationally indistinguishable from truly random data [Gol90]). As mentioned before, the truly random data is used only as a seed for deterministic pseudorandom data generator and after seeding, an arbitrary amount of pseudorandom data is always available. Moreover, pseudorandom data generators are typically better theoretically examined (similarly as other algorithms) and pseudorandom data have better statistical properties than truly random data (which may contain in certain cases bias and correlation [Dav00]). From the qualitative point of view, pseudorandom data generators allow us to spread such simple statistical defects into a longer sequence of bits.

The pseudorandom data generator is in fact a deterministic finite state machine, which implies that it is at any point of time in a certain internal state. This generator state is secret (because the generator output must be unpredictable) and during the generating of pseudorandom data is repeatedly updated (because generator must produce different outputs). Of course, a careful designer of pseudorandom data generators should expect that a secret state compromise may occur (e.g., due to the error in implementation or low quality seed as in [GW96]).

Recovering a pseudorandom data generator with a compromised state is a difficult task which is typically based on mixing small amounts of entropy to the secret state. However, if the amount of entropy between two requests for pseudorandom data is limited (e.g., to 30 bits), the attacker can simply make frequent requests for pseudorandom data and try all 2^{30} possibilities for the inputs to obtain secret state. The best solution how to prevent this problem is to pool (i.e., collect) incoming entropy to sufficient amount, and then to mix it to the secret state. We

call the generators that allow to increase the inner state entropy by periodical reseeding and/or continual accumulation (pooling) of truly random data as *hybrid pseudorandom data generators*. More detailed discussion that covers design and analysis of pseudorandom data generators can be found in [KSF99, FS03], and many practical problems have been addressed also in [Gut98, Gut04].

Currently the most sophisticated pseudorandom data generators are Yarrow-160 [KSF99] and its successors Tiny [VM01] and Fortuna [FS03] – all designed with respect to known cryptanalytical attacks [KSWH98]. The usability of these (and also many other existing generators) in various mobile computing environments is questionable and strongly dependent on particular types of mobile devices. Aside from the elementary difficulty to gather truly random data in mobile computing environments, these mobile devices are typically very slow and performing cryptographic functions (which are used in almost every pseudorandom data generator) is very time and/or energy consuming (even if these functions are hardware accelerated, e.g., in cryptographic smartcards). Note that the card (accelerator) owner is not necessarily the device owner/user.

2.2.2.1 Cryptographic pseudorandom data generators

Basic categorization and description of pseudorandom data generators can be found in [MvOV01]. The most commonly used pseudorandom data generators (primarily for simulation purposes and for implementation of randomized algorithms) include the linear congruential generator (LCG). However, the output of LCG is predictable and thus for cryptographic purposes is this generator completely inappropriate [Boy89, Kra90].

The next and probably simplest construction of pseudorandom data generator is based on linear feedback shift register (LFSR). This class of generators can be very effectively implemented in hardware, is capable to generate very long pseudorandom sequences with a high-quality statistical distribution, and has very well examined properties. Since the generator itself is not too secure, it is typically used as a basic block of more complicated generators. Advanced designs of such generators can be based on a non-linear combination of several LFSRs, or on using one (or several) LFSR to clock another (or combination of more) LFSR. Detailed description of LFSRs can be found in [Neu04, Zen04].

A completely different class of generators is created by cryptographically secure pseudorandom data generators that are based on hard problems of number and

complexity theory. Modular arithmetic (that is typically used) makes these generators very slow, but this can be partially addressed by a sufficient hardware acceleration.

Three generators from this category are described in [MvOV01]: RSA, Micali-Schnorr, and Blum Blum Shub (BBS) pseudorandom data generators. The first two are based on the well-known factorization problem (similarly as the RSA cryptosystem) and the last generator is based on the quadratic residuosity problem. A faster version of BBS generator utilizes Montgomery multiplication (hardware accelerated in many cryptographic smartcards) to improve efficiency (see [PKS00]).

The generator based on the discrete logarithm problem with short/small exponents belongs to the fastest generators in this class. This generator repeatedly performs the exponentiation of the basis to a short exponent – after each iteration several bits are produced as a output and the remaining bits are used as exponent in the next iteration. Detailed description can be found in [Gen00]. A bit slower generator proposed in [DRV02], that is based on the factorization problem, works similarly. A general technique that can be used to speed up pseudorandom generators based on iterating one-way permutations is also presented.

Most commonly used pseudorandom data generators are based on typical cryptographic primitives. Well-known (and for long time the only standardized) generators are ANSI X9.17/X9.31 [ANS85] and FIPS 186 [NIS94b]. ANSI X9.17/X9.31 generator is based on using two- or three-key 3DES, and the generators described in FIPS 186 are based on single DES and/or SHA-1. In addition to that, four new pseudorandom data generators were recently standardized in [BK07] but the security of the generator based on elliptic curves (Dual_EC_DRBG) is now disputed [SF07].

Almost all pseudorandom data generators described above have the property that they will never recover from a state compromise (or they recover only after a very long time). This problem is solved in well-designed Yarrow-160 pseudorandom data generator [KSF99] that is also immune to all known attacks (especially to those described in [KSWH98]). The original design is based on three key 3DES and on SHA-1, but in principle AES can be used together with SHA-2. The successors of Yarrow-160 are Tiny [VM01] and Fortuna [FS03], both encrypting a counter using AES to produce successive blocks of output.

Quite novel design of the Cilia generator [Ng05] uses cryptographic hash functions to update its secret state and double encryption to generate pseudorandom data. The speed of the Cilia is very similar to the speed of double encryption, and

like Yarrow-160 and its successors also Cilia is designed with respect to known cryptanalytic attacks on pseudorandom data generators.

Many commonly used cryptographic libraries (as for example RSAREF, CryptoLib, or OpenSSL) and applications (e.g., GnuPg, Putty, TrueCrypt) uses its own design and implementation of generators that are often inspired by [Gut98, Gut04]. We analyzed source code of several such generators and our results are described in [KKL+08]. We found that a lot of them is non-standardized and without any appropriate documentation and rigorous security analysis (especially mechanism of pooling and extracting randomness from the pool). There are also often a big differences between MS Windows and Linux versions of the same generator – randomness sources, lock of pool in memory, access to the filesystem, etc. The implementation of a seed file is not transparent or is even missing in some cases. The multithreading support of some pseudorandom data generator implementations is another issue.

2.2.2.2 Cryptanalytical attacks on pseudorandom data generators

Only very few papers that deal solely with cryptanalysis of pseudorandom data generators were published. An analysis of pseudorandom data generators from the attacker perspective is described in [KSWH98]. Requirements for pseudorandom data generators are presented along with a basic functional model of such generators, and the list of possible attacks against them. Special attention is given to examining the attacker methods used to cause a given generator to fail, or methods abusing some generator outputs (such as initialization vectors) to guess other generator outputs (such as session keys). The attacks are divided into three categories: cryptanalytic attacks, input-based attacks, and state compromise extension attacks. Pseudorandom data generators ANSI X9.17, FIPS 186, RSAREF 2.0, and CryptoLib are analyzed with respect to these attacks. Countermeasures against the discovered attacks are also provided in [KSWH98].

Careful security analysis of ANSI X9.17 and FIPS 186 pseudorandom data generators is described in [DHY02], together with examining the ways in which these generators can be made more efficient and more secure. Special attention is given to the inputs/outputs, secret state, and used cryptographic functions. Reverse-engineering and analysis of MS Windows 2000 pseudorandom data generator can be found in [DGP07] whereas a description and analysis of open-source Linux pseudorandom data generator is discussed in [ZG06].

A survey and analysis of existing attacks on LFSR-based pseudorandom data generators can be found in [Zen04]. However, the goal this survey paper is design and deployment of pseudorandom data generators and used cryptanalysis has not a destructive, but a constructive character – only by improving the understanding of possible problems, it is possible to propose new design criteria for cryptographic systems. Reviewed cryptanalytical attacks are divided to generic and specific attacks. The former are applicable even if the attacker does not know the design and the structure of the generator. In the latter case, the attacker must know the internal structure of the generator. Deployment of pseudorandom data generator in stream ciphers is also discussed.

2.2.2.3 Pseudorandom data generators in mobile environments

Hardware or software implementation and integration of the pseudorandom data generator into programmable cryptographic smartcards or other mobile devices is often straightforward. The most important properties of pseudorandom data generators for mobile devices are besides security (in the meaning of unpredictability) also speed, low energy requirements (in the case of both software and hardware implementation), and low memory requirements (in the case of software implementation).

Authors of [ARV95] present a pseudorandom data generator based on single DES. More efficient implementation of this generator (described in [ARV99]) requires only 3 KB of memory (instead 16–32 KB as required originally) and is thus more suitable for cryptographic smartcards or other memory bounded single chip devices.

We dealt with secure integration of ANSI X9.17/X9.31 and FIPS 186 pseudorandom data generators into cryptographic smartcards (more precisely JavaCards) in [CHK+05]. The performance of software implementations of such generators is in this computation environment significantly lower than performance of built-in true random data generators. In 2005, our fastest JavaCard GemXpresso Lite-Generic was able generate 7973 KB of truly random data in one hour, 333 KB of pseudorandom data generated by ANSI X9.17/X9.31 generator in one hour, and only 49 KB of pseudorandom data generated by FIPS 186 generator in one hour.

The situation is much better in other mobile computing environments (especially mobile phones or personal digital assistants) that have significantly higher performance. In contrary to the smartcards, such mobile devices typically do not

provide secure computing environment and this implies new security issues to be addressed. Security analysis and modification of ANSI X9.17/X9.31 and Fortuna pseudorandom data generators are discussed in chapter 4.2 and secure integration into mobile devices is the primary content of chapter 5 of this thesis.

2.2.3 Randomness extractors

Randomness extractors (sometimes called entropy extractors) are an interesting alternative to pseudorandom data generators. Extractors are capable to produce (almost) uniformly distributed output from arbitrary distributions with sufficient randomness (min-entropy). The main difference between these two types of constructions lies in the fact that pseudorandom data generators rely on unproven assumptions whereas randomness extractors typically provide some provable guarantees of its output (for more details see [Sha02]).

We can distinguish deterministic or non-deterministic randomness extractors[4]. The former work only on limited classes of randomness sources (with fixed cumulative probability of each such set of inputs) while the latter do not have this restriction. Non-deterministic extractors also need an additional truly random input. The main limitation of both classes of randomness extractors is that probability distributions of post-processed randomness sources must be precisely defined. Otherwise, it is not possible to construct appropriate and well working randomness extractor (explicit extractors constructions, based on results of [Tre99], are discussed in [Sha02]). In addition to that, the usage of an randomness extractor makes the generation of truly random data even slower – i.e., the length of output sequence is equal or (typically) less than the length of input (biased) random data.

Deterministic randomness extractors are sometimes used before pseudorandom data generators – especially in single chip devices like smartcards. In this case the extractor is often implemented as a part of the hardware truly random data generator and applied to the data sampled directly from the randomness source. This is reasonable for a device of the same family/type with the same sources of randomness. Such simple deterministic extractors are capable to reduce statistical dependencies that are typically inducted by hardware generators during the sampling of the analogue randomness source [Dav00, JK99].

[4]Formally, mathematical functions that transform biased input to output with (almost) uniform probability distribution with sufficient min-entropy.

A good example of deterministic randomness extractor is von Neumann corrector (described, e.g., in [Neu51] or [JK99]) that simply converts pairs of bits into output bits by converting the bit pair 01 into an output 1, converting 10 into an output 0, and outputting nothing for 00 or 11. The consequence of von Neumann corrector application is a variable bitrate of the generator.

The non-deterministic randomness extractor is function that in contrary to the deterministic extractor makes use of an additional auxiliary uniformly distributed random input. This means that such data must be pre-generated, and it can be, e.g., stored in the memory and refreshed via network after a fixed number of extractor iterations (meanwhile the output of the extractor is used). This is an obvious drawback, but such an extractor can be used to post-process a wide range of input probability distributions, namely any probability distribution with sufficiently high (pre-chosen) min-entropy [Sha02].

Issues related to integration of non-deterministic randomness extractor into mobile devices including several practical experiments and statistical tests are discussed in section 4.1.2.

2.3 Statistical testing

Statistical tests are used to verify that produced (pseudo)random data have good statistical properties. This can be done manually (for example by someone interested) when a new (but sometimes insufficiently documented) generator is published, and also automatically to avoid breaking or influencing the generator during its life cycle. Statistical test suites (sometimes called test batteries) such as NIST [NIS00] or DIEHARD [Mar95] are typically used for manual testing. Elementary tests integrated into a hardware device are used for automatic testing that can be performed occasionally (after every restart of the device [NIS94a, NIS01]) or continuously (during the process of generating random data [Sch01]).

Standard FIPS 140-1/2 [NIS94a, NIS01] recommends the execution of statistical tests after every restart of a device or card, and (pseudo)random data can be used only if all implemented tests are successfully passed. In contrary to hardware security modules only few cryptographic smartcards perform these tests. However, the vendors of such devices typically implement digital post-processing of all data acquired from the randomness source before sending them outside the

device. All online tests (and even offline tests performed, e.g., by official testing laboratories) are performed with post-processed data and this "cheating" often devaluates the test results. Due to this fact a new FIPS 140-3 [NIS07] standard (still under development) requires continuous testing of both generator and used entropy source involving min-entropy assessment on each output of the entropy source.

According to [Sot00], there exist five basic sources dealing with statistical testing of randomness – the well-known books [MvOV01, Knu97] and the test batteries DIEHARD [Mar95], Crypt-XS [Cry06] and NIST [NIS00]. Various statistical tests and sometimes also the basic test methodology are described (or implemented). However, some tests are obsolete (e.g., those described in [Knu97]), because even obviously weak generators pass all these tests (as claimed in [HSS04]).

On the contrary, the DIEHARD test suite was for a very long time considered as the most complete, but in [MT02] a new suite of three tests was designed, which is claimed to be better than DIEHARD. If some generator passes these three tests (implemented in [Cen02]) then it also (with a very high probability) passes all DIEHARD tests. Few mistakes in the test settings were found also in the NIST statistical test suite – these are described together with the proposed corrections in [SJUH04]. In [HSS04] new SAC (strict avalanche criterion) test is also described, which is effectively capable to detect statistical anomalies of output sequences.

Recent development in the field of statistical testing resulted in two novel statistical test batteries. The first is RaBiGeTe [Pir05] that contains 22 statistical tests including several older tests from DIEHARD, NIST, and FIPS 140-2. The second is TESTU01 that comprises almost all existing statistical tests – the test suite is described together with a survey and a classification of statistical tests for RNGs in [LS07].

An ongoing project pLab [Hel06] is focusing on uniform pseudorandom data generators for stochastic simulation including their statistical testing of randomness. However, pseudorandom data generators for cryptographic purposes are not studied (yet) – the only exception is an analysis indicating that AES (e.g., in OFB or counter mode) is suitable for pseudorandom data generation [HW03].

Several important practical problems related to the process of statistical testing were identified [NIS00]. Firstly, no clear evidence about the independence of particular tests was given, thus nobody knows how much of existing tests should be used. Secondly, no analysis regarding the necessary length of tested sequence

(and number of their subsequences) exists. Thirdly, no analysis regarding the number of repetition of the test procedure (if a clear conclusion cannot be done) exists.

2.3.1 Performed experiments

We have dealt with random and pseudorandom data generators of selected cryptographic smartcards and hardware security modules (HSMs) during our smartcard security research for the Czech National Security Authority. Due to specific conditions of these research grants we are allowed to describe performed experiments only at the conceptual level and the details (including experiment settings and results) cannot be included in this thesis.

The state of the current research in the field of hardware true random data generators for single chip devices can be found in [CKM+04]. We performed several practical experiments focused on black-box testing of true random data generators in Infineon chips and Eracom HSMs. Statistical tests of sequences generated by devices in standard operating environment (normal temperature, etc.) are described in [CHK+05, BCK+06]. Several 50 MB sequences and in the case of HSMs also 500 MB sequences of truly random data were always generated and statistically tested at the confidence level 0.01 (longer sequences also at confidence level 0.001). Our testing utilized both NIST and DIEHARD test batteries and some tests from TESTU01 and RaBiGeTe test batteries. Note that we adopted and integrated into DIEHARD test battery the mechanism of empirical results interpretation from the NIST test battery (i.e., the examination of the proportion of sequences that pass a statistical test and testing the uniformity of resulting p-values).

In [KKK+07] we describe our experiments focused on influencing of chips integrated in cryptographic smartcards. Several attack scenarios with the goal to disable/influence a true random data generator or to change the content of SRAM or EEPROM memory were proposed and also realized. The methods of influencing the chip involve the change of temperature (freezing by a liquid nitrogen – figure 2.1) and particle radiation (emitting alpha and beta particles from open/unsealed radiation source – figure 2.2). The sequences of truly random data generated during influencing of chip were a subject of our statistical testing. Moreover, we verified the integrity of data in allocated memory arrays to detect possible fault injection.

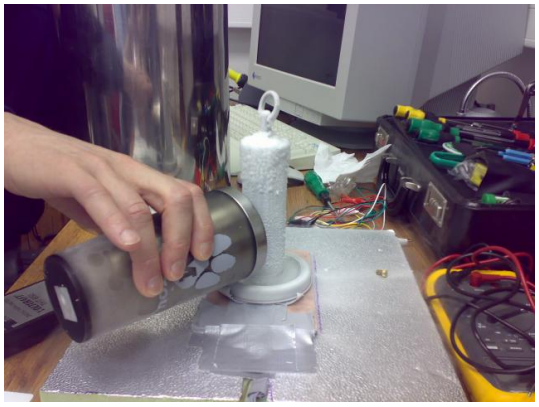

Figure 2.1: Freezing a smartcard chip by liquid nitrogen.

Figure 2.2: Unsealed sources emitting alpha and beta particles.

During the experiments with liquid nitrogen we also practically verified that some chips have a protection against such influencing – a sensor that is able to temporally stop the chip functionality when the temperature reaches a specific threshold.

As we mentioned above, the secure integration of ANSI X9.17/X9.31 and FIPS 186 pseudorandom data generators into JavaCards is described in [CHK+05]. Several sequences generated by those pseudorandom data generators were also a subject of our statistical testing.

2.4 Distributed generation

This section is based on [KSM08] and discusses basic security aspects of distributed random data generation in potentially hostile environments. The goal is to outline and discuss a distributed approach, which comes to question in the case of attacker being able to target one or several mobile devices. We define communication paths and attacker models instead of providing technical details of local generation that are discussed in section 2.2. This part also includes a discussion of several issues of such distributed approach.

2.4.1 Motivation

Since mobile devices typically use a wireless channel for communication, the security of transmitted data plays a very important role for many applications –

consider, e.g., mobile banking. High-quality and unpredictable cryptographic keys, padding values, or per-message secrets are critical to securing communication by modern cryptographic techniques. Their generation thus requires a good generator of truly random and pseudorandom data.

The security of local generation of truly random (and also pseudorandom) data relies primarily on the quality of used sources of randomness. The mobile phones typically provide some good sources of randomness – e.g., noise present in audio and video input – that we analyze in [KŠM07] and describe also in chapter 3.

However, the possibility to influence and predict such sources of randomness implies a possibility to predict generated data. Modern *hybrid pseudorandom data generators* periodically use truly random data also during the whole generation process – this improves the generator security by increasing resistance against state compromise attacks at the expense of higher demands on truly random data.

Since mobile devices or their sources of randomness can be under attack – consider, e.g., malware or influencing video input by changing ambient light intensity – we can involve several cooperating mobile devices in the generation process. These devices can perform generation at the beginning of (or during ongoing) communication with other devices. This distributed approach can support better random or pseudorandom data generation in case of attackers being able to target only some (but not all) of the mobile devices.

Local generation, from the attacker point of view, is obviously strongly dependent on the attacker possibilities to control a mobile device and to influence or predict used sources of randomness. The situation is quite different when we consider distributed random data generation. In this case, the attacker possibilities depend also on the communication model and methods for secure gathering and post-processing remotely generated data.

The following parts are organized as follows: In the next section, we define attacker models for local random data generation. Section 2.4.3 focuses on definition of basic communication paths and describes several problems that we encountered. Section 2.4.4 sketches possible mechanisms for gathering random data in hostile environments and discusses problems that should be considered.

2.4.2 Attacker model for standalone mobile devices

Recall that random data for cryptography purposes must have good statistical properties and must be unpredictable. These two conditions are jointly satisfied only if the truly random data is generated with utilization of a good physical source of randomness and post-processed by a cryptographic pseudorandom data generator.

Such sources can be under attack (i.e., influenced by an attacker) and this can result in non-uniform random data or in completely predictable (or even worse constant) data that is not random. Utilizing several sources of randomness and combining their outputs is a common practice to avoid a prediction of a generator output in the case when the attacker influences some (but not all) sources of randomness. Better statistical quality and faster generation is accomplished by utilizing digital post-processing, e.g., by cryptographic pseudorandom data generator. Hybrid generators then also allow to increase the inner state entropy by periodical reseeding and continual accumulation (pooling) of truly random data.

We often assume that the attacker has no access to the generating device – in this case the post-processing can hide many statistical defects or even influence the source of randomness that results in generation of constant data without entropy. The situation is more difficult if an attacker somehow obtains an inner state of pseudorandom data generator – e.g., due a design flaw, implementation error, or by readout of the memory content. In this case, the attacker can also easily predict all pseudorandom data before next reseeding. Potential simultaneous influence of the randomness source then allows to predict pseudorandom outputs even after reseeding.

Currently, mobile phones that want to access European global system for mobile communications (GSM) network are typically equipped by the subscriber identity module (SIM). It is a smartcard that provides secure storage, secure computational environment, and it also contains physical truly random generator – typically based on sampling of several free running oscillators. However, SIM cards are under control of the mobile network operator and there are very limited possibilities of their usage by common users – often restricted only to the secure storage of contacts or short text messages.

The future technical progress may result in mobile phones with a second smartcard that will be under full user control. In this case, all cryptographical operations, including generation of random data, could be performed inside the card

similarly as in classical SIM Toolkit applications, and the external sources of randomness can only serve as an additional (but non-reliable) input.

The main problem of mobile devices is that their computational environment is not secure. Such devices can contain malware (malicious software as viruses, Trojan horses, etc.) and all generated random data could be easily replaced by non-random data before they reach the appropriate application (located in mobile device or inside SIM card). One possibility to prevent unwanted malicious software or even firmware installation lies in the introduction of a trusted platform module (TPM).

All this implies that the real attacker model for standalone mobile devices is strongly dependent on the attacker possibilities to control the device or used sources of randomness. We define four classes of attacker for standalone mobile devices:

Type I (weak outsider) – the attacker had temporary read access to the mobile device and knows the internal state of device – including pool – before beginning of the generation. He has no possibility to access the mobile device again, but he has access to the information about the environment of the victim (he can stay in the proximity of victim, he can record audio/video of the victim to the camera, etc.) and he has also a limited capability to influence this environment (e.g., disturbing the signal, overexposure the lens of the digital camera, etc.).

Type II (strong outsider) – the attacker has in addition to the weak outsider almost all information about victim's environment and he is capable of fully influencing this environment. The term "almost" reflects uncertainty arising from interactions between user, device, and environment (several physical effects, errors in measurement, etc.).

Type III (weak insider) – extends the capabilities of previous strong outsider by adding a full control over the mobile network operator SIM card with the possibility of remote reading from or writing to the SIM card.

Type IV (strong insider) – in addition to previous scenario, the attacker has a temporal write access to the mobile device. Therefore, he can also compromise the firmware/software of the mobile phone (e.g., by rewriting flash, installing malware) and hence he has a full control over the phone (including interprocess communication) with a possibility of remote access to the mobile device again.

Clearly, a digital post-processing by cryptographic pseudorandom data generator causes that an attacker always needs to know the device internal state. However, as we discussed in the introduction, a careful user should always expect that sources of randomness in a standalone generating devices can be under attack. A successful attack (performed by weak or strong outsider) and the knowledge of internal state always results in a predictable data output.

It is extremely difficult to guarantee that the external source of randomness is not under an ongoing attack. Several online statistical tests can be performed, but they can probabilistically detect only a basic (and limited) set of statistical defects. An additional and more convenient way how to secure a pseudorandom output is to prevent attacker from copying the internal state by utilizing secure storage inside the SIM card. This works only until the mobile network operator starts to act as/with an attacker, being capable to read/write content of SIM card that is in the operator's ownership.

Preventing an attacker with full access to the device is the most difficult task that can be meaningfully accomplished only by introducing a trusted platform module (TPM). Since we want to keep our discussions realistic, we expect a type four attacker being able to target several (but not all) remote devices. However, we assume the local devices (including SIM card) always behave correctly. This guarantees (in terms of probability) that a trustworthy local device obtains at least some random data from remote parties and thus is resistant against the first three types of attackers. The clarification of this strict assumption is described below.

2.4.3 Communication model

Since we are interested in random data generation in mobile environments, we will distinguish between a *consuming mobile device* that requests random data and a *generating mobile device* that (e.g., upon a request) generates random data. Sometimes we consider a *generation computer* located in the Internet or GSM network that (e.g., upon a request) also generates random data.

In the basic scenario we expect that the owner of a trustworthy consuming device always selects trusted remote users to generate random data. A particular generating device replies with a message that includes requested random data and declaration about the amount of entropy in this data. Based on the user reputation the consuming device makes a decision about the amount of claimed

entropy of the obtained random data. This reputation can be predefined by the consuming device owner and we call it static reputation.

Unfortunately, there is no way how to assess the statistical quality of the obtained sample of random data and how to validate the amount of entropy in such data. Even a device of a user with a good reputation could become the victim of the malicious code (viruses, Trojan horse, etc.) that can produce only pseudorandom data with no entropy. This prevents also using all dynamic reputation systems that automatically recalculate reputation. The only meaningful solution of this problem is using random data from several devices where at least one device is expected to be honest and the communication between these devices is well secured.

In this section we focus on the communication issues and we restrict ourselves to the situation when both communicating devices behave correctly.

2.4.3.1 Communication paths

For a precise definition of attacker models in the distributed environment, it is essential to know the communication model that includes network topology, used security mechanisms and their fundamental vulnerabilities. All these communication properties are briefly discussed in appendix A of [KSM08] and a detailed description of these issues can be found, e.g., in [EVB01] or [Xia07].

We define several path types that are used in definitions of attacker models. The device at the beginning of path is always a consuming mobile device; the device at the end of the path is the generating mobile device or computer. The path can also lead through the Internet and the first computer that provides Internet connection to that devices on end-points of the path is denoted as a *gateway*.

Type 1 – the simplest local path can be established between two mobile devices or a between mobile device and a computer. These paths can be point-to-point (via, e.g., IR or USB interface) or point-to-multipoint (via, e.g., Bluetooth or WiFi). Paths between two mobile devices can lead over one or several intermediate devices. For example, in the case of Bluetooth the path can lead over one superior/master device. Another example is a large WiFi network where two mobile devices can be connected to different access points. Therefore, communication path between these mobile devices can lead through several access points.

Type 2 – the GSM communication path established between two mobile devices can lead over several GSM networks. These paths can be created by standard GSM technologies (e.g., SMS or MMS). Moreover, the mobile network operator has a capability to improve his network by additional special GSM services that can extend network functionality (e.g., servers that provide random data on demand). In this case the communication is established between mobile device and such GSM service server.

Type 3 – the mobile device can communicate with other mobile devices or computers through the Internet. The access to the Internet can be established through a gateway, which could be a personal computer or a wireless access point. The path between the consuming mobile device and the gateway (and between the gateway and the generating mobile device) is covered by the simplest paths of type 1. Internet or similar packet oriented network (based on TCP/IP, X.25, etc.) is either used between gateways or between a gateway and the generation computer.

Type 4 – hybrid paths through the Internet where one or two gateways on the path are in fact GPRS support nodes of different GSM networks. The path between the consuming mobile device and the gateway (and between the gateway and the generating mobile device) is covered by the paths of type 1 and type 2. For example, the consumer mobile device can request random data from the generating computer or the generating mobile device connected to the Internet through the gateway – another computer or access point. Moreover, the gateway for the generating mobile device can also be a GPRS support node.

Note that leased lines can be used to interconnect different GSM networks. The description of mechanisms that secure data flow in such lines is not publicly available. The lack of this information implies that the leased lines should not be trusted even when the mobile network operator (or its employees) behaves honestly.

2.4.3.2 Attacker model for distributed systems

Since the attacker is able to eavesdrop some communication links, they have to be secured in terms of authenticity, confidentiality and integrity. In order to design secure systems, which support distributed random data generation, we should take into consideration the communication paths and the corresponding attacker

models. We defined four different attacker types according to the communication paths described above – for details see appendix B of [KSM08]. However, as we are not able to detect potential modification or observation of the transferred data, we must assume that each particular communication link in the path must be secured. This can be done either by the owner of the infrastructure (e.g., GSM network operator) or by the end-point mobile device (by means of end-to-end security). This implies that our attacker models degrade and all types of attackers can be prevented by securing the whole communication path.

In the next section, we consider cryptographic protocols that allow two or more distributed parties to establish a shared secret key and contribute to the process of its generation. The authenticated protocols are designed to work in hostile environments and to provide end-to-end security, and can be used to prevent attacker types I–III.

2.4.4 Gathering of random data in hostile environments

As was mentioned above, the local random data generation can be performed in a hostile environment and the mobile devices and their sources of randomness can be under ongoing attacks. Therefore, we suggest to involve more parties (mobile devices) in the process of random data generation. Such mobile devices can be considered as independent (remote) sources of randomness[5]. In general, the more generating mobile devices are used, the less probability to attack all of them is – due to usage of different devices, environment, and to certain point also a communication paths.

The consuming device can obtain random data from generating devices either per explicit request or as a secondary product of ongoing communication (e.g., audio/video conference). In the former case the user sends the request for random data to one or several generating mobile devices. In the latter case the random data is transferred during communication. This functionality can be supported either by a mobile network operator service or by a third party application, which in turn uses existing network services (e.g., SMS, MMS).

In the distributed environment we can also distinguish between two methods of obtaining the random data per explicit request – direct or indirect. In both methods the consuming mobile device requests another device to provide random data. Direct method means that the response is sent directly back to the con-

[5]The remote mobile device can provide both raw and post-processed random data.

suming device. Indirect method, on the other hand, means that the response is sent through other mobile devices (can be predefined by user), which add their own random data. The last device sends the accumulated random data back to the consuming device. It is an open question whether such method brings some significant advantages (e.g., for ad-hoc networks) and so would be more effective than the direct one.

Technology improvement (e.g, 3G/4G networks) introduces the possibility of audio/video conferences. To assure confidentiality in such conferences, the participants typically have to agree upon a shared secret key, which is used to encrypt the ongoing communication. This scenario requires all participants to contribute the random data to the shared key. This fact benefits particularly the consuming devices which are locally influenced by the attacker (type I or II). The shared key could be treated as a random data generated in the distribution manner.

The disadvantage of this method is that all participants share the same random data that are predictable (with no entropy) for an adversary inside the group. Therefore, we propose to keep a distinct pool of random data for each communication group. The random data stored in a pool associated with a particular group can be used to secure communication only within this group.

2.4.4.1 Distributed contribution protocols

In this section, we discuss (multi-party) cryptographic protocols that can be used for distributed random data generation. These protocols enable each user to accumulate random data from multiple participants – therefore, we call them *distributed contribution protocols*.

We consider the group key agreement (GKA) protocols as possible candidates that can be utilized for distributed generation of random data. A brief description of several basic GKA protocols can be found in [BM03]. These protocols are typically based on the Diffie-Hellman key establishment and work in several rounds. Each participant of protocol generates new (or modifies obtained) Diffie-Hellman value(s) and sends it/them back to the initiator (or to the next member) of the group. The detailed messaging is dependent on the particular logical topology of a protocol, however the modification of exchanged values always involves randomly generated data.

These protocols provide either no authentication or many-to-many/many-to-one authentication (typically based on secret key/password of the remote party). The

many-to-one authentication scheme assumes that all participants authenticate themselves only to one participant (e.g., initiator of the communication) and they have no direct assurance who are the group members. The non-authenticated protocol could be easily transformed to the authenticated one – for example, by the means of digital signatures.

More sophisticated protocol versions are designed to fit a concrete physical topology (e.g., GSM network), but the advantage here is only in more effective messaging or in offloading of complex computations from resources-restricted mobile devices [BCEP04]. Another class – fault-tolerant GKAs – allows to detect parties that do not follow the protocol [Tse05]. However, since the underlying Diffie-Hellman problem is considered as hard, all exchanged values can be always observed by an attacker.

Note that classical authentication protocols often rely on shared secrets that are typically stored inside the device and this fact implies only the device authentication. More sophisticated password-based authentication protocols (e.g., password-based GKA) provide better user authentication, but often also require more random data. Several innovative methods perform user post-authentication by audio-visual means, which requires even less random data than classical authentication. This kind of authentication relies on the ability to recognize the user face/voice and other behavioral characteristics [LP08]. Another scenario utilizes visual checking of exchanged Diffie-Hellman values that can be transformed to usual language words for easier verification [ČH04].

2.4.4.2 Analogy with the chicken-and-egg problem

The distributed approach to random data generation has one significant drawback. As we mentioned above, the transfer of random data must be secured. However, common mechanisms for ensuring authenticity, confidentiality and integrity (but also, e.g., anonymity or information hiding) are based on classical cryptography, which in turn is dependent on random data as secret keys, padding values, etc. This implies a classical chicken-and-egg problem and breaking this circle seems to be impossible – from both information-theoretic and complexity-theoretic points of view.

The main reason is that encryption of high-entropy data behaves similarly as a pseudorandom data generator. The maximal entropy that can be obtained from encrypted data is limited by the entropy of the secret encryption key. We

cannot count on more entropy than the entropy of the encryption key, because the adversary has a possibility to attack the secret key. The solution could be reestablishing a shared key for each request for random data, however, this process requires that all involved parties have a reliable source of random data.

Despite this drawback, we propose to use both raw and post-processed random data obtained from remote parties as an independent additional input for next digital post-processing – e.g, by a hybrid pseudorandom data generator or a non-deterministic randomness extractor. Such secure nondeterministic transformation is the only way how to solve the problem that the whole group shares the same random data, usage of which then has to be restricted to this group. We are aware that both techniques require some amount of truly random data and the entropy outside the group will be restricted to the entropy of that truly random data. The main benefit here is that we obtain more secure and reliable method of generation (regardless of entropy) in the presence of an attacker of types I or II.

As we have mentioned, the direct usage of random data obtained from remote parties can be done only for securing communication within a particular group. Due to the entropy issues, we recommend its usage only for short-term encryption keys or other secrets that are intended for applications where the data to be protected are sensitive only for a limited time period – e.g., daily stock price forecasting or common day-to-day audio/video conferences. Other direct usage without digital post-processing is not recommended. Note that this holds also for local generation in the presence of an attacker of types I or II, especially in the case when we want to use local random data for first key agreement or establishment.

The only way to completely avoid usage of digital post-processing is utilizing the SIM card at least to establish a secure communication channel. In this case is also recommended to use the secure storage of the SIM card to protect seed files and all randomness pools.

2.4.5 Summary

Since a mobile device (or its sources of randomness) could be influenced by an attacker, there will be scenarios where one should consider extending a local random data generation to the distributed one, with several mobile devices involved. We suggested this approach in [KSM08] and discussed some of its issues.

We examined various communication paths that can be used to interconnect mobile devices (e.g., Bluetooth, GSM, Internet, etc.) and looked closely at different attacker types according to the communication paths involved. Further, we discussed multi-party cryptographic protocols that could be used for distributed random data generation. We considered one class in more detail – the group key agreement protocols, where the shared secret key could be treated as random data generated in a distributed manner.

Our aim was to point out distributed random data generation as a possible way to obtain high quality random data. Obviously, even the distributed approach has its advantages and disadvantages. We presented some possible solutions and we hope to encourage a further discussion of this approach.

Chapter 3

The sources of randomness in mobile devices

This chapter deals with issues related to the generation of truly random data in mobile computing environments. The main goal is to examine randomness sources available in current mobile phones or other mobile devices. We identify potential sources of randomness and perform an analysis focused on the camera and the microphone noise as promising sources of randomness. Moreover, we analyze the quality of these sources of randomness by statistical testing and perform estimation of entropy (in a given time period) in the generated data.

Our work covers the possibilities of random data generation from both external and internal environmental characteristics. The examined external sources are audio and video streams that can be captured by a mobile device. With respect to the internal sources, we investigated the information accessible to each user through the application programming interfaces (API) of mobile devices like the actual battery charge level and other accessible system statistics.

Identified sources of randomness are tested on the Nokia N73 with the Symbian OS and E-Ten X500 and M700 with Windows Mobile OS. Significant part of this chapter is based on [KŠM07, KMŽ09], but some issues were discussed at the conceptual level also in [KŠMS07].

3.1 Specifics of the 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[1].

On the other hand, mobility and small physical size of devices bring 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 and future inner state of the generator and fast recovery of its entropy level (after a time-limited compromise). Other disadvantages can be low performance (CPU frequency of best smartphones is roughly 300 MHz) and restricted random access memory size (in the order of tens of MBs).

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. It can be expected that the user will be more tolerant to the delay in such special cases (caused by his involvement in the generating process) than in the case of normal voice encryption (where is necessary the short response time). Application requirements on random or pseudorandom data were discussed in section 2.1.2.

There are several good potential sources of randomness in the mobile devices. All of them can be used for generation of truly random data and categorized as follows:

1. The external environment – sources located in the proximity of a user. They can be used to generate truly random data – their characteristics (samples) can be obtained by the different mobile phone input devices.

[1] We are aware that in general purpose computers use of audio/video data for generating random data is not a novel idea (see, e.g., [ECS94]), but mobile devices with embedded cameras and microphones have a clear advantage with respect to fitness and practical applications of such techniques.

Typical examples of these characteristics can be the location (Location API JSR-179), digital image, audio/video record (Mobile media API JSR-135), or strength of the incoming signal in the range of base transceiver stations.

2. The internal environment – sources inside the mobile device that are inaccessible to a remote attacker. An example is the execution of computational operations, memory access time, or true random data generator on the SIM card (in European countries often cryptographic chip card, such as Gemplus, Philips, Schlumberger). Clearly, the processes performed inside the phone will be always (to a certain level) dependent on the external environment. We will expect that this dependency (that can lead to influencing of this kind of sources) is very complex and cannot be observed by the attacker or even misused. Typical example is the noise originated in the microphone, CCD chip or A/D converter. However, it is extremely important to make precise categorization of particular sources. For example, the microphone in hands-free sets should not be considered as an internal source. A poorly performed Bluetooth pairing process can lead to the eavesdropping of digitalized voice communication including the noise of such microphone.

3. Interaction with the surrounding environment – sources that are highly influenced by the feedback from the external environment. Typical example is sound reverberation, the periodical changes in battery level (depends also on temperature or strength of signal), or fluctuations in the incoming signals.

Important characteristics of all sources are: availability in different environments, unpredictability and uninfluenceability for the attacker. The characteristic from the internal environment (e.g., some kind of noise) cannot be definitely influenced by an attacker as much as in other cases (e.g., external signals or images).

The opportunities of an attacker to influence the generator are discussed in section 2.4.2 and more detailed discussion of possible sources of randomness can be found in [KŠMS07].

3.2 Analysis of selected sources of randomness

This section deals with practical experiments performed on two smartphones Nokia N73 (figure 3.1) with the Symbian OS and two similar PDA phones E-Ten

X500 (figure 3.2) and E-Ten M700 (figure 3.3) 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.

Figure 3.1: The mobile smartphone Nokia N73.

Figure 3.2: The mobile device E-Ten X500.

Figure 3.3: The mobile device E-Ten M700.

Due to the API restrictions in the Symbian OS (our primary platform), 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 with fluctuations typically only in the two least significant bits).

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 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.

3.2.1 Theoretical entropy estimation

Recall that the basic measure for randomness is in information theory often called uncertainty or entropy and is typically defined [Sha48] as:

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

where the sample x is drawn from random distribution X with probability $P_X(x)$. The logarithm base typically correspondents to the unit of measured information

– in information theory base 2 is often used and that implies that the unit will be bits. This entropy measure is often referred as Shannon entropy or alternatively information entropy.

Unfortunately, Shannon entropy may be inappropriate for our purposes, since it is in fact only average case entropy. We cannot make any assumptions about distributions formed by our sources of randomness. The problem is that an attacker can (at least theoretically) force the source of randomness to produce the most probable values that contain minimum entropy. To cope with this situation the min-entropy measuring the worst case entropy is often used (especially in the theory of randomness extractors).

It is defined as:

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

where the sample x is drawn from random distribution X with probability $P_X(x)$. It can be easily seen that min-entropy is always less or equal than Shannon entropy (the tight example is for uniform distribution). These two entropy measures are special cases of generalised so-called Rényi entropy [Rén60].

However, even this measure may provide biased results. As we stated above, we cannot make any assumptions about distributions formed by our sources of randomness. This fact implies that even the attacker of types I or II could be also capable to completely change the probability distribution.

The ideal solution to this problem is to remove all statistical defects and dependencies (that implies also uniformity of resulting statistical distribution) and then perform entropy estimation or randomness (entropy) extraction. As far as we know, this task is impossible for two main reasons. Firstly, infinite statistical defects exist and removing one or several obvious defects could induce new (hidden) defects. Secondly, current randomness extractors have always strong assumptions regarding statistical distribution of the input data.

The best approach we can do in practice would be to use *attacker-aware methodology*. It means that we perform all our experiments with mobile devices during worsened conditions – e.g., in several different external conditions (temperature, ambient light, acoustic noise etc.) – that should simulate an ongoing attack. All results and entropy estimations then correspond to the worst values (i.e., lower measured entropy values).

3.2.2 Microphone

We wanted to evaluate all microphones in the worst possible conditions, therefore the first idea in our experiment 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 arbitrary music sample recording. Since the recorded noise is also present in music or other audio samples, the worst conditions for the microphone arise from recording of noise in an extremely quiet environment (which was in our case a closed quiet room in the night).

An embedded or hands-free microphone is used as a voice input device in mobile devices. Almost all commonly used microphones are typically based on an oscillating membrane and some mechanisms that transform the oscillation to the voltage representing particular signal elements. Supported sampling frequency, modulation method, and number of bits used for representing the value of one sample are the most important parameters of such devices. The Nokia N73 smartphone and both E-Ten devices use 16-bit pulse coded modulation (a signed PCM) at the frequency 8000 Hz for sampling a sound wave – the data throughput is thus 16 000 B/s.

We analyzed several microphones that have obviously different characteristic, e.g., due to different solidity of membrane or other manufacturing differences. Our goal was to estimate the amount of entropy in the sound signal captured by the microphone – we focused mainly on measuring min-entropy of the noise originated in the microphone.

3.2.2.1 Smartphone Nokia N73

In the case of Nokia N73 we restricted ourselves only to the small number of 204 800 samples that corresponds to 25.6 seconds of sound due to memory restrictions of the inspected device. 204 800 captured noise samples were used to form a histogram of values.

We used the fast Fourier transform (FFT) algorithm to compute a discrete Fourier transform (DFT) for analyzing the basic frequency components present in the noise (ideal noise is expected to have all frequencies uniformly present). We performed this analysis of the embedded Nokia N73 microphone on the sound

sample of both recorded music (figure 3.4) and noise (figure 3.5). Moreover, we analyzed also the hands-free microphone on the sound sample of noise (figure 3.6). We are interested only in the spectrum of frequencies between 0 and 4000 Hz due to the Nyquist theorem (as we are sampling at 8000 Hz). The results show that there are significant differences in the observed frequency spectrums.

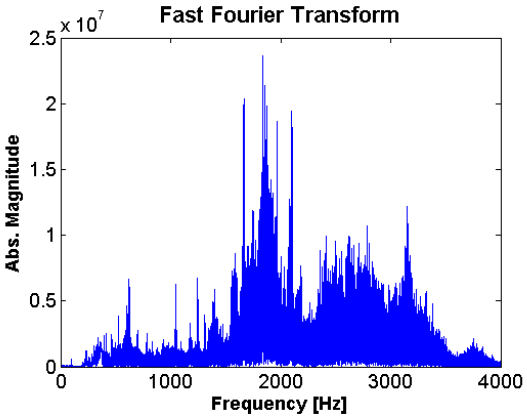

Figure 3.4: Music sample (Nokia N73 embedded microphone).

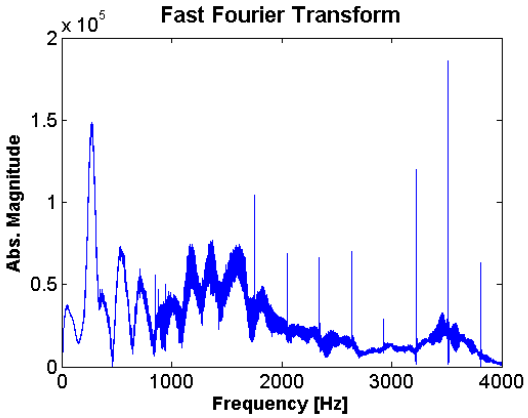

Figure 3.5: Noise sample (Nokia N73 embedded microphone).

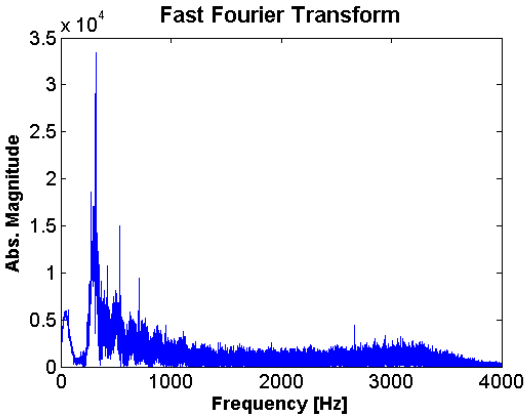

Figure 3.6: Noise sample (Nokia N73 hands-free microphone).

Finally, we analyzed histograms of all these recordings. They have approximately normal distribution and the actual minimum/maximum values are –32764/32767, –14220/8856, and –347/574, respectively. However, especially in the case of noise, these numbers are strongly influenced by the sharp peak at the beginning of each recording trace – caused by the device turn on. More accurate limit values –12/13 and –9/8 are obtained when this peak is removed. Hence, we can make a very preliminary estimation that at most $log_2 18 = 4.2$ bits of entropy can be presented

in each sampled value. Of course, this is only the upper bound of the maximum possible entropy.

Focusing only on the more sensitive hands-free microphone, the estimations are 2.9 bits of entropy according to the Shannon formula and 0.5 bits of entropy according to the min-entropy formula. These estimations (and especially probabilities used in those formulas) were also calculated from histograms with the assumption of full independency within the samples.

3.2.2.2 PDA phones E-Ten X500 and M700

Microphones embedded in the E-Ten devices are more sensitive and thus also capable to record significantly more noise than the microphones used in the Nokia N73. Basic frequency components are present in the music (figure 3.7) and noise (figures 3.8 and 3.9). Several harmonic frequencies (narrow peaks in a spectrogram) were revealed in the spectrum of noise captured by E-Ten devices and the best result (i.e., smoothest frequency spectrum) belongs definitely to the Nokia N73 hands-free microphone with only few harmonic frequencies (for comparison see figure 3.6).

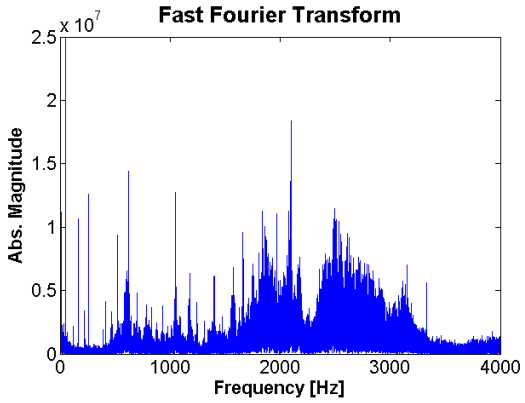

Figure 3.7: Music sample (E-Ten M700 embedded microphone).

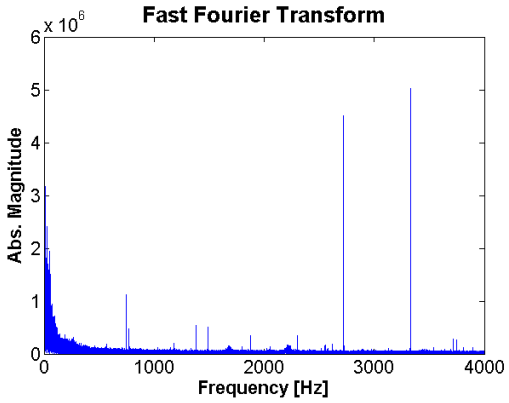

Figure 3.8: Noise sample (E-Ten M700 embedded microphone).

The histograms of all these recordings have also approximately normal distribution and the actual minimum/maximum values are –32258/32767, –1958/1633, and –806/716, respectively. There are no peaks at the beginning of each recording trace. Hence, we can make a very preliminary estimation that at most 11 bits of entropy (given by encoding) can be extracted from each sampled value. Of course, this is (again) only the upper bound of the maximum possible entropy.

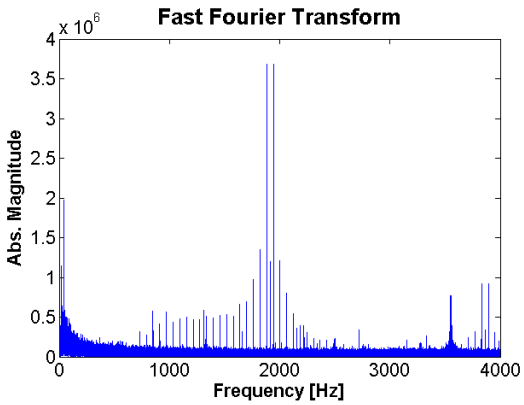

Figure 3.9: Noise sample (E-Ten X500 embedded microphone).

The precise estimations are 10.124 and 9.365 bits of an entropy according to Shannon formula and 0.016 and 0.023 bits of entropy according to min-entropy formula for M700 and X500, respectively.

3.2.2.3 Correlation tests

We performed also several correlation tests (described in the camera section below) 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 to lower the estimated entropy at least 5 times with respect to the amount calculated for each sample value.

3.2.3 Digital camera

Digital optical input devices (such as photocameras, microscopes, or scanners) can be based on several different silicon *optical sensors* (such as CCD, CMOS, EMCCD, and ICCD) [Ken06]. Digital cameras for mobile devices typically use CCD (e.g., Sony-Ericsson S700i) or CMOS (e.g., Nokia N* series) sensors. These sensors use an array of semiconductor photo-sensors to transfer an accumulated electric charge to a voltage. Low-quality optical sensors that are often used in mobile devices have generally higher noise presence than sensors in high-end cameras. All these optical sensors could be also influenced by thermal noise and they have specific problems with vignetting, blooming, sensitivity to some colors,

etc. Some of these problems are solved by the manufactures by purely software means that are often kept secret.

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 [Ken06]). 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.

A significant part of current mobile devices (cell phones, smart phones and PDAs) is equipped with a built-in camera that can be used for random data gathering. For camera we suggest that a output data should be extracted from an optical sensor noise during "view finding" rather than from a high-resolution picture of the surrounding environment. The data output from view finding is more suitable source of randomness than a high-resolution snapshot output for two main reasons:

1. View finding is not post-processed by software noise reduction and compression algorithms.

2. Data acquisition is much faster – commonly between 10–15 frames per second – and has more suitable size for the additional manipulation. A single frame (180×240 pixels for Nokia N73) can be stored and processed in the RAM memory ($\sim$130 KB), which is unlikely for 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 (described below) confirmed the presence of low level post-processing (details are not provided by the manufactures, similarly as for the downsampling algorithm) as, for example, correction of the light intensity towards the border of lens or automatic ISO level correction. All these proprietary techniques make the entropy estimation much harder.

Following the attacker-aware methodology, the worst conditions for these sensors involve recording of a static image (white or black background). The output from the digital camera is often available even when the camera cover is closed or the

lens is covered with an object. This is both convenient and useful – it serves as an important defense against an active attacker that illuminates the sensors and forces them to produce biased or constant values.

Direct illumination of the Nokia N73 CCD sensor by a halogen lamp is depicted in figure 3.10. In this case the central area of a sensor is forced to produce maximum constant values (225) and all all possible entropy obtainable from (not only white) noise is effectively removed. A similar effect can be also caused by a strong direct sunlight.

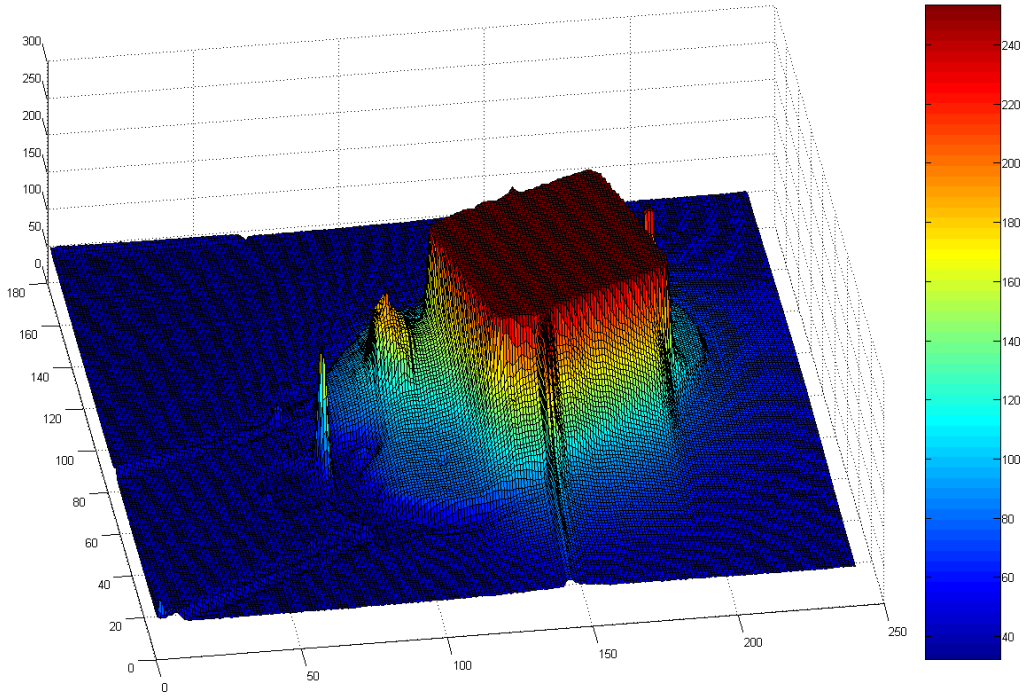

Figure 3.10: A visualized overexposure caused by a halogen lamp. X- and Y-axes indicate resolution (240 × 180), and Z-axis indicates the pixel value (0–255) of red color component.

To successfully mount such attack, the attacker must be able to overexpose almost the whole area of the camera chip. Degradation during random data generation is prevented if the lens are equipped with a closable cover or are shielded by a finger. This assumption is reasonable when the device is controlled by the user and an attacker can only manipulate the surrounding environment.

At the application level the random data gathering mechanism should be able to test the output stream for overexposed values or other simple defects and lower the estimated amount of gathered entropy in real-time.

3.2.3.1 Smartphone Nokia N73

In this section, we would like to estimate the amount of entropy extractable from a digital camera output of smartphone Nokia N73. We proceed in accordance with our attacker-aware methodology as follows:

1. We developed a custom application for storing large amount of frames produced by a mobile device camera into a removable memory card. Systematic defects of the input due to post-processing were examined using visualization and statistical tools.
2. Noise dependency on the surrounding temperature was experimentally measured and evaluated.
3. The correlation between neighbor pixels, pixels in the same row and column was computed, as well as autocorrelation and fast Fourier transformation of values from a single pixel in time (subsequent frames).
4. The distribution of values for separate color components was computed for the temperature with the least noise (5 ℃) and entropy was estimated based on Shannon entropy and min-entropy formulas.
5. A simple deterministic technique for entropy extraction (with least possible post-processing) was implemented and we tested the resulting binary stream by NIST test battery to evaluate its statistical properties.

The systematic defects introduced into camera frames due to post-processing and optical sensor technology (like row-dependent readouts) are clearly visible in figure 3.11.

There are hot pixels (permanently glowing pixels) around borders, significant rips in the rows, centered circle rips and significantly different intensity towards center of the frame. Especially, the blue color component shows visible row-dependent rips (caused by the readout technology) and the red color component has significantly increased value towards chip borders. This effect is caused by camera post-processing (out of our control) to balance light drop due to different lens mass towards borders. Such an effect may lead to a suspicion that some pixels are systematically correlated.

However, when values for each single pixel are normalized by subtracting mean of the pixel in a longer time period, most systematic effects vanish. The color

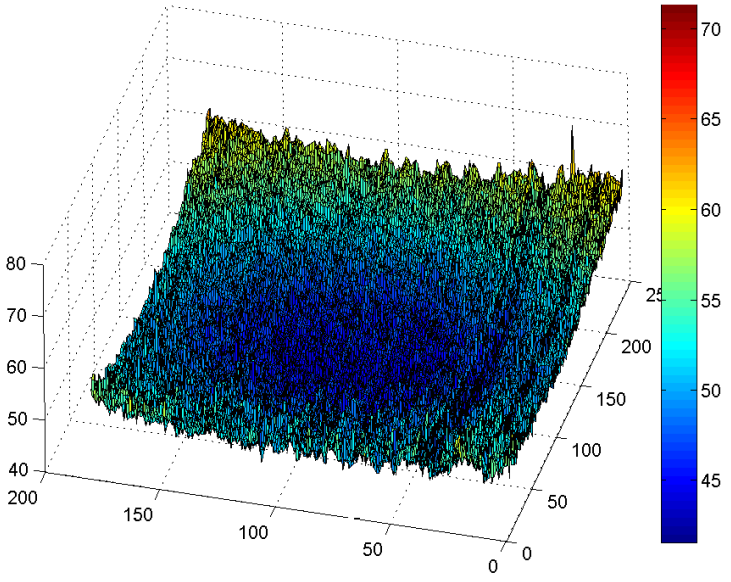

Figure 3.11: The average value of the blue color component over the whole camera's frame with closed lense cover. X- and Y-axes indicate resolution (180×240), and Z-axis indicates average pixel value.

histograms of normalized samples taken over 2000 subsequent frames are depicted in figure 3.12 (histograms are centered to 0). All colors exhibits Gauss-like distribution, blue color providing more entropy (in Shannon sense) than other color components. The presented frames were taken in a temperature around 5 ℃.

A well documented property of the optical sensors is a noise dependency on the temperature – the noise component should be reduced by lower temperatures (e.g., cooling of chip by liquid nitrogen). We tested the noise presence (with closed lens plastic cover) in temperatures of 5 ℃ and 45 ℃ (more precise measurement of optical sensor is not possible without depackaging). The differences between temperatures were detectable, but negligible[2] and we can expect slightly decreased (but still comparable) amount of entropy for the lower temperatures. We used noise obtained from measurements in 5 ℃ for entropy estimation. In the common camera devices the raw data from the optical sensor are not directly accessible to the user – they first go through a few correction steps, which are usually incorporated into the firmware of the optical sensor. Such corrections take care of some of the different systematic effects, which are partially inherent to the optical sensor working principle and partially occur in fabrication tolerances of the manufacturing process.

[2]Difference in values probability only in order of 1/1000.

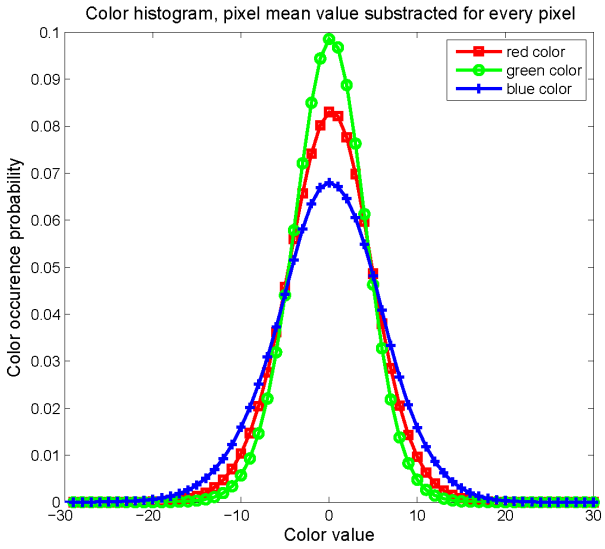

Figure 3.12: The intensity histogram for each color component after a normalization process (removing of the pixel post-processing effects).

For an overall entropy estimation, we have to know the number of independent (uncorrelated) pixels in one frame. Moreover, we also need to know whether values of a single pixel are independent between two or more subsequent frames or which frames have to be discarded to obtain uncorrelated values (e.g., pixel value from every 3rd frame). Finally, we must estimate the expected entropy provided by a such single independent pixel.

A cross-correlation function (Matlab *corrcoef*) was used to verify whether neighboring pixels and pixels in the same row and column are independent. No statistically significant correlation was found on significance level set to 0.01[3]. We generated streams of a binary data from red, green and blue color components and tested those streams with the NIST battery for a further verification of results obtained from cross-correlation. See section 3.2.4 for details.

The camera view finding mode on Nokia N73 provides us with 12 frames per second and thus 12 different values per single pixel (10–15 frames for other cameras). The key question for entropy estimation is the independency of consequent values of that pixel. The auto-correlation tests were performed with the vector containing the values taken in time from a single pixel. Statistically significant

[3]This result means that the number of tested vectors with a correlation coefficient lower than 0.01 (strong correlation) is not significantly higher than the number of correlated ones between truly random vectors. It means that only around 10 vectors from 1000 are expected to be correlated for this significance level.

deviations from the characteristics of white noise (where significant correlation value is present only for $lag = 0$ and can be omitted otherwise) were not detected. This implies that the consequent frames should be independent.

All view finding pixels (180×240) were separately tested within a sequence of 2100 frames. The fast Fourier transformation (FFT) was applied to the same vector (values of a single pixel in time) to detect dominant frequencies. FFT provided an almost uniform output (no dominant frequency) for all tested pixels and thus confirmed independence of values between frames from an auto-correlation test. Note that as view finding provides 12 frames per second, only frequencies between 0 to 6 Hz can be tested by FFT due to the Nyquest theorem.

The final step is to compute the amount of entropy carried by the noise in a single (independent) pixel. As can be seen in figure 3.12, histograms of all color components follow the common Gauss distribution without significant deviations. Overall entropy of camera view finding source can be computed using a simple formula [bits/s]:

$$E = IndPixelsPerFrame * IndFramesPerSecond * EntropyPerPixel.$$

An entropy estimation according to Shannon and min-entropy formulas for a single independent pixel (*EntropyPerPixel*) are in table 3.1. Note, that extrapolation to whole input (180 × 240 pixels/frame, 12 frames/second) is valid only if the pixels and frames are independent (see 3.2.3.1 for discussion and performed tests).

Color pixel	Shannon entropy [bit]	Min-entropy [bit]
Red	3.920	3.198
Green	4.037	3.328
Blue	4.761	3.928

Table 3.1: An entropy estimation for single color pixels (Nokia N73, 5 ℃).

3.2.3.2 PDA phones E-Ten X500 and M700

E-Ten PDA phones X500/M700 are based on the MS Windows Mobile 5 OS (WM5) that can be easily upgraded also to the MS Windows Mobile 6 OS (WM6). These operating systems support a special native Camera API for a direct camera control. Unfortunately, this low-level API is in turn not supported by the tested

E-Ten devices (regardless of the OS, tested for both WM5 and WM6) and so we used a special video driver capable of capturing and sending the actual screen image over the USB port to a PC.

This approach has its advantage – the data can be retrieved and stored faster than within the Nokia N73 phone and the total amount of the captured data is not limited by the phone flash memory. However, there are also a drawbacks – transferred pictures have only a limited number of possible color values (red and blue color components are limited to 5 bits per pixel; green color component is limited to 6 bits per pixel – while the original resolution was 8 bits per pixel) and can only be retrieved at the rate of one picture per second.

In this section we summarize the results from the same set of our experiments with PDA phones Glofish E-Ten X500 and Glofish E-Ten M700. The first significant difference to N73 is that the camera of E-Ten devices is automatically turned off when the lens is covered. This would imply that a direct capture of noise on a perfectly black background is not easily available.

The noise produced by a camera chip is again dependent on the temperature, similarly as for the N73 device. At 45 ℃ the noise from the chip is so significant that prevents a software switch off (discused above) even when the camera lens is completely covered. Unfortunately, the threshold temperature for this behavior is around 35 ℃ and thus not reachable during the usual operating temperatures. However, it allows us to obtain the noise from a black-only input, similarly to the scenario with the closed cover in the case of N73.

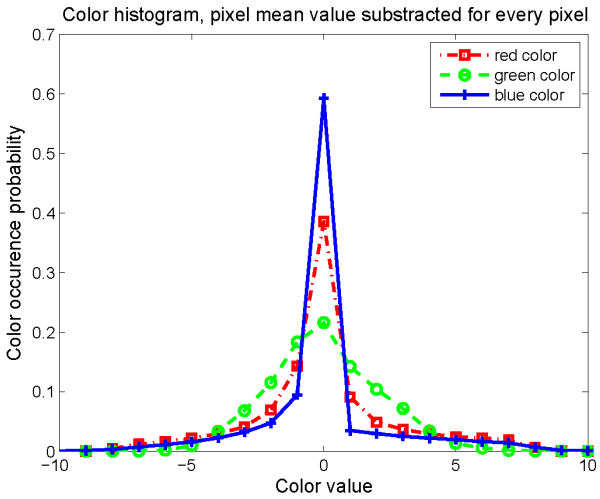

Figure 3.13: E-Ten X500 – probabilities of colors differences from long term pixel mean.

E-Ten X500 and E-Ten M700 also have different camera chips with visually different noise patterns. This was confirmed by the statistical analysis of the intensity histograms for each color component after removing the pixel post-processing effects – see figures 3.13 and 3.14 with histograms of distances from the mean value. The significant peak at zero point on X-axis (mean value) for all colors means that actual color value of the pixel often tends to be equal to the mean value. This can be caused by a limited precision of the captured images.

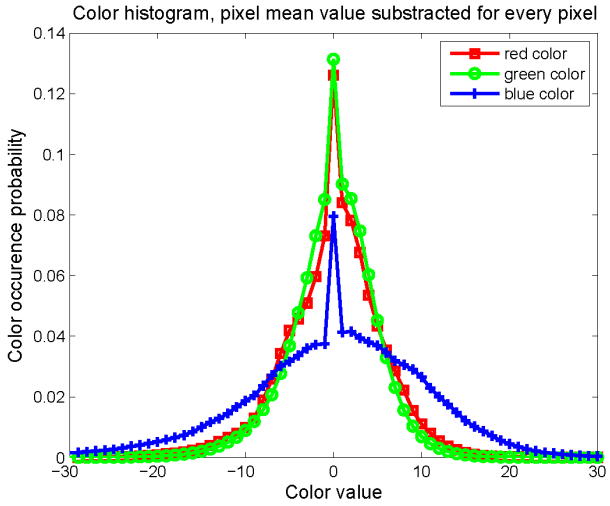

Figure 3.14: E-Ten M700 – probabilities of colors differences from long term pixel mean.

Distributions of the noise (presented above) are taken from the images captured in an 8 ℃ environment as this is the scenario with lowest noise presence within common usage operational conditions.

Both cameras of our E-Ten devices very likely utilize an automatic ISO sensitivity correction – with the same temperature, more noise can be obtained if the light conditions are worsened (decrease of ambient light or an object put close to the lens). As the ISO sensitivity is automatically increased by the camera controller, the noise generated by the chip is amplified and more noise is present in captured images. This hypothesis based on visual observations was also experimentally tested. The camera chip was heated up to 50 ℃ at which the camera is not turned off by the post-processing and series of the images were taken. The temperature was then lowered to 40 ℃ with the lens still covered and next series of images were taken.

At the hardware level the noise level decreases (as expected) with a decreasing temperature. But the resulting noise present in the captured images exhibits an opposite trend – there is more noise for the lower temperature level. This

seemingly contradictory result is caused by the automatic ISO correction. As the real noise level produced by the chip decreases with the decreasing temperature, camera controller (out of our control) detects a smaller range of color values at the actual ISO level and automatically increases the level. Increased ISO level results in an amplification of the noise generated by the chip and thus more noise is propagated into the captured picture.

Estimates of entropy present in samples from cameras of E-Ten X500 and M700 are summarized in tables 3.2 and 3.3.

Color pixel	Shannon entropy [bit]	Min-entropy [bit]
Red	3.085	1.279
Green	3.122	2.189
Blue	2.480	0.755

Table 3.2: An entropy estimation for single color pixels (E-Ten X500, 8 ℃).

Color pixel	Shannon entropy [bit]	Min-entropy [bit]
Red	4.463	2.988
Green	4.258	2.928
Blue	5.376	3.653

Table 3.3: An entropy estimation for single color pixels (E-Ten M700, 8 ℃).

3.2.4 Statistical testing with the NIST battery

As described in the section 3.2.3, we also wanted to verify the statistical properties of binary stream extractable from the view finding input. We extensively tested many sequences extracted from the input by various different deterministic techniques (incorporating only particular colors, pixels, and/or utilizing deterministic simple digital post-processing).

Our goal was to find the easiest way to extract binary sequences that will pass common statistical tests. This should help us to assess how much entropy can be present in input data that we will use for seeding a pseudorandom data generator or randomness extractor (for details see chapter 4). Notice that our techniques are far away from theory of universal non-deterministic randomness extractors (as mentioned in section 2.2.3) that are constructed by a considerably different way (a stimulating discussion can be found in [BST03]).

Recall that the input sequence is in the NIST test suite first divided to some number of subsequences and several statistical tests are applied to these subsequences – each yielding so-called *p-value*. There are two ways of interpreting the results of tests. The first method evaluates the proportion of successfully passed subsequences (successful pass means that the *p-value* is less than selected significance level) and the second evaluates uniformity of *p-values* (using chi-square goodness-of-fit test).

Extraction methods that we used for our statistical testing are described below.

Raw values only – we begin with testing the raw input for concrete nine pixels and their basic colors (in both cases separately) and all used NIST tests always failed. The reason is that there is an obvious non-uniformity of values in particular colors and thus also in the whole pixel (that consists of three basic colors saved one after another). The statistical distribution of its values is depicted on the figure 3.12.

Least significant color bit – we then extract only the least significant bit from each pixel and color (one bit per frame is extracted from the single pixel). All binary strings (for each pixel and its color) constructed this way also failed the tests. All tested sequences constructed from least significant bits (LSB) failed both the proportion of successfully passed subsequences and the uniformity of *p-values*. However, for each pixel/color there was a part (40–60%) of subsequences that passed the runs, frequency and cumulative-sums test. This proportion was still too small to pass the whole test, but these tested sequences have slightly better statistical properties.

Color value combination using XOR – we XORed all three colors together but NIST tests also failed for all nine pixels. We also tried to construct the sequence from bits obtained by XORing all eight bits of each value of color. The results were similar as in the case of using LSB. We conclude that these elementary techniques cannot suppress all statistical defects introduced in sequences constructed from one pixel and its colors and utilizing of more pixels is thus required.

Flip-flop bit extraction – simple but more robust technique that extracts one bit per one pixel color component. Extracted bit is 0 or 1 when actual color value is odd or even. Concrete mapping between the color and bit value is reversed after each processed pixel.

Let us describe the flip-flop extraction technique in more details. Note that the purpose of the flip-flop extraction is not a new extraction technique for real usage, but to design simplest possible technique used only for our statistical testing. This technique then extracts some randomness (entropy) from the input, but does not propagate it through the sequence (confusion in the cryptographic sense) like, e.g., SHA-1 does. Flip-flop is used only during entropy estimation, not in an implementation of a real generator (where, e.g., SHA-1 may be used). For a given frame, all pixels are processed row by row:

1. For every even pixel do: if pixel value *mod 2 = 0*, then set output bit b to 0 otherwise to 1.
2. For every odd pixel do: if pixel value *mod 2 = 0*, then set output bit b to 1 otherwise to 0.
3. Bit b is appended at the end of the bit stream.

Set of the possible values from the range $0-255$ is divided into two groups by this extraction technique. The first group contains only even values and the second group contains only odd values. Both groups should have almost equal sum of probabilities. Values from the first group will result in bit value 0, second group in value 1. Unfortunately, separation into two groups with the same probability is not possible. As groups will not have exactly the same probability, fixed bit assignment rule may result in a significant difference between number of ones and zeroes. We chose to invert this bit value for half of the pixels thus balancing more probable value 0 for one pixel by a higher probability of 1 for the next one and vice versa.

The independent binary streams were constructed for each color component using described extraction technique with all pixels within the frame (2100 frames were used, gathered in a burst of 7 consequent frames followed by approximately a 5 second delay needed to save data to removable card). All sequences (divided to 100 subsequences) were tested with the NIST battery at the significance level 0.01. One sequence failed for the red color, all passed for the green color and two sequences failed for the blue color (one template in non-overlapping template tests). We are aware of the possible impact of relatively short length of sequences, but we were restricted by the camera memory, acquisition speed (especially time needed to store the captured frame on a removable card) and 1 bit per pixel extraction technique.

3.3 Recapitulation

We have seen that mobile devices provide us with several sources of randomness that can be utilized for generating truly random data. Namely the microphone and camera that are available in (almost) each mobile phone have a promising potential and should allow for generating data with a sufficiently large amount of entropy. Such data that can be used for cryptographic purposes.

We used the 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. We performed several correlation tests that revealed a correlation in all recorded audio samples of noise, therefore we recommend to lower the estimated entropy at least 5 times with respect to the amount calculated for each sample value.

We always captured at least 2000 noise samples (at temperature between 5–8 °C) from digital camera that were used to create a histogram of values for a particular color pixel. The following analysis confirmed that all three tested devices have clearly different camera chips with different 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.

We used the NIST battery for statistical testing of the quality of noise samples, 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 image frames. We also verified the statistical properties of binary streams deterministically extracted (own flip-flop technique) from viewfinder output.

In spite of this fact, we were able to visually detect several systematic defects inherent to the optical sensor working principle. For example, blue color component on Nokia N73 shows visible row-dependent rips (caused by the readout technology) and red color component has significantly increased value towards chip borders. These defects indicate that there could be some statistical dependencies, but their quantitative analysis (in terms of min-entropy) is difficult.

Chapter 4

Digital post-processing

In the previous chapter we identified that both the microphone and camera noise are the most promising candidates for good sources of randomness. This noise is inherently presented in raw data produced by camera chip or microphone. Unfortunately, we cannot use such raw data directly as a random data, because they has typically unsuitable probability distribution. We need some kind of digital post-processing ensuring the uniformity of output and overcoming certain correlations or statistical dependencies caused by hardware sources of truly random data.

Therefore, we employ a pseudorandom data generator or randomness extractor for secure random data generation in mobile environments. Both could be used to postprocess the data captured directly from a physical randomness source, but pseudorandom data generators are universal (they have no assumptions regarding probability distributions of its input).

This chapter is mainly based on [BKMŠ09] and [KMŽ09] and is focused on the security aspects of integrating randomness extractors or selected pseudorandom data generators into mobile devices.

4.1 Randomness extractors

In order to use our sources of randomness for cryptographical purposes, we need to obtain a sequence of random bits distributed according to a uniform distribution. The problem we need to solve is that our device does not output (in general) a

biased distribution, but this distribution in not known in advance since it might be influenced by an adversary. Therefore, to postprocess our data we have to use some randomness extractor. The main criteria for random data generation in our setting are provable security against active adversaries and sufficient bit rate of the randomness source.

Our analysis concentrated on the camera and the microphone input noise. Digitalized microphone input is slightly (auto)correlated, and captured camera frames contain, due to post-processing and optical sensor technology (such as row-dependent readouts), several systematic defects (for details see chapter 3). These defects result in a decrease of the entropy of probability distribution supplied by these sources.

The camera provides higher data rate than the microphone and this also implies a higher entropy (and min-entropy) yield in the same time period. Unless explicitly mentioned, all experiment results refer to camera data obtained at 8 ℃, i.e., data with the smallest amount of noise, as expected.

4.1.1 Processing randomness

The most straightforward way to construct the post-processing software would be to estimate the probability distribution of camera output first ($180 * 240 * 8 * 3 = 1\,036\,800 \approx 2^{20}$ sample points), or at least the min-entropy of the distribution, and then to design a suitable randomness extractor. The obvious obstacle to this method is the size of the sample space that essentially prevents reasonable statistical testing (the number of samples would be practically unreachable).

We will limit the amount of data used from each frame to 4 bits only to allow for statistical testing. In order to use more bits from the frame (e.g., to increase expected min-entropy) we would not take 4 raw bits directly obtained from the camera, but rather as a function of a number of input bits.

To extract random bits from a CCD/CMOS chip we adopted the following general approach (see figure 4.1):

1. Acquire a picture (frame) from a CCD/CMOS chip of the camera (when the lens is closed).
2. Apply a function f (see below) that distills few random bits from the input frame.

3. Repeat steps 1. and 2. until a sufficiently long output sequence is accumulated. Then input this sequence to the randomness extractor (see below) to obtain the final random sequence.

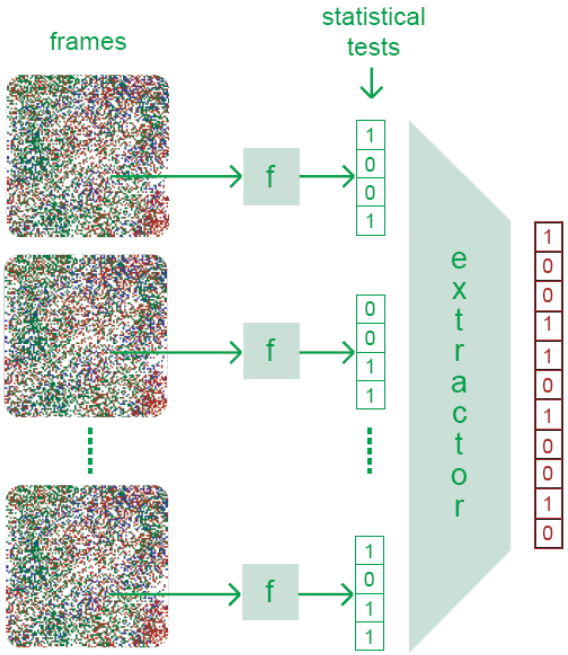

Figure 4.1: Scheme of the image data post-processing.

The role of the function f is to pre-process the random data in a way that (regardless of the limited actions of the adversary) the probability distribution of output sequences of the function f has with high probability min-entropy sufficient for successful randomness extraction (for chosen extractor). The main problem when designing the function f is the limited knowledge of the CCD/CMOS chip specification and the post-processing done by the camera. Moreover, these details change with both models and producers of the mobile devices. We use the function f to separate the extractor from these technical assumptions.

Following the aforementioned motivation we designed the function f to be an XOR (parity) of least significant bits (LSBs) of selected (colors of) pixels of the CCD/CMOS chip. The choice of LSBs has the obvious motivation that this bit is hard to predict and to flip this bit a very accurate (physical) attack is needed[1].

[1] As a further improvement we discuss also XORing all bits of particular pixel instead of LSBs only.

We make one important assumption in our analysis – that bits obtained from different frames are independent. To assure this, we cyclically change pixels used as inputs for the function f. Also, we limit the sampling frequency to 12 frames per second.

Let us concentrate on the noise arising from the CCD/CMOS chip in the smartphone Nokia N73. The generator we considered takes 12 pictures per second from the view finding (resolution 180×240 function of the cell phone), while the lens of the camera is shut and therefore the influence of the output by external lights is minimized. Randomness contained in pictures comes mainly from the heat inducing electrical charge in cells of the CCD/CMOS chip. The size of possible pictures is too large to allow for a thorough statistical testing and therefore we use a function f that extracts 4 bits from each picture. We proceed with sampling a sufficient number of 4-bit sequences to estimate the probability distribution our source delivers with a reasonable confidence. We run the same tests in various external conditions that may influence the source (cold, heat – 8 ℃, 20 ℃ and 45 ℃) and obtain limitations on probability distributions an adversary can achieve. More details of the performed tests are presented in section 4.1.3.

More precisely, we used 180×240 pixels from each frame. Function f logically divides pixels in one frame to 300 squares of 12×12 pixels and uses exactly one pixel from each square. These pixels are selected deterministically in such a way that the neighbors are always from different row and column. They are selected from the diagonal of the square – concretely, the position of the pixel on the diagonal (between 1–12) is computed as a sum (modulo 12) of indexes of X- and Y-axis of the left top corner in the particular square. Changing the position of selected bits in each frame (cyclically) improves independence of outputs acquired from consecutive frames.

Finally, we extract only four bits from these 15×20 squares/pixels. Firstly, we apply the XOR function to all pixels (and colors) in a column. Then we transform the resulting 20 bytes (1 byte per column) into 4 bytes by XORing together several different non-neighbor columns together: $(1, 5, 9, 13, 17)$, $(3, 7, 11, 15, 19)$, $(2, 6, 10, 14, 18)$, and $(4, 8, 12, 16, 20)$. At the end we can apply XOR also to all bits in a byte, or alternatively extract only the LSB from each byte. The output is in both cases exactly four bits per frame. For statistical properties of the output see section 4.1.3.

4.1.2 Randomness extractor

In this section we provide a brief description of our randomness extractor based on Carter-Wegman universal classes of hash functions. The principal author of this description (that was published in [BKMŠ09]) is Jan Bouda. The randomness extractor is a function $e : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ that takes n bit random (biased) input, d bit auxiliary uniformly distributed random input and outputs m bit sequence that should be distributed according to the (almost) uniform probability distribution. The processed data obtained from the camera are used as the first parameter, while the second parameter requires true random (uniformly distributed) data. This means that such data must be pre-generated, and it can be, e.g., stored in the memory and refreshed via network after a fixed number of extractor iterations (meanwhile the output of the extractor is used). This is an obvious drawback, but such an extractor can be used to post-process a wide range of input probability distributions, namely any probability distribution with sufficiently high (pre-chosen) amount of randomness.

The other possibility is to use a deterministic extractor $e : \{0,1\}^n \to \{0,1\}^m$ requiring no uniformly distributed input. Unfortunately, possibilities of such a source are strongly restricted – it partitions all inputs into fixed subsets, each of which is mapped to one fixed output bit sequence. Therefore, this extractor gives unbiased output only as long as the probability of each such subset is unchanged. It may be difficult to explicitly construct one suitable deterministic extractor, however, non-deterministic extractor with random, but public and fixed auxiliary input can do very well [BST03] in some situations. For more details on randomness extractors see the survey paper [Sha02].

To fight a general and powerful adversary we decided to use a (non-deterministic) randomness extractor, i.e., extractor requiring true randomness as an auxiliary input. This allows us to concentrate only on the analysis of the min-entropy of our randomness source (denoted k). Due to a high min-entropy of our source (see section 4.1.3) we can choose freely from a wide variety of extractors. First parameter of extractor we should consider is the length d of the auxiliary random input. In our case this should be considered only in the context of the randomness efficiency and we should compare the length of the output sequence with the amount of randomness in the input, i.e., the length of the output should be approaching $d + k$.

Possibly limiting factor of the initial sequence can be memory of the mobile device, but essentially all devices with a camera already have memory of size (at least)

few megabytes (few hundreds for contemporary mobile phones) and therefore we are not limited by a reasonably large auxiliary input that has to be stored (we consider approximately 1000 bits for our purposes).

Since the absolute value of the true random input is not limiting and we have a high min-entropy, we can concentrate on the computational efficiency of our extractor. Computing power of mobile devices is still limited and intensive calculations increase battery exhaustion and thus the computational efficiency is of critical importance. The amount of true randomness is limited, and therefore we would like to reuse it or use the output of our extractor instead. A straightforward choice from this point of view are extractors based on Carter-Wegman universal classes of hash functions.

Any *XOR universal* class of functions is universal$_2$ as well [Sho96]. Therefore, we may use any of the classes of hash functions designed for message authentication and heavily optimized towards computational efficiency. In our tests we used the universal$_2$ class of functions based on shift registers proposed in [IZ89]. Implementation of this class is efficient and straightforward – all necessary calculations are scalar product (i.e., bitwise XOR and mod 2 addition) and bit string rotation. Formal definitions and suitable parameters setting for our extractor can be found in [BKMŠ09]. As a final output of the extractor we would like to obtain a random sequence that is at least 2^{-64} close[2] to the uniform distribution. Note, that if we use a part of our output as the d auxiliary bits or we reuse original randomness, the quality of the output distribution decreases.

In our settings we are able to extract from each 838 bits of input 629 bits of output (compare to the theoretic optimal value 797 bits) giving bit rate approximatively 36 bits per second. We stress that the bit rate is limited only to allow statistical testing and can grow easily to approximatively to 18 000 bits per second (100 pixels used to generate 4 output bits).

4.1.3 Analysis of acquired random data

To verify the extractor input and its min-entropy we ran a number of statistical tests on outputs of the function f. To obtain an acceptable quality of the tests on an achievable number of samples (we used 1.5 million sample frames) we output only 4 bits from each frame (16 different values).

[2] As a measure we use the trace distance (also called variational or Kolmogorov distance) between the probability distributions.

First battery of tests concentrates mainly on the statistical *stability* of the output distribution – i.e., that the output data have all time (even in different external conditions) almost the same distribution (not necessarily uniform). We applied Pearson's (χ^2) goodness of fit test on several sequences and subsequences. As the expected distribution we take the relative frequency of 4-bit values from the whole sequence generated from all 1.5 million sample frames. In this case (data captured at 8 ℃) are relative frequencies given by values: 0.0630, 0.0608, 0.0674, 0.0629, 0.0607, 0.0697, 0.0639, 0.0607, 0.0642, 0.0606, 0.0573, 0.0580, 0.0642, 0.0627, 0.0612, 0.0625. As the observed distribution we use the relative frequency of the 4-bit values from various subsequences – in our case, e.g., given by values: 0.0642, 0.0642, 0.0706, 0.0645, 0.0545, 0.0691, 0.0621, 0.0594, 0.0624, 0.0624, 0.0561, 0.0542, 0.0624, 0.0585, 0.0606, 0.0645. The resulting p-value (in this particular case 0.22) at the significance level $\alpha = 0.01$ clearly confirms our hypotheses that the sequences are at 99% from the same probability distribution.

Relative frequencies that we obtained (outcomes only for normal temperature) indicate that the min-entropy of tested distribution is approximatively 3.984, which is almost the maximum of 4. As a further evidence of high entropy of our source we test the hypothesis that an output sequence was generated by a source with min-entropy smaller than $log_{14} = 3,8$[3]. We calculated the minimum of the Pearson's χ^2 test of our estimated distribution over all distributions with min-entropy at most $\log_2 14$. We calculated

$$\min_{H_\infty(q) \leq \log 14} \chi^2(p, q)$$

with p being our estimated distribution. The outcome is $\chi^2 = 338.9947$, p-value is 2.0167×10^{-63}, what strongly rejects such a hypothesis.

As a part of our testing we considered other functions f as well. While it is easy to see that XORing an extra random bit to a given random bit can only decrease possible bias (or preserve it when the bit is deterministic), it would be nice to use as few input bits as possible. The reason for such optimization is not only the computational efficiency, but also allocation of resources. As already mentioned, in our testing we used only a 4-bit output, but it might be useful to use just few input bits to design extractors with a higher bit rate, where the independence of input data is assured at hardware level. Our analysis showed that green color behaves in general more deterministically than red and blue color. Also, XORing few pixels together already helps significantly to obtain a higher

[3]We designed the extractor the way it works for any distribution with min-entropy at least $\log_2 14$ bits.

entropy probability distribution. When comparing the function f as described in section 4.1.1, it has only a negligible (improving) impact if we XOR all bits from the pixel color channel instead of using only LSBs. Therefore, a suitable function f should XOR together bits from several pixels distributed over the frame and use data from all color channels. On the other hand, it is sufficient to use LSBs only.

Furthermore, we performed several statistical tests of the output produced by our implementation of the randomness extractor (described in section 4.1.2). In this case we applied the well-known NIST statistical test suite to a 40 MB output. All sequences passed 15 out of 16 statistical tests (frequency test, runs test, etc.). We tested division to 100 subsequences at the confidence level $\alpha = 0.01$. Passing the test means that a large fraction of subsequences passed particular sub-tests and also that the resulting p-values have uniform distribution. The Chi-square (χ^2) goodness of fit test is used to test the uniformity as well, but the confidence level is more strict in this case ($\alpha = 0.001$). The only one exception was "the serial test" that failed due to the small fraction of passing subsequences – their fraction 0.9500 was lower than (for 100 subsequences) the expected 0.960150.

4.1.4 Summary

We described a true random data generator delivering a bit sequence distributed according to a probability distribution very close to the uniform probability distribution. The initial randomness is acquired partly as an initial true random bit sequence and partly as a processed output of a built-in camera. In our analysis we used only a negligible portion of randomness contained in each frame acquired from the camera. This was only to allow reasonable statistical testing of data, while in practice much larger amount of data can be obtained from each frame. This would increase the bit rate significantly.

In the next section we discuss a universal alternative to randomness extractors – pseudorandom data generators – that require no assumptions regarding probability distributions of its input from randomness sources. The only drawback of such generators is that they rely on unproven assumptions whereas randomness extractors typically provide some provable guarantees of their output.

4.2 Secure pseudorandom generators

The security of pseudorandom data generation relies on the design of a particular cryptographic pseudorandom data generator and its resistance to cryptanalysis. Three classes of such generators can be distinguished – generators based on linear feedback shift registers, classical cryptographic functions, and hard problems of number and complexity theory. If these generators behave correctly (e.g., in a secure computing environment), we can conclude that the order of these three classes often reflects both speed and security of particular generators. The generators in the first group are the fastest but less secure, while generators in the third group provide excellent security but they are often the slowest. In practice, generators based on classical cryptographic functions are typically used.

Unfortunately, the vast 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, etc.). Therefore a careful generator design must take into account both forward and backward security after a compromise of the generator internal secret state. Forward/backward security means that an attacker with the knowledge of the 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 the use of one-way functions and backward security requires periodical refreshing of the internal state with additional truly random data.

Modern *hybrid pseudorandom data generators* periodically use truly random data during the whole generation process – this improves the generator security by increasing resistance against state compromise attacks at the expense of higher demands on truly random data. The consequence of periodical refreshing of the internal state is utilizing all available random samples – with the possibility of their gathering and accumulation (so-called pooling) even in the time when no generation of pseudorandom data is required. Such hybrid generators behave deterministically only between two successive reseedings.

We selected ANSI X9.31 and Fortuna pseudorandom data generators for our reference implementation (for details see chapter 5) and their basic architecture and security properties are briefly described below.

4.2.1 ANSI X9.31 pseudorandom data generator

The ANSI X9.31 (former X9.8) pseudorandom data generator uses only one cryptographic primitive, a block cipher. The NIST specification [Kel05] presents 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 data. E_K denotes the 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:

$$I = E_K(DT), \tag{4.1}$$
$$R = E_K(I \oplus V). \tag{4.2}$$

And then the seed V is updated in each iteration:

$$V = E_K(R \oplus I). \tag{4.3}$$

It is obvious that the generator has no embedded pooling mechanism and the entropy gathering[4] is performed on the fly. This implies relatively easy integration into almost all user applications. 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. Since the mobile devices provide no secure computing environment (as, for example, smartcards), we strongly recommended the regeneration of key K – at least at the beginning of the application process.

Another part of inner state, the value R, is original design produced directly as an output[5]. However, revealing any part of the inner state is not a good security feature of any general purpose pseudorandom generator. The DT vector is based solely on low-entropy date/time and this implies slow recovery after a state compromise and therefore weak backward security. We thus suggest to employ additional randomness captured from microphone or digital camera.

[4]Note that we defined the entropy as a measure – the gathering, accumulating, adding or similar operations are obviously always performed with data that are expected to have sufficient amount of entropy.

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

It was not our intention to discuss and describe all security aspects of this generator, thus we restricted ourselves only to this summary of very basic facts and our suggestions. Detailed security analysis with several proposed improvements (e.g., integration of the entropy pool) to this generator can be found in [Gut98] and [KSWH98].

4.2.2 Fortuna pseudorandom data generator

The Fortuna [FS03] is a famous successor to the Yarrow pseudorandom data generator [KSF99]. Fortuna has an improved and automated mechanism of pooling and extremely simplified (almost removed) process of entropy estimation. It currently represents one of the most sophisticated pseudorandom generator – designed more for integration into operating system or cryptographic libraries than for integration into end-user applications.

Fortuna pseudorandom data generator consists of three components. The first part is the *generator* itself, it takes a fixed-size seed and outputs an arbitrary amount[6] 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 is 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 (i.e., rekeying is in this case equal to reseeding) and this property prevents short cycles that could occur due to repeated 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.

[6] In practice it is often restricted to 1 MB per single request.

The accumulator is also responsible for periodical reseeding of the generator – this mechanism is very special. 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 determines the minimal pool size necessary for reseeding – but it can be quite optimistic, e.g., 64 bytes for required 128 bits of entropy.

Last but not least, the *seed file* is another component of the Fortuna pseudorandom data generator. It preserves the generator internal state and ensures that even after rebooting the generator can produce good pseudorandom data. Seed file serves as the storage of high-entropy data, which is used after a restart of the generator to initialize the generator properly. Quite a simple concept in theory, but its practical implementation is very difficult and depends extremely on the environment and platform, because it must follow several strict requirements.

The seed file must be available to the pseudorandom data generator only, otherwise an attacker can reproduce the generator output at least until the first reseeding. Its content needs to be refreshed before the user can request first random data, otherwise the attacker can request data and reboot before the seed file update. Other issues include the first reboot, where there is no seed file yet, and backups of file system, because restoring the file system clearly results in the restoring and usage of an old seed file.

The detailed description and security analysis of Fortuna can be found in [FS03].

4.3 Recapitulation

We investigated relevant security aspects of integration of randomness extractors or selected pseudorandom data generators into mobile devices. Both methods are significantly different and both methods have their own pros and cons that should be carefully considered before their practical application.

The theory of randomness extractors brings an interesting ideas and solution how to post-process truly random data. However, a lot of current randomness extractors relies on a very strong assumptions and their practical usage is therefore very questionable. Regardless of our findings, we prefer pseudorandom data generators that we also use in our design prototype. Detailed description including implementation details of this prototype can be found in chapter 5.

Chapter 5

Design prototype

This chapter is based mainly on [KMŽ09] and deals with the integration of our design prototype into selected mobile device. We use the Symbian-based smartphone Nokia N73 as our target platform and we summarize details of integration of ANSI X9.31 and Fortuna pseudorandom data generators into the Symbian OS 9.x. We also discuss the performance analysis of such generators and the power consumption of our Fortuna implementation. The reference implementation of the design prototype into smartphone Nokia N73 was performed within a master thesis work of Jiří Žižkovský.

5.1 Suitable target platform

The basic (and the most important) requirement for generation of truly random data on mobile devices is the availability of one (or several) good source(s) of randomness. The experiments that we described in chapters 3 and 4 show that the modern mobile devices (with embedded microphone and digital camera) fulfil this requirement.

Apart from good sources of randomness we also need a suitable method of digital post-processing by randomness extractor or pseudorandom data generator. Post-processing by randomness extractor is reasonable for devices of the same family and type (i.e., with the same sources of randomness). Since we do not want to restrict ourselves to one particular family of devices, our implementation uses only a pseudorandom data generators (which are more universal).

As we want to post-process all data from the randomness sources, we need a sufficient performance for aforementioned digital post-processing that must be computationally feasible on such devices. To estimate an acceptable amount of raw data from randomness sources, we performed practical performance test (described also in [KŠMS07]) on mobile phones Nokia N73 (Symbian OS v9.1), Nokia N73 (JavaME), Sony-Ericsson k750i (JavaME), Nokia 6230 (JavaME), Nokia 6021 (JavaME). SHA-1 hash function were used as an entropy extraction/mixing function and the resulting throughput of such mobile phones is 2200 KB/s, 426 KB/s, 84 KB/s, 67 KB/s and 4.65 KB/s, respectively.

Both programming environments (Symbian C++ and Java) of Nokia N73 provide enough computation power for real-time extraction from view finder (camera input 180× 240 pixels, ∼380 KB/sec) and microphone (16-bit mono PCM 8000 Hz, ∼16 KB/sec). We thus selected a Series60 smartphone Nokia N73 with a Symbian OS 9.x as our target platform.

5.2 ANSI X9.31 pseudorandom data generator

The ANSI X9.31 pseudorandom data generator was implemented as a simple GUI application. We used two cryptographic primitives for our implementation – the AES encryption function and the SHA-1 hash function. SHA-1 is 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 [Hen05]. The generator itself is implemented in one class *CX931_AES*.

The class is derived from the class *CCoeControl* and can be used for capturing events from the keyboard. The class also implements two interfaces *MCameraObserver* and *MMdaAudioInputStreamCallback*, the former is used to capture events from the camera, and the latter to capture events from the microphone.

The class *CX931_AES* also consists of various data objects. There are object attributes that preserve the generator secret state: AES cipher context, the 128-bit AES secret key, the 128-bit seed, and the 128-bit date/time vector. Further there are object attributes necessary for the generator initialization – the most important is a disposable 320-bit pool of entropy from the camera, the microphone, and the keyboard. Construction and initialization of the camera and microphone objects run asynchronously and the speed of data acquisition from these sources of randomness is 1495 KB/s.

The simple 320-bit pool 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. Since the estimated amount of entropy from these samples radically exceeds 320-bit, we can expect that the pool contains 320-bit of initial entropy.

There are two important public methods for users of our X9.31 pseudorandom data generator implementation – namely `GetRandom(TUint8 *rnd)` and `GetRandomFileL(RFile &aFile, TInt aCount)`. The former returns 16 bytes of pseudorandom data in its parameter `rnd`. The latter uses `GetRandom(TUint8 *rnd)` to write `aCount` bytes of pseudorandom data into `aFile`.

When the user calls one of these methods for the first time, the secret state of the generator is initialized and preserved in above mentioned data object attributes. The entropy pool is used only for the following (initialization) 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 (some issues regarding energy requirements are discussed in section 5.3.1). 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 (long-term) 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. He simply decrypts the seed V and encrypts DT, then computes previous random bits by XORing these values. We also implemented a second version of ANSI X9.8 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. This modification was originally suggested in [Gut98].

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 proven 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 X9.8/X9.31 are discussed in [KSWH98].

The actual speed of our implementation is only 2.44 KB/s with the reference non-optimized AES implementation of encryption speed (without key scheduling) 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 as a general purpose pseudorandom data generator is not advisable.

5.3 Fortuna pseudorandom data generator

While the above described ANSI X9.31 pseudorandom data generator belongs to the simpler (and not so secure) pseudorandom data generators, Fortuna is a very good pseudorandom data generator of sophisticated design. In this section we describe the implementation of Fortuna generator on Nokia Series60 mobile devices with the Symbian OS 9.x. 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 pseudorandom data generator is designed to run continually, gathering entropy and servicing user requests for pseudorandom data. Therefore we decided to implement the Fortuna by means of the Symbian OS *Client-Server Framework*. It implies that the implementation is divided in two parts. The server component is the application without a graphical user interface, implementing whole functionality of the Fortuna. The client component is the interface to the Fortuna pseudorandom data generator, implemented as a dynamic-link library. Users, who want to use the Fortuna, have to link their applications against this library and then they can use Fortuna via the exported interface.

During the implementation of Fortuna were developed also two other applications. The first one is an application dedicated to control the run of the server. We recommend using it only for debugging purposes, and for practical usage the server should be started with the device booting and stopped when the device goes down. The second auxiliary application is an example application using the Fortuna generator to get some pseudorandom data. It simply detects whether the Fortuna server is running and in the positive case requests some pseudorandom data, and saves it into a binary file.

5.3.1 Fortuna server

The Fortuna server is a crucial part of the Fortuna pseudorandom data generator, 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. It also includes a handling mechanism for all sources of entropy. The class `CFortunaPRNG` implements all three components of the Fortuna, therefore it consists of: `TFGenerator` implementing the generator component, `TFPool` implementing the accumulator component and `RFile` as the seed file component.

The class `TFGenerator` is a straightforward implementation of the generator component. It 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 consists of the array of 32 objects of the `TFPool` class. Each object of the `TFPool` class 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[1], therefore the class consists of two data object attributes *iData*, representing events data, and *iEntropy*, representing the estimate of entropy included in the pool. Initialization of the pool is easy, the SHA-256 context *iData* is created and the *iEntropy* value is set to zero.

The seed file component does not have its individual implementing class, the mechanism is implemented directly in the `CFortunaPRNG`. The read-part is a

[1] One can observe several differences between the Fortuna design and our implementation, but all changes are consistent. The authors of the Fortuna proposed the solution with random events in unbounded strings and the entropy of such pool was estimated according to the length of the pool string. Yet with respect to the running hash our pool string has a fixed length.

constructor in the second-phase. It tries to open the seed file in the `/Private` folder of the application. This folder is accessible only to this application, except for processes with the special 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 long enough, the data is not used and the generator is not initialized. If the seed file is not found, it will create an empty one, 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 `CFortunaPRNG` class contains two other object attributes relevant to reseeding mechanism. The *ReseedCnt* represents 64-bit counter incremented at each reseeding. The *lastReseed* value represents the time when the last reseeding was done. This value is checked during next reseeding attempt, and there have to be at least 100 ms between two successive reseedings.

Both the attribute *src* and method `void AddRandomEvent (TInt srcNum, const TDesC8 &event, TInt entropy)` are relevant to the accumulator component. The sources of randomness do not have direct access to the particular pools, but they use `AddRandomEvent` method to add new random data to the accumulator. It ensures that the pools are filled in a cyclical fashion. Every source is identified by a unique number *srcNum* and according to this ID the accumulator chooses the appropriate pool in which to add the event data. This choice is made on the basis of the *src* array, in which the current pool number for each randomness source is saved.

In principle, all sources of randomness are asynchronous services, therefore the concept of active objects was chosen. We implemented three sources of randomness: the keyboard (`CFKeyRandSrc`), the microphone (`CFAudioRandSrc`) and the camera (`CFCamRandSrc`). The first randomness source gathers entropy from keyboard events. The randomness source starts waiting for the key events and when the key is pressed, the window server[2] 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 of key event arising is sent to the appropriate pool. According to our analysis (described in section 5.2) the entropy of data generated by this randomness source was estimated at 12.96 bits.

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

We now describe two remaining sources of randomness that are much more powerful. Let us start with the randomness source gathering entropy from the microphone device. It is implemented in the `CFAudioRandSrc` class that contains the `MMdaAudioInputStreamCallback` interface. This class implements callback functions to handle the audio data from the microphone. When the requested audio data is prepared, the method `void CFAudioRandSrc::MaiscBufferCopied` is called back and then the requested data from buffer is sent to the appropriate pool.

According to our entropy estimation the 1 KB sample contains approximately 51 bits of entropy. The period and 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 – but it is always a trade-off between the security and usability.

The last implemented randomness source gathers entropy from the camera device of the phone – it is very similar to the `CFAudioRandSrc`. The class implements the `MCameraObserver` interface, which specifies five callback methods that must be implemented by the classes to capture data from the camera. As one frame from the viewfinder provides 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 it is possible to capture the viewfinder frames continually, it would considerably limit the device user. The camera has a significant power consumption and moreover it would be reserved for the Fortuna server only. As well as in the case of `CFAudioRandSrc`, this should be considered carefully while developing a particular application. We capture only one viewfinder frame every minute.

We performed five battery life tests[3] with continuous utilization of different sources of randomness in the Fortuna server. Keyboard events have been monitored at all times, but no keys were pressed during measurements. The lowest speed of our implementation in the continuous[4] capturing mode is 13.85 KB/s. The detailed results are summarized in table 5.1 – the first column presents the average time of battery life and the second column then standard deviation of our measurements.

[3] All measurements were performed with a Li-Pol battery type BP-6M (970 mAh) that was roughly 1.5 years old.

[4] The generator speed is independent of the used sources of randomness or speed of the event sampling.

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

Table 5.1: Fortuna energy requirements – battery life.

Our current Fortuna implementation captures both audio and video periodically 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 (i.e., without any entropy pooling).

For illustration, a continuous utilization of all sources of randomness (i.e., keyboard, camera, and microphone) compared to our current reference Fortuna implementation (i.e., sampling once per minute) drains the battery 26.53 times faster. For the keyboard with camera only it is 23.74 times more power demanding, and for the keyboard with microphone only setting it consumes the battery just 5.17 times faster. Energy requirements of our current reference Fortuna implementation are 2.78 times higher then energy requirements of implementation without any entropy pooling.

5.3.2 Fortuna client

The client part of the Fortuna represents an interface to the pseudorandom data generator functionality. It is implemented as a library with four exported functions. The method `TInt Start()` allows to start the Fortuna server via non-member function `TInt StartTheServer()`, which creates new process and runs the server. The method `TInt Connect()` creates new session with the running server using `RSessionBase::CreateSession()`.

Two remaining methods are operations with an established session. Both methods use inter-process communication (`RSessionBase::SendReceive()`) to communicate with the Fortuna server. The `TInt GetRandomData(TInt aSize, TDes8 &aDes)` requests the Fortuna server for *aSize* bytes of pseudorandom data. The `TInt StopServer()` method requests to stop the server. It should be mentioned that this last part of the interface should be removed in the final version of interface to avoid the attacker stopping the server.

5.4 Recapitulation

We described several implementation issues and technical details of our design prototype that we integrated into the Nokia N73 with Symbian OS 9.x. We conclude that our ANSI X9.31 implementation is suitable for the usage as a part of any user application whereas our Fortuna implemetation can be used as a server providing pseudorandom data on demand of one (or several) user application(s). A tradeoff between security and energy requirements was also discussed together with tests of Fortuna energy requirements.

Chapter 6

Conclusions

This dissertation thesis dealt with the generation of truly random and pseudorandom data in mobile computing environments (e.g., mobile phones, personal digital assistants, cryptographic smartcards, etc.). We examined basic requirements on random and pseudorandom data for cryptographic purposes and the methods of its generation in general purpose computer systems. We showed a practical approach to random data generation for the mobile environments, providing a clear path to developers of many security-critical applications for the mobile environments.

In chapter 2 we examined a basic requirements on random and pseudorandom data for cryptographic purposes. We investigated several possible ways of generating both random and pseudorandom data in general purpose computer systems and mobile devices. We described our experiments with hardware generators of selected cryptographic smartcards and hardware security modules. In the last part of this chapter we discussed basic security aspects of distributed random data generation in potentially hostile environments. This distributed approach can be a good starting point for better random or pseudorandom data generation in case of the attackers being able to target only some (but not all) of the mobile devices.

We started chapter 3 with identification and analysis of available sources of randomness and assessment of their quality and performance. All results and our (min-)entropy estimations in this chapter correspond to the worst values (i.e., lower measured entropy values) that were obtained during several experiments with mobile devices. Devices were running in worsened (but for the devices still acceptable) conditions – e.g., in several different external conditions (temperature,

ambient light, acoustic noise, etc.). Our analysis showed that mobile devices have several good sources of randomness – we confirmed that at least the microphone and camera noise contain a sufficient amount of entropy and thus can be reliably used as a good sources of truly random data. We also detected several systematic defects inherent to the optical sensor working principle or defects caused by its controller firmware.

Our work in chapter 4 then led to the investigation of the truly random data post-processing with the use of randomness extractors or pseudorandom data generators. We discussed the need for some kind of digital post-processing that ensures the uniformity of output and overcomes certain correlations or statistical dependencies caused by hardware sources of truly random data. We analyzed the possibilities of secure, efficient and straightforward implementation of randomness extractors based on Carter-Wegman universal classes of hash functions, ANSI X9.31 and Fortuna pseudorandom data generators.

Chapter 5 dealt with the integration of our design prototype into selected mobile devices. We used the smartphone Nokia N73 as our target platform for the integration of ANSI X9.31 pseudorandom data generator. Devices based on Symbian OS 9.x were used for the integration of the Fortuna pseudorandom data generator. We also discussed a tradeoff between security and energy requirements of our implementations.

During our work described in chapters 2, 3, 4 and 5 we also performed an extensive statistical testing. We tested hundreds of truly random and pseudorandom sequences generated by cryptographic smartcards, hardware security modules, mobile devices or even general purpose computers (PCs). Our statistical testing utilized predominantly the NIST and the DIEHARD test batteries but some of the sequences generated by the cryptographic smartcards were also tested by the TESTU01 and the RaBiGeTe test batteries.

Although our experiments revealed (in some cases) no statistically significant defects or dependencies in generated sequences, we note that no certain conclusions about a specific generator or source of randomness can be made (i.e., there could be some statistical defects or dependencies in generated sequences and these defects could be detected by another testing approaches).

6.1 Future work

We want to point out that on top of the truly random data sources that we examined in this thesis, 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 based on Symbian OS phones can also be addressed by teams that have an access to lower-level of API. We were working with API that is available to other ordinary developers.

We also proposed a distributed approach to the generation of random data and discussed some of its non-trivial shortcomings and obstacles. In spite of that we think that this pioneer idea could still have a potential and is interesting for future research. We elaborated this idea on a theoretical level in the GSM mobile network environment, but it would also be interesting to reconsider our proposed approach for fully distributed environments.

Bibliography

[ANS85] ANSI X9.17 (Revised), American National Standard for Financial Institution Key Management (Wholesale), American Bankers Association, May 1985.

[ARV95] W. Aiello, S. Rajagopalan, and R. Venkatesan. Design of Practical and Provably Good Random Number Generators. In *5th Annual ACM-SIAM Symposium of Discrete Algorithms*, pages 1–8, 1995.

[ARV99] W. Aiello, S. Rajagopalan, and R. Venkatesan. High-Speed Pseudorandom Number Generation with Small Memory. In *Lecture Notes in Computer Science*, volume 1636, pages 290–304. Springer, 1999.

[Bal93] D. Balenson. RFC 1423 – Privacy Enhancement for Internet Electronic Mail: Part III: Algorithms, Modes, and Identifiers, 1993. Available at: http://www.ietf.org/rfc/rfc1423.txt [online; accessed 2009].

[BB99] V. Bagini and M. Bucci. A Design of Reliable True Random Number Generator for Cryptographic Applications. In *Proceedings of the 1st Workshop Cryptographic Hardware and Embedded Systems (CHES) 1999*, volume 1717 of *Lecture Notes in Computer Science*, pages 204–218. Springer, 1999.

[BBL04] H. Bock, M. Bucci, and R. Luzzi. An Offset-Compensated Oscillator-Based Random Bit Source for Security Applications. In *Cryptographic Hardware and Embedded Systems (CHES) 2004*, volume 3156 of *Lecture Notes in Computer Science*, pages 268–281. Springer, 2004.

[BCEP04] E. Bresson, O. Chevassut, A. Essiari, and D. Pointcheval. Mutual Authentication and Group Key Agreement for Low-Power Mobile Devices. In *Fifth IFIP–TC6 International Conference on Mobile and Wireless Communications Networks*, pages 59–62, 2004.

[BCK+06] J. Barnat, D. Cvrček, J. Krhovják, V. Lorenc, V. Matyáš, Z. Říha, J. Staudek, and P. Švenda. Smartcards, final report for the Czech National Security Authority, December 2006.

[BGL+03] M. Bucci, L. Germani, R. Luzzi, A. Trifiletti, and M. Varanonuovo. A High-Speed Oscillator-Based Truly Random Number Source for Cryptographic Applications on a Smart Card IC. *IEEE Transactions Computers*, 52(4):403–409, April 2003. Available at: http://csdl.computer.org/dl/trans/tc/2003/04/t0403.pdf [online; accessed 2009].

[BH05] B. Barak and Shai Halevi. A model and architecture for pseudo-random generation and applications to /dev/random. In *Proceedings of the ACM CCS'05*, November 2005.

[BK07] E. Barker and J. Kelsey. Recommendation for Random Number Generation Using Deterministic Random Bit Generators (Revised). National Institute of Standards and Technology – Special Publication 800-90. NIST Computer Security Division, March 2007. Available at: http://csrc.nist.gov/publications/nistpubs/800-90/SP800-90revised_March2007.pdf [online; accessed 2009].

[BKMŠ09] J. Bouda, J. Krhovják, V. Matyáš, and P. Švenda. Towards True Random Number Generation in Mobile Environments. Submitted to the special issue of Computing. Springer, 2009.

[BL05] M. Bucci and R. Luzzi. Design of Testable Random Bit Generators. In *Cryptographic Hardware and Embedded Systems (CHES) 2005*, volume 3659 of *Lecture Notes in Computer Science*, pages 147–256. Springer, 2005.

[BM92] S. M. Bellovin and M. Merritt. Encrypted Key Exchange: Password-based protocols secure against dictionary attacks. In *Proceedings IEEE Computer Society symposium on Research in Security and Privacy*, pages 72–84, May 1992. Available at: http://www.cs.columbia.edu/~smb/papers/neke.pdf [online; accessed 2009].

[BM03] C. Boyd and A. Mathuria. *Protocols for Authentication and Key Establishment*. Springer-Verlag, 2003. ISBN 978-3540431077.

[Boy89] J. Boyar. Inferring sequences produced by pseudo-random number generators. *Journal of the ACM (JACM)*, 36(1):129–141, 1989.

[BR94] M. Bellare and P. Rogaway. Optimal Asymmetric Encryption – How to Encrypt with RSA. In *Eurocrypt '94*, volume 950 of *Lecture Notes in Computer Science*, pages 92–111. Springer, 1994. Available at: http://www-cse.ucsd.edu/~mihir/papers/oae.pdf [online; accessed 2009].

[BST03] B. Barak, R. Shaltiel, and E. Tromer. True Random Number Generators Secure in a Changing Environment. In *Cryptographic Hardware and Embedded Systems (CHES)*, pages 166–180. Springer-Verlag, 2003. Available at: http://theory.csail.mit.edu/~tromer/papers/rng.pdf [online; accessed 2009].

[BTSL01] M. Bucci, E. Trichina, D. De Seta, and R. Luzzi. Supplemental Cryptographic Hardware for Smart Cards. *IEEE Micro*, 21(6):26–35, December 2001. Available at: http://csdl.computer.org/dl/mags/mi/2001/06/m6026.pdf [online; accessed 2009].

[Cen02] Center for Information Security and Cryptography (CISC). Library of Tests for Random Number Generators, 2002. Available at: http://www.csis.hku.hk/cisc/download/idetect/ [online; accessed 2009].

[ČH04] M. Čagalj and J.-P. Hubau. Key agreement over a radio link. Technical Report IC/2004/16, EPFL-IC Technical Report, 2004.

[CHK+05] D. Cvrček, V. Holer, J. Krhovják, V. Lorenc, V. Matyáš, Z. Říha, P. Švenda, and P. Švéda. Smartcards, final report for the Czech National Security Authority, December 2005.

[CKM+04] D. Cvrček, J. Krhovják, V. Matyáš, Z. Říha, P. Švenda, and P. Švéda. Smartcards, final report for the Czech National Security Authority, December 2004.

[Cor00] J.-S. Coron. On the Exact Security of Full Domain Hash. In *Crypto 2000*, volume 1880 of *Lecture Notes in Computer Science*, pages 229–235. Springer, 2000. Available at: http://www.iacr.org/archive/crypto2000/18800229/18800229.pdf [online; accessed 2009].

[Cry06] Information Security Research Centre at Queensland. Crypt-XS Test Suite, University of Technology in Australia, 2006. Available at: http://www.isi.qut.edu.au/resources/cryptx/ [online; accessed 2009].

[Dav00] R. Davies. Hardware random number generators, 2000. Available at: http://www.robertnz.net/hwrng.htm [online; accessed 2009].

[DGP07] L. Dorrendorf, Z. Gutterman, and B. Pinkas. Cryptanalysis of the Windows Random Number Generator. 2007. Available at: http://eprint.iacr.org/2007/419.pdf [online; accessed 2009].

[DH76] W. Diffie and M. E. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, November 1976.

[DHY02] A. Desai, A. Hevia, and Y. L. Yin. A Practice-Oriented Treatment of Pseudorandom Number Generators. In *EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 368–383. Springer, 2002.

[DIF94] D. Davis, R. Ihaka, and P. Fenstermacher. Cryptographic Randomness from Air Turbulence in Disk Drives. In *Proceedings of Advances in Cryptology – CRYPTO'94*, volume 839 of *LNCS*, pages 114–120, 1994.

[DRV02] N. Dedic, L. Reyzin, and S. Vadhan. An Improved Pseudorandom Generator Based on Hardness of Factoring. In *Third Conference on Securuty in Communication Networks (SCN '02)*, 2002. Available at: http://eprint.iacr.org/2002/131.pdf [online; accessed 2009].

[ECS94] D. Eastlake, S. Crocker Cybercash, and J. Schiller. RFC1750: Randomness Recommendations for Security. MIT, December 1994.

[ECS05] D. Eastlake, S. Crocker, and J. Schiller. RFC4086 – Randomness Requirements for Security, 2005. Available at: http://www.ietf.org/rfc/rfc4086.txt [online; accessed 2009].

[EHK+03] M. Epstein, L. Hars, R. Krasinski, M. Rosner, and H. Zheng. Design and Implementation of a True Random Number Generator Based on Digital Circuit Artifacts. In *Proceedings of the 5th Workshop Cryptographic Hardware and Embedded Systems (CHES) 2003*, volume 2279 of *Lecture Notes in Computer Science*, pages 152–165. Springer, 2003.

[Ell95] C. Ellison. IEEE. P1363 Appendix E – Cryptographic Random Numbers, 1995. Available at: http://theworld.com/~cme/P1363/ranno.html [online; accessed 2009].

[EVB01] J. Eberspaecher, H.-J. Voegel, and C. Bettstetter. *GSM Switching, Services, and Protocols*. Wiley, 2001. ISBN 978-0471499039.

[FCRV03] A. Fort, F. Cortigiani, S. Rocchi, and V. Vignoli. Very High-Speed True Random Noise Generator. In *Analog Integrated Circuits and Signal Processing*, volume 34, pages 97–105, 2003.

[FMC84] R. C. Fairfield, R. L. Mortenson, and K. B. Coulthart. An LSI Random Number Generator (RNG). In *Proceedings of CRYPTO 84 on Advances in cryptology*, pages 203–230, 1984. Available at: http://dsns.csie.nctu.edu.tw/research/crypto/HTML/PDF/C84/203.PDF [online; accessed 2009].

[FS03] N. Ferguson and B. Schneier. *Practical Cryptography*. John Wiley & Sons, first edition, 2003. ISBN 0-471-22357-3.

[Gen00] R. Gennaro. An Improved Pseudo-Random Generator Based on the Discrete Logarithm Problem. In *Proceedings of the 20th Annual International Cryptology Conference on Advances in Cryptology*, pages 469–481, 2000. Available at: http://www.research.ibm.com/security/newprng.ps [online; accessed 2009].

[Gir08] D. Giry. Keylength.com – Cryptographic Key Length Recommendation, December 2008. Available at: http://www.keylength.com/ [online; accessed 2009].

[Gol90] O. Goldreich. A Note on Computational Indistinguishability. In *Information Processing Letters*, volume 34, pages 277–281, May 1990.

[Gut98] P. Gutmann. Software generation of practically strong random numbers. In *Proceedings of the 7th USENIX Security Symposium*, pages 243–257. USENIX Association, January 1998. Available at: http://www.usenix.org/publications/library/proceedings/sec98/full_papers/gutmann/gutmann.pdf [online; accessed 2009].

[Gut04] P. Gutmann. Cryptographic Security Architecture Design and Verification, 2004. ISBN 978-0-387-95387-8.

[GW96] I. Goldberg and D. Wagner. Randomness and the Netscape Browser. In *Dr. Dobb's Journal, Special issue on Encoding: Encryption, Compression, and Error Correction*, 1996.

[HCD97] W. T. Holman, J. A. Connelly, and A. B. Dowlatabadi. An Integrated Analog/Digital Random Noise Source. *IEEE Transactions on Circuits and System I: Fundamental Theory and Applications*, 44(6):204–218, June 1997.

[Hel06] P. Hellekalek. pLab: Theory and Practice of Random Number Generation, research project led by Peter Hellekalek at the Mathematics Department of the University of Salzburg, 2006. Available at: http://random.mat.sbg.ac.at/ [online; accessed 2009].

[Hen05] P. Henkel. Port of Rijndael Block Cipher to Symbian OSs. 2005. Available at: http://www.newlc.com/AES-Encryption.html [online; accessed 2009].

[HSS04] J. C. Hernndez, J. M. Sierra, and A. Seznec. The SAC Test: A New Randomness Test, with Some Applications to PRNG Analysis. In *ICCSA 2004*, volume 3043 of *Lecture Notes in Computer Science*, pages 960–967. Springer, 2004.

[HW03] P. Hellekalek and S. Wegenkittl. Empirical evidence concerning AES. In *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, pages 322–333. ACM Press, 2003. Available at: http://random.mat.sbg.ac.at/ftp/pub/publications/peter/aes_sub.ps [online; accessed 2009].

[IZ89] R. Impagliazzo and D. Zuckerman. How to recycle random bits. In *Proceedings of the 30th IEEE Symposium on Foundations of Computer Science*, pages 248–253, 1989.

[Jab96] D. Jablon. Strong password-only authenticated key exchange. *ACM SIGCOMM Computer Communication Review*, 26(5):5–26, October 1996. Available at: http://www.jablon.org/jab96.pdf [online; accessed 2009].

[JK99] B. Jun and P. Kocher. The Intel Random Number Generator. Cryptography Research, 1999. Available at: http://www.cryptography.com/resources/whitepapers/IntelRNG.pdf [online; accessed 2009].

[JMV01] D. Johnson, A. Menezes, and S. Vanstone. The Elliptic Curve Digital Signature Algorithm (ECDSA). Certicom Research, May 2001. Available at: http://www.comms.scitech.susx.ac.uk/fft/crypto/ecdsa.pdf [online; accessed 2009].

[KC02] D. J. Kinniment and E. G. Chester. Design of an On-Chip Random Number Generator using Metastability. In *Proceedings of the 28th European Solid-State Circuit Conference (ESSCIRC) 2002*, pages 595–598, September 2002. Available at: http://www.staff.ncl.ac.uk/david.kinniment/Research/papers/ESSCIRC2002.PDF [online; accessed 2009].

[Kel05] S. S. Keller. NIST-recommended random number generator based on ANSI X9.31 Appendix A.2.4 using the 3-key triple DES and AES algorithms. NIST Computer Security Division, January 2005. Available at: http://csrc.nist.gov/groups/STM/cavp/documents/rng/931rngext.pdf [online; accessed 2009].

[Ken06] J. Kennedy. Digital camera fundamentals. Andor Technolog, 2006. Available at: http://www.andor.com/pdfs/DigitalCameraFundamentals.pdf [online; accessed 2009].

[KKK+07] J. Krhovják, M. Kumpošt, J. Kůr, V. Lorenc, V. Matyáš, Z. Říha, J. Staudek, and P. Švenda. Smartcards, final report for the Czech National Security Authority, December 2007.

[KKL+08] J. Krhovják, J. Kůr, V. Lorenc, V. Matyáš, P. Pecho, Z. Říha, J. Staudek, P. Švenda, and J. Žižkovský. Smartcards, final report for the Czech National Security Authority, December 2008.

[KMH07] A. N. Klingsheim, V. Moen, and K. J. Hole. Challenges in Securing Networked J2ME Applications. In *Computer*, volume 40, pages 24–30, 2007. ISSN 0018-9162.

[KMŽ09] J. Krhovják, V. Matyáš, and J. Žižkovský. Generating random and pseudorandom sequences in mobile devices. Submitted to The First International ICST Conference on Security and Privacy in Mobile Information and Communication Systems (MobiSec '09). Springer, 2009.

[Knu97] D. E. Knuth. *The Art of Computer Programming: Seminumerical Algorithms, Volume 2*. Addison Wesley, third edition, 1997. ISBN 0-201-89684-2.

[KR02] V. Klíma and T. Rosa. Further Results and Considerations on Side Channel Attacks on RSA. In *CHES '02*, volume 2523 of *Lecture Notes in Computer Science*, pages 244–259. Springer, 2002.

[Kra90] H. Krawczyk. How to predict congruential generators. In *Advances in Cryptology-CRYPTO'89*, volume 435 of *Lecture Notes in Computer Science*, pages 138–153. Springer, 1990. Available at: http://dsns.csie.nctu.edu.tw/research/crypto/HTML/PDF/C89/138.PDF [online; accessed 2009].

[Krh06a] J. Krhovják. Analysis, demands, and properties of pseudorandom number generators. In *Santa's Crypto Get-Together '06*, pages 55–65, December 2006. ISBN 80-903083-7-6.

[Krh06b] J. Krhovják. Cryptographic random and pseudorandom number generators. In *6th Central European Conference on Cryptography – NyírCrypt '06*, June 2006.

[KSF99] J. Kelsey, B. Schneier, and N. Ferguson. Yarrow-160: Notes on the Design and Analysis of the Yarrow Cryptographic Pseudorandom Number Generator. In *Proceedings of the 6th Annual International Workshop on Selected Areas in Cryptography*, pages 13–33. Springer-Verlag, August 1999. Available at: http://www.schneier.com/paper-yarrow.ps.gz [online; accessed 2009].

[KŠM07] J. Krhovják, P. Švenda, and V. Matyáš. The sources of randomness in mobile devices. In *Proceeding of the 12th Nordic Workshop on Secure IT System*, pages 73–84. Reykjavik University, October 2007.

[KSM08] J. Krhovják, A. Stetsko, and V. Matyáš. Generating random numbers in hostile environments. In *16th Security Protocols Workshop – SPW '08*, April 2008. To appear in Springer-Verlag Lecture Notes in Computer Science.

[KŠMS07] J. Krhovják, P. Švenda, V. Matyáš, and L. Smolík. The sources of randomness in smartphones with Symbian OS. In *Security and Protection of Information '07*, pages 87–98. University of Defence, May 2007.

[KSWH98] J. Kelsey, B. Schneier, D. Wagner, and C. Hall. Cryptanalytic attacks on pseudorandom number generators. In *Fifth International Workshop Proceedings of Fast Software Encryption '98*, pages 168–188. Springer-Verlag, March 1998. Available at: http://www.schneier.com/paper-prngs.pdf [online; accessed 2009].

[Lav00] The LavaRnd Random Number Generator, 2000. Available at: http://www.lavarnd.org/ [online; accessed 2009].

[LP08] S. Laur and S. Pasini. SAS-Based Group Authentication and Key Agreement Protocols. In *Public Key Cryptography (PKC) 2008*, volume 4939 of *Lecture Notes in Computer Science*, pages 197–213. Springer, 2008.

[LS07] P. L'Ecuyer and R. Simard. TestU01: A C Library for Empirical Testing of Random Number Generators. In *ACM Transactions on Mathematical Software*, volume 33, 2007.

[Mar95] G. Marsaglia. DIEHARD Statistical Tests, 1995. Available at: http://stat.fsu.edu/pub/diehard/ [online; accessed 2009].

[MT02] G. Marsaglia and W. W. Tang. Some difficult-to-pass tests of randomness. *Journal of Statistical Software*, 7(3), 2002. Available at: http://www.jstatsoft.org/v07/i03/ [online; accessed 2009].

[MvOV01] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, August 2001. ISBN 0-8493-8523-7.

[Neu51] J. von Neumann. Various techniques used in connection with random digits. In *Monte Carlo methods*, volume 12 of *Applied Mathematics Series*, pages 36–38, 1951.

[Neu04] D. Neuenschwander. *Probabilistic and Statistical Methods in Cryptology*. Springer, 2004. ISBN 3-540-22001-1.

[Ng05] H. Ng. Simple Pseudorandom Number Generator with Strengthened Double Encryption (Cilia), 2005. Available at: http://eprint.iacr.org/2005/086.pdf [online; accessed 2009].

[NIS94a] National Institute of Standards and Technology. Federal Information Processing Standards Publication 140-1, Security Requirements for Cryptographic Modules, January 1994. Available at: http://csrc.nist.gov/publications/fips/fips140-1/fips1401.pdf [online; accessed 2009].

[NIS94b] National Institute of Standards and Technology. Federal Information Processing Standards Publication 186, DIGITAL SIGNATURE STANDARD (DSS), May 1994. Available at: http://www.itl.nist.gov/fipspubs/fip186.htm [online; accessed 2009].

[NIS00] National Institute of Standards and Technology. Federal Information Processing Standards Special Publication 800-22. A Statistical Test

Suite for Random and Pseudorandom Number Generators for Cryptographic Applications, October 2000. Revised May 2001.

[NIS01] National Institute of Standards and Technology. Federal Information Processing Standards Publication 140-2, Security Requirements for Cryptographic Modules, May 2001. Available at: http://csrc.nist.gov/publications/fips/fips140-2/fips1402.pdf [online; accessed 2009].

[NIS06] Digital Signature Standard (DSS). Draft Federal Information Processing Standards Publication 186-3, National Institute of Standards and Technology. NIST Computer Security Division, May 2006.

[NIS07] National Institute of Standards and Technology. Federal Information Processing Standards Publication 140-3 (Draft), Security Requirements for Cryptographic Modules, July 2007. Available at: http://csrc.nist.gov/publications/fips/fips140-3/fips1403Draft.pdf [online; accessed 2009].

[OP00] T. Okamoto and D. Pointcheval. PSEC-3: Provably Secure Elliptic Curve Encryption Scheme. In *IEEE P1363a*, May 2000. Available at: http://grouper.ieee.org/groups/1363/P1363a/contributions/psec3v2.pdf [online; accessed 2009].

[PC96] C. S. Petrie and J. A. Connelly. Modeling and Simulation of Oscillator-Based Random Number Generators. In *International Symposium on Circuits and Systems*, volume 6, pages 324–327, May 1996.

[PC00] C. S. Petrie and J. A. Connelly. A Noise-Based IC Random Number Generator for Applications in Cryptography. In *IEEE Transactions on Circuits and Systems*, volume 47, pages 615–621, May 2000.

[Pir05] C. Piras. RaBiGeTe: Random Bit Generators Tester, 2005. Available at: http://www.webalice.it/cristiano.pi/rabigete/ [online; accessed 2009].

[PKS00] M. G. Parker, A. H. Kemp, and S. J. Shepherd. Fast Blum-Blum-Shub Sequence Generation Using Montgomery Multiplication. *IEEE Proceedings of Computers and Digital Techniques*, 147(4):252–254, July 2000.

[Qua04] Id Quantique. Random Numbers Generation using Quantum Physics. White paper, 2004. Available at: http://www.idquantique.com/products/files/quantis-whitepaper.pdf [online; accessed 2009].

[Rén60] A. Rényi. On measures of information and entropy. In *Proceedings of the 4th Berkeley Symposium on Mathematics, Statistics and Probability*, pages 547–561. University of California Press, 1960.

[Riv92] R. Rivest. RFC 1321 – The MD5 Message-Digest Algorithm, 1992. Available at: http://www.ietf.org/rfc/rfc1321.txt [online; accessed 2009].

[RSA99] RSA Security. PKCS#5: Password-Based Cryptography Standard, ver. 2.0, 1999. Available at: ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-5v2/pkcs5v2-0.pdf [online; accessed 2009].

[RSA02] RSA Security. PKCS#1: RSA Cryptography Standard, ver. 2.1, 2002. Available at: ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-1/pkcs-1v2-1.pdf [online; accessed 2009].

[Sch01] W. Schindler. Efficient Online Tests for True Random Number Generators. In *Proceedings of the Third International Workshop on Cryptographic Hardware and Embedded Systems*, pages 103–117, 2001.

[SF07] D. Shumow and N. Ferguson. On the Possibility of a Back Door in the NIST SP800-90 DualEcPrng, 2007. Available at: http://rump2007.cr.yp.to/15-shumow.pdf [online; accessed 2009].

[Sha48] C. E. Shannon. A Mathematical Theory of Communication. In *The Bell System Technical Journal*, 1948.

[Sha02] R. Shaltiel. Recent developements in explicit constructions of extractors. In *Bulletin of the EATCS*, pages 67–95, 2002.

[Sho96] V. Shoup. On fast and provably secure message authentication based on universal hashing. In *Proceedings of Crypto '96*, pages 313–328, 1996. Available at: http://www.shoup.net/papers/macs.pdf [online; accessed 2009].

[Sho01] V. Shoup. OAEP Reconsidered. In *Crypto '01*, 2001. Available at: http://www.shoup.net/papers/oaep.pdf [online; accessed 2009].

[SJUH04] K. Song-Ju, K. Umeno, and A. Hasegawa. Corrections of the NIST Statistical Test Suite for Randomness, January 2004. Available at: http://eprint.iacr.org/2004/018.pdf [online; accessed 2009].

[SK01] T. Stojanovski and L. Kocarev. Chaos-Based random number generators – Part I: Analysis. *IEEE Transactions on Circuits System-1: Fundamental Theory and Applications*, 48(3):281–288, 2001.

[SMH05] K. I. F. Simonsen, V. Moen, and K. J. Hole. Attack on Suns MIDP Reference Implementation of SSL. In *10th Nordic Workshop on Secure IT Systems*. Tartu University, 2005.

[Sot00] J. Soto. Statistical Testing of Random Number Generators. National Institute of Standards and Technology, 2000. Available at: http://csrc.nist.gov/groups/ST/toolkit/rng/documents/nissc-paper.pdf [online; accessed 2009].

[Tre99] L. Trevisan. Construction of extractors using pseudorandom generators (Extended Abstract). In *31st ACM Symposium on Theory of Computing*, pages 141–148, 1999.

[Tse05] Y.-M. Tseng. An Improved Conference-Key Agreement Protocol with Forward Secrecy. *Informatica*, 16(2):275–284, 2005.

[Vau02] S. Vaudenay. Security Flaws Induced by CBC Padding – Applications to SSL, IPSEC, WTLS. . . . In *Eurocrypt '02*, volume 2332 of *Lecture Notes in Computer Science*, pages 534–545. Springer, 2002. Available at: http://lasecwww.epfl.ch/pub/lasec/doc/Vau02a.ps [online; accessed 2009].

[VHKK08] I. Vasyltsov, E. Hambardzumyan, Y. Kim, and B. Karpinskyy. Fast Digital TRNG Based on Metastable Ring Oscillator. In *Cryptographic Hardware and Embedded Systems (CHES) 2008*, pages 164–180, 2008.

[VM01] J. Viega and G. McGraw. *Building Secure Software: How to Avoid Security Problems the Right Way*. Addison Wesley Professional, 2001. ISBN 0-2017-2152-X.

[WF01] S. Walker and S. Foo. Evaluating Metastability in Electronic Circuits for Random Number Generation. In *Proceedings IEEE Computer Society Workshop on VLSI*, pages 99–101. IEEE Computer Society, April 2001.

[Wu97] T. Wu. The Secure Remote Password protocol. In *Proceedings of the 1998 Internet Society Symposium on Network and Distributed Systems Security*, pages 97–111, November 1997. Available at: ftp://srp.stanford.edu/pub/srp/srp.ps [online; accessed 2009].

[Xia07] Y. Xiao. *Link Layer Security in Wireless LANs, Wireless PANs, and Wireless MANs*. Springer, 2007. ISBN 978-0387263274.

[Zen04] E. Zenner. *On Cryptographic Properties of LFSR-based Pseudorandom Generators*. PhD thesis, Mannheim University, 2004. Available at: http://th.informatik.uni-mannheim.de/people/zenner/pub/phdzenner.pdf [online; accessed 2009].

[ZG06] T. Reinman Z. Gutterman, B. Pinkas. Analysis of the Linux Random Number Generator. 2006. Available at: http://eprint.iacr.org/2006/086.pdf [online; accessed 2009].