ROMANIAN JOURNAL OF INFORMATION SCIENCE AND TECHNOLOGY
Volume **13**, Number 3, 2010, 219–240

Randomness in Digital Cryptography: A Survey

Kinga MARTON, Alin SUCIU, Iosif IGNAT

Technical University of Cluj-Napoca,
Computer Science Department
26-28 George Bariţiu street, Cluj-Napoca, Romania
E-mail: {kinga.marton,alin.suciu,iosif.ignat}@cs.utcluj.ro

Abstract. Digital cryptography relies greatly on randomness in providing the security requirements imposed by various information systems. Just as different requirements call for specific cryptographic techniques, randomness takes upon a variety of roles in order to ensure the proper strength of these cryptographic primitives. The purpose of the survey is to emphasize the importance of randomness in digital cryptography by identifying and highlighting the many roles random number sequences play and to carry forth the significance of choosing and integrating suitable random number generators as security flaws in the generator can easily compromise the security of the whole system.

1. Introduction

The security of cryptographic systems is strongly related to randomness, as the output of these systems must be seen by any adversary as a sequence of random values carrying the secret but revealing absolutely no clues about the precious information. Hence randomness makes it possible for cryptographic systems to verify Kerckhoffs' principle [53] which affirms that the security of cryptographic techniques must rely solely on the knowledge of the key – in an environment where all the complexity introduced by deterministic confusion and diffusion methods becomes easily solvable as soon as the algorithm becomes public.

But what are the properties that make random sequences indispensable in cryptography? Although a standard formal definition is missing, randomness refers to the outcome of a probabilistic process that produces independent, uniformly distributed and unpredictable values that can not be reliably reproduced [79]. Such

a process called "randomness generator" satisfies the above properties to a certain extent, thereby the quality of the sequences produced by various generators are different. Hence, if randomness had offered the key to many problems of cryptography, it certainly provides a series of difficult questions that need to be addressed, like what kind of generator should be used for a specific task, how can randomness be measured and compared and what are the risks of using weak randomness sources.

Cryptography comes with a wide range of techniques in order to provide solutions for different security requirements – and these techniques require random sequences for many different purposes, therefore randomness takes upon a variety of roles that our survey tries to identify and highlight. However, before presenting these many faces of randomness lets focus a bit more on what randomness is.

The major ingredients for randomness are *unpredictability* – or lack of predictability, *independency of values* – or lack of correlation, and *uniform distribution* – or lack of bias. Some of the properties of a random sequence are statistical and hence can be measured by using various statistical randomness tests. However, the most important problem in connection with randomness is the lack of certainty. By extensive analysis and thorough testing one can get a high confidence in the generator but cannot be absolutely sure. This is why there is a wide range of statistical test suites (NIST [66], Test01 [59], Diehard [61], ENT [94], etc.) that try to rule out sequences which do not verify certain statistical properties, yet can never guarantee perfect randomness.

The statistical analysis of the random sequences is very important but alongside the application of statistical tests that assess the outcome of a randomness generator, there must be a serious analysis of the source the generator extracts randomness from. Consider the result of a statistical test suite that would indicate good random properties of a sequence but then it is pointed out that the sequence was in fact built from Pi's digits - no unpredictability there. Though applicable in various situations where security is not an issue, random sequences used in cryptography must be unpredictable, irreproducible and should not allow the adversary to learn or predict former or subsequent values.

Depending on the nature of the randomness source, generators are classified in three categories presented briefly in the following.

A. *True random number generators (TRNG)*, where the source is a natural physical phenomenon and the properties of independence and unpredictability of the generated values are guaranteed by physical laws. While TRNGs offer the highest level of entropy [84, 52] (meaning measure of uncertainty: number of symbols that have to be known in order to remove uncertainty associated with a random variable, and also information content: number of symbols necessary to encode all possible values of a variable) they do not necessarily present uniform distribution and most of them need to be filtered (post processed) [64, 10, 29] in order to reduce possible bias - tendency towards a particular value, and correlation, and make the output more similar to perfectly random sequence. Nonetheless true random sources need specialized hardware and many are slow and tend to become impractical. Examples of TRNGs are generators based on radioactive decay [93], thermal noise [19], quantum mechanics [75, 90], lavalamp images

[58], jitter extraction [85] etc.

B. *Unpredictible random number generators (URNG)*: are based on the unpredictability inherent to human computer interaction and on the undeterminism introduced by the complexity of the underlying phenomenon (e.g. Linuxs /dev/random [44], HAVEGE [82], Data flow entropy collector [91], etc.). URNGs use easily available devices, like computer components, as entropy sources and provide a high level of randomness, but special attention must be given to the way components built for being deterministic can be used for generating randomness.

C. *Pseudorandom number generators (PRNG)*, where the source of randomness is a random initial value, called *seed*, which is expanded by means of a deterministic recursive formula, providing a modality for generating random sequences using only software methods. As a result the unpredictability level resumes to the unpredictability of the seed value and the output is completely determined by the starting state of the generator an therefore by the seed. Still, the practical features of PRNGs such as high generation speed, good statistical results and no need for additional hardware devices, made these generators very attractive and are the most widely used random number generators in cryptographic systems. However, the reduced level of unpredictability that the majority of PRNGs show is not sufficient for security applications because these can be easily compromised by using a low quality randomness source, as many successful attacks have demonstrated. Thus there are ongoing research efforts that aim to reveal secure ways of using PRNGs in cryptography and this search has brought about several PRNG designs that are considered cryptographically secure (CSPRNG) like PRNGs built on cryptographic primitives such as hash functions [69] or block ciphers [73, 6], mathematical problems considered to be extremely difficult such as Elliptic curve generators [70], or integer factorization such as the Blum-Blum-Shub generator [11].

A more detailed introduction to what randomness means and what the advantages and disadvantages of various PRNGs are, is offered in [55]. Discussion on different types of randomness sources can be found in [30], and in [27] the authors give recommendations for choosing suitable generators for security applications and describe several attacks on PRNGs emphasizing the benefits of hardware sources of true random numbers. If hardware sources are not available, software techniques have to be applied and Gutman in [42] alongside with presenting problems in current PRNG designs, provides specific details and guidelines on the generation of practically strong randomness using PRNGs pointing out the importance of preprocessing the input and postprocessing the output of PRNGs. In [51] the authors present the major vulnerabilities, enumerate several attacks on PRNGs and propose a model for building secure PRNGs highlighting the essential ingredients such as a constant reseeding process with highly unpredictable input and frequently changing internal state to avoid compromises, and draw the conclusion that some random number generator designs may be secure for a specific purpose but insecure for another, hence security analysis

has to be made for each situation the generator is applied in. A brief introduction to the importance of random numbers in data security systems is provided in [26] focusing mainly on the need of using hardware random number generators. Gennaro offers a very meaningful glance on the practical uses of randomness in cryptography [39].

2. Cryptographic Keys

One of the most important roles randomness plays in cryptography is represented by cryptographic keys which determine the transformation of the plaintext into ciphertext and vice versa.

Considering that both the encryption and the decryption algorithms are publicly known together with all the ciphertexts transmitted between the sender and receiver, the security of the whole cryptosystem is dependant on how the key information is managed: generated, agreed on, applied, stored and destroyed. The knowledge of the key entails the access to the secret message, thus the choice of the key space and the key derivation method is critical.

Cryptographic keys must be unpredictable for the adversary meaning a high information content and high uncertainty, and as we saw in the previous section the measure of these properties is *entropy*. Yet high entropy is not enough, and a very good example in this direction is a compressed file which besides its high entropy value has a highly structured content. Thus sequences chosen for cryptographic keys must also exhibit independency of values, uniform distribution and irreproducibility. As a result what cryptography needs for its keys is randomness. But randomness comes in many flavors and deciding on a certain source of randomness for a given application is a difficult task, considering the quality and quantity of randomness necessary for a key to withstand possible attacks. Furthermore, aspects such as the performance implications of working with a long key sequence and the effort of managing such a key also have to be taken into account.

Properties that a key sequence can provide, classify the cipher using these keys in different security categories. The most demanding requirements on the quality of the key sequence is stated by unconditional security. The most notable work in this area is Shannons demonstration [84] that a Vernam cipher which uses a perfectly random key and applies it only once, known as one time pad (OTP), is unbreakable.

In situations where the conditions of perfect security can not be met or dealt with, mainly because the management of a large random key is impractical, other probabilistic algorithms that need shorter keys have to be used instead. In the following we point out the way random keys are applied in several cryptographic techniques.

2.1. Keys in Symmetric Cryptography

In symmetric cryptosystems both encryption and decryption are performed using the same secret key shared by the communicating parties. Block ciphers break up the plaintext (M) in fixed size blocks (b_i) and encrypt one block at a time using complex encryption (and decryption) functions parameterized on a fixed size key (K). Stream

ciphers combine each symbol of the key (K_i) with a different symbol (d_i) of the plaintext (M) at a time using very simple encryption (and decryption) functions and therefore the security rests on the generated keystream.

Block ciphers	Stream ciphers
$E_K(M) = E_K(b_1)E_K(b_2)...$	$E_K(M) = E_{K_1}(d_1)E_{K_2}(d_2)...$

The initial random secret key, K, of the stream and block ciphers is usually expanded through the stream ciphers' *key scheduler* or the block ciphers' *key expansion algorithm* in order to facilitate the encryption of every symbol or block of the plaintext under different subkeys, a necessary aspect considering that the length of the initial key is usually much smaller that the length of the message to be encrypted. Consequently, these expansion algorithms are in essence pseudorandom number generators — PRNGs.

Stream ciphers do not enjoy a well established standard like block ciphers do (AES), and with the intention to identify new stream ciphers that would be suitable for adoption as a standard, the ECRYPT Stream Cipher Project – eSTREAM [31] has started a competition in 2004. After rigorously analyzing the submitted stream ciphers, a portfolio of the most promising ciphers was published. The submitted ciphers also emphasize the fact that *synchronous* stream ciphers have the main focus from researchers in detriment of *selfsynchronous* stream ciphers.

The keystream of the synchronous stream ciphers is generated independently from the plaintext stream, as a result the ciphers' inner state can not be influenced by an attacker and are the most widely used category of stream ciphers, but require additional mechanisms for synchronization and error correction. Some modern designs may rely on computer networks which provide these properties (like Phelix [31]) and can significantly reduce synchronization problems. Figure 1 depicts the general structure of a synchronous stream cipher where the keystream generator is in fact a PRNG – usually a pseudorandom bit generator (PRBG) and the key is the seed of the PRBG.

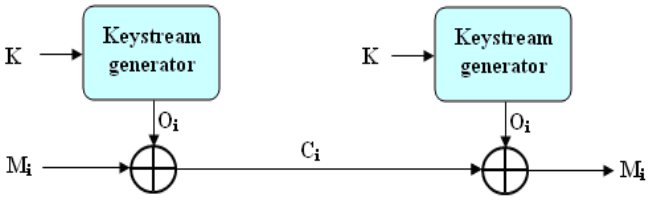

Fig. 1. General structure of a synchronous stream cipher (encryption and decryption).

Self-synchronous stream ciphers on the other hand, generate the keystream from the current key and a fixed number of symbols of the ciphertext (see Fig. 2), and as a result each symbol of the message affects every subsequent symbol of the ciphertext. Consequently resynchronization is performed automatically.

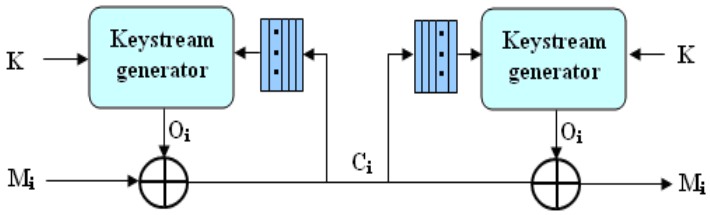

Fig. 2. General structure of a selfsynchronous stream cipher (encryption and decryption).

The block structure of a keystream generator is presented in Fig. 3, where the main components are the input consisting of the secret key and possibly an initialization vector (IV) (see Section 3), the internal state, the function that updates the internal state at each step and the filtering function that processes the internal state in order to output the keystream.

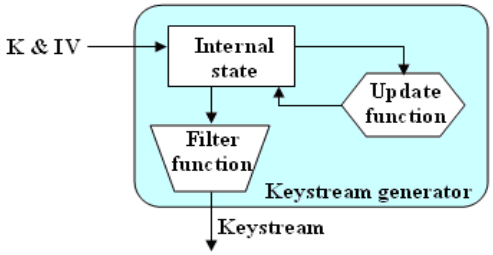

Fig. 3. Block structure of a keystream generator.

Linear Feedback Shift Registers (LFSRs) are the most widely used building blocks for keystream generation in stream ciphers because of their very efficient hardware implementation, good statistical properties, large period, large linear complexity and ease of analysis using algebraic techniques. The secret key in these ciphers is the LFSRs initial state. Figure 4 depicts the general structure of an LFSR.

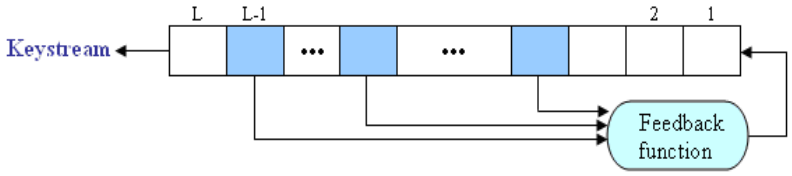

Fig. 4. General structure of a LFSR.

Though LFSRs can guarantee a large period ($2^L - 1$ for an L stage FSR) the high linearity makes the output sequence of an L-stage LFSR easily predictable using the Berlekamp Massey algorithm [8] if there are more than 2L output bits available, and thus LFSRs should never be used for keystream generation in their original form. But the significant benefits of cost-effectiveness have led to the design of various methods

for achieving a large period, low correlation with the input bits and for destroying the linear properties. These designs use one or more of the following methods:

- *combining methods*: combines the output of several LFSRs in parallel using a nonlinear combining function (e.g. Geffe generator). The output of LFSRs can be input to an adder with carry adding a nonlinear memory (e.g. summation generator – see Fig. 5, SNOW, E0 in Bluetooth);

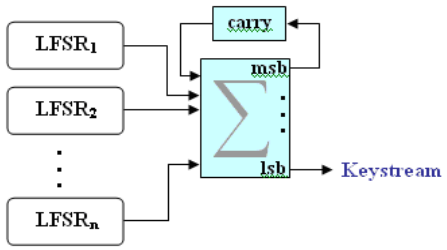

Fig. 5. The summation generator.

- *filtering methods*: use one single LFSR and compute the keystream by filtering the bits at several positions of the LFSR using a boolean function (e.g. SFINKS [31], Sober – see Fig. 6);

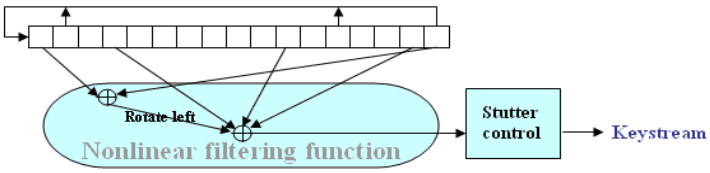

Fig. 6. General structure of a Sober cipher keystream generator.

- *irregular clocking methods* (stop and go control): clocking is performed an unpredictable number of times depending on an internal state (i.e. alternating step generator – depicted in Fig. 7, A5 in GSM, Shrinking and self-shrinking generators, and the ciphers Pomaranch, DECIM, Mickey-128 [31]).

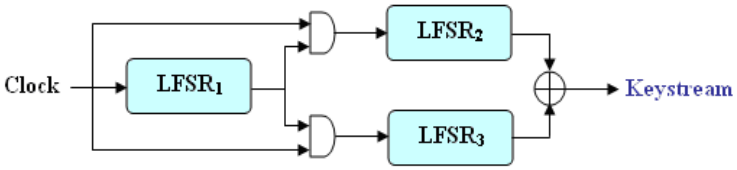

Fig. 7. General structure of an alternating step generator.

LFSRs are not suitable for software implementation due to their slowness, namely each iteration of the update function produces only one bit of keystream, therefore alternative designs have to be used that are not based on LFSRs, such as NLFSRs (NonLinear Feedback Shift Registers). NLFSRs are the building blocks of stream ciphers like Dragon, which uses a word based NLFSR in addition to a nonlinear filtering function and memory, Trivium, Achterbachn and Grain, and furthermore, of the well known stream ciphers RC4 – see Fig. 8 and SEAL).

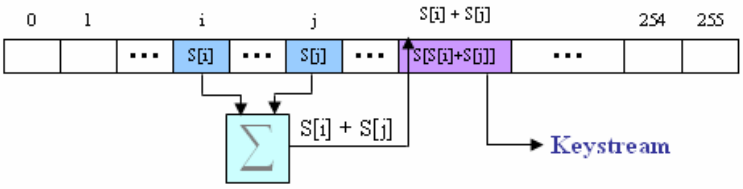

Fig. 8. Structure of RC4 key scheduling process.

Block ciphers are the most popular cryptographic primitives, thanks to the standardization of DES followed by AES, but also due to the fact that block ciphers constitute some of the fundamental building blocks for pseudorandom number generators, stream ciphers, hash functions and message authentication codes.

Most block ciphers apply a simple encryption function, round function, iteratively a certain number of times - called rounds, using round keys derived from the initial secret key using a key expansion algorithm. There are two main types of iterated ciphers: Feistel ciphers (e.g. DES, Blowfish, CAST-128) and Substitution-Permutation network ciphers (e.g. AES, Serpent).

The key expansion algorithm computes the round keys from the initial random secret key K, and can be a very simple function like splitting the initial key into a number of pieces according to the number of rounds (e.g. TEA) or more elaborate like in DES, where the initial key is divided into two halves which are rotated left with one or two bits and in each round 24 bits are extracted from each half creating a 48 bit round key; even more complex key schedulers are applied in modern designs such as in AES. Figure 9 presents the way key expansion functions are integrated in the encryption process of one plaintext block using the AES cipher.

As a result, besides the highly random key sequence, special attention must be given to the round key derivation method because a weak key expansion algorithm can lead to the compromise of an otherwise secure cipher. This is the reason why strong key schedulers are needed to generate subkeys such that relations between the subkeys of the same round and of any rounds is infeasible to predict.

In order to evaluate the randomness characteristics of AES candidate algorithms, NIST conducted an analysis on the suitability of candidate algorithms as random number generators [87, 88] applying several statistical tests on the output (ciphertext) produced by the ciphers with the goal of identifying whether the ciphertext could be computationally distinguished from the output of a truly random source. The results indicate that the five finalist algorithms (Mars, RC6, Rijndael, Serpent, Twofish) show no deviation from randomness. At the same time the importance of using several

rounds is highlighted based on the fact that partial round testing on the output of each round of the tested algorithms show that random output is achieved only after the first few rounds (Mars at round 6, RC6 at round 4, Rijndael at round 3, Serpent at round 4, Twofish at round 2) when using 128 bit random keys generated by a cryptographically secure pseudorandom number generator Blum Blum Shub (BBS) and low density plaintext blocks.

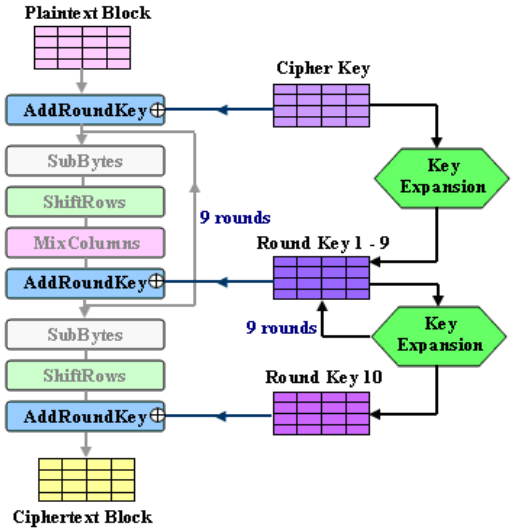

Fig. 9. Illustration of AES subkeys generated by the key expansion algorithm.

Another class of symmetric algorithms where cryptographic keys are used is given by *Message Authentication Codes* (MACs) which provide data integrity and authenticity of the message they accompany. MACs receive as input the arbitrary length message and a secret key and compute the fixed length MAC value, or checksum, of the message that is dependent on every bit of the message and the key. Being generated and verified using the same secret key, MACs can be used only in symmetric systems where parties trust each other, and thus they are different from digital signatures and can not provide non-repudiation. Though MACs provide weaker guarantees than digital signatures, they are widely used in many banking applications, internet security etc. because they are significantly faster and require fewer resources than digital signatures.

As in the case of encryption, unconditional security can be achieved with MACs using one time truly random secret keys [89] resulting in One time MACs (OTM) which can guarantee that for each (MAC, message) pair each value is equally likely to be the MAC value of the message. Similar to OTP ciphers, the one time key and the MAC value have to be the same size as the message, which in case of long messages is impractical. This leads users towards computationally secure MACs which are usually constructed from other symmetric cryptographic primitives like cryptographic hash

functions, block ciphers, stream ciphers or universal hashing and extract their security level from the security of the underlying primitives.

Hash based MACs called HMACs [4, 5] rely on unkeyed hash functions like MD5 or the SHA family of hash functions. The importance of the random key is evident considering that general hash functions do not require a random input, consequently anyone could generate a suitable checksum for the message as the hash function is public. The secret random key offers the possibility of using cryptographic hash functions for authentication on insecure channels. Computing the message authentication code of a plaintext (M) using HMAC implies the use of a hash function H, the random key K brought to the same size as the block length of the input of the hash function. The mathematical formula of the performed operation (where || stands for the append operation) is presented in [33]:

$$HMAC_K = H((K \oplus opad)||H((K \oplus ipad)||M)) \; . \tag{1}$$

The security of HMAC functions is dependent on the length and quality of the random cryptographic key that is generally considered to be in the domain of $[L/2, L]$ where L is the length of the output block produced by the hash function. The security of hash functions is dramatically affected by collisions, yet HMACs are less exposed to these types of attacks on the underlying hash functions as presented in [56].

Block cipher based MACs can be more suitable for information systems where block ciphers are used in other purposes as well as for authentication (like encryption) [68] and these MACs compute the checksum from the ciphertext produced by the last block of the cipher. The most popular block cipher based MACs are the OMAC1 included in NIST specification as CMAC that is an improved version of CBC-MAC (also called XCBC) [40].

Stream cipher based MACs generally rely on the ciphers key scheduling algorithm and output that is subject to a compression function like in [2], a post mixing algorithm (like in Phelix [31]) or a shift register (like in Sfinks [31]).

Another very important issue regarding the use of cryptographic keys in providing authentication as well as confidentiality using symmetric primitives is the message span of the key [68]. It is not sufficient to choose a high quality random key but it is also very important to limit the message span of a single key in order to reduce the possibility of an adversary to find several messages encrypted with the same key or in case of MACs, having the same checksum.

Furthermore, there are several studies on whether imperfect randomness might provide perfect security. The work of McInnes and Pinkas [60] concludes that private key cryptosystems in which parties do not have access to perfect randomness are not secure but combining a public source of perfect randomness with the slightly random key can provide perfect security. The ongoing research of Dodis et al. [24, 22] relates to the extent to which the security of cryptographic primitives can rely on imperfect sources of randomness. They focus mainly on the randomness requirements for secret key encryption and authentication and conclude that suitable sources are in between the domain of extractable sources – nearly perfect randomness, such as true random sources that generate highly independent output and are processed with bias removing

methods like the ones described in [64, 10, 29] – and simulatable sources - week random sources, which can provide slightly random bits only applying certain relaxations.

2.2. Keys in asymmetric cryptography

Random keys take a different form in asymmetric cryptography, where instead of using the same shared secret key in both encryption and decryption (as symmetric algorithms do), the communicating parties each have a mathematically related private - public key pair, one used for encryption and the other one for decryption, with the condition that it is computationally infeasible to determine the private key knowing the related public key.

The security of public key cryptography relies on computationally difficult problems [62] like integer factorization (e.g. RSA, Rabin), discrete logarithm problem (e.g. ElGamal) or quadratic residues problem (e.g. Goldwasser-Micali) – problems that are parameterized by random values.

Considering the RSA key generation algorithm random numbers play a very important role in selecting the two prime numbers p and q in such a way that factoring $n = pq$ is computationally infeasible. Thereby the primes have to be approximately the same length so that elliptic curve factoring can be avoided, but values should not be too close, as this would make the factoring operation efficient. Randomly generated primes can provide the unpredictability, similar length and suitable difference for this algorithm. Furthermore it has been shown that random primes offer approximately the same level of security in the RSA algorithm as the recommended [77] strong primes do [62].

Unlike the deterministic encryption algorithms applied by RSA, Rabin and Knapsack, there are several public key algorithms that use randomness in the encryption phase as well as in the key generation, like ElGamal, Goldwasser-Micali or McEliece, in order to overcome the limitation of producing the same ciphertext every time the same plaintext is encrypted under a fixed public key. Therefore the latter are called probabilistic encryption algorithms, while the former, the deterministic algorithms, can be transformed into randomized encryption algorithms (see section **??**) increasing the cryptographic security, but generally these schemes are not provably secure.

Asymmetric algorithms provide authentication and non-repudiation by using digital signatures where random sequences are used in a similar manner to public key encryption algorithms with the observation that private keys are used for signing and public keys for verifying the signature. These techniques rely on public key algorithms and consequently can be classified in deterministic signature schemes (i.e. RSA, Rabin) and randomized signature schemes (DSA, ElGamal, Feige-Fiat-Shamir).

3. Initialization Vectors

Initialization vectors (IVs) are used in both stream ciphers and block ciphers, and though the implementation differs, the goal of applying IVs is the same, namely to ensure that the ciphers produce a unique output even under the same encryption key,

in order to avoid the laborious work of rekeying.

When using synchronous stream ciphers, the sender and receiver need to be synchronized in order to decrypt the ciphertext, but any symbol deletion or insertion (results of an attack or a transmission error) in the ciphertext leads to desynchronization of the communication. For the purpose of resynchronization ciphers are frequently reset to a value known to both parties but if the ciphers inner state is reset to the same secret key, the security of the cipher can be highly compromised [96], therefore users must use a new secret key, which is a costly operation, or run the initialization step with a new *IV* [57], a random value that combined with the secret key should yield the new secret internal state of the cipher.

In self synchronous stream ciphers the keystream depends on the current key and a number of symbols from the ciphertext, but at the moment of initialization no digits of ciphertext are available and therefore this role is played by the initialization vector.

In symmetric key block ciphers, where each block of plaintext is encrypted using the same secret key, initialization vectors play a very important role within confidentiality modes of operation [67] in overcoming the limitations of the Electronic Code Book (ECB) mode which transposes the input plaintext block pattern into the output ciphertext block pattern, making the confidentiality easily compromisible (see Fig. 10).

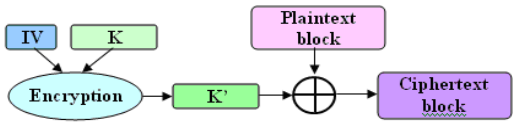

Fig. 10. Initialization vectors in block ciphers.

In Cipher-block chaining (CBC) and Cipher feedback (CFB) modes unpredictable IVs are needed while in Output feedback (OFB) mode IVs have to be unique (e.g. counter or message number) for each execution. IVs are generally the same size as the block length but not necessarily secret and may be sent to the communication party together with the ciphertext in plaintext or encrypted with the secret key. There are two recommended methods for generating unpredictable IVs: using a random number generator or encrypting a unique nonce (see section **??**) with the secret key used by the cipher.

In Counter (CTR) mode each plaintext block is encrypted (decrypted) independently, similar to ECB mode, but at each block IVs are applied in form of successive values of a counter. The counter must be unique, like the IVs in OFB mode or otherwise plaintext blocks can be easily revealed. In order to reduce the possibility of reusing a value it is recommended to produce unique IVs by adding the counter value to a random nonce.

4. Cryptographic Salt

A *cryptographic salt* is a random sequence related to the concept of nonce (see Section 6) and used generally as one of the input values for key derivation functions

(KDF) [71], which generate additional cryptographic keys based on an initial keying material such as a single master key (like in Transport Layer Security (TLS)), a password or a passphrase [92, 76], nonces exchanged in a key agreement protocol [49, 50], etc. The most popular KDFs use hashes or block ciphers [71] as derivation functions and can operate without a salt value, but using cryptographic salt adds significantly to the security of the output key as shown in [92] and are of special focus when the keying material is a password or a passphrase which otherwise would be exposed to easily performable dictionary attacks [76]. The general formula used for designing password based KDFs [92] is:

$$key = H^{(count)}(pass||salt) \ . \tag{2}$$

where || is the concatenation operator and the key is derived from the password pass and the known random value salt applying a derivation function H, such as a hash function or a block cipher, for a number of count iteration.

Salts are usually nonsecret, can be applied with several initial keying materials instead of using it just once, and the randomness quality may vary, still it is important to note that even with short, weak and publicly known salt values the security of KDF can be much improved, and the strength of the salt increases with the quality of randomness, the length and the secrecy of the value [50]. Cryptographic salt values must be independent from the keying material used to derive the output and ensured that is not manipulated by the attacker.

5. Padding

Symmetric key block ciphers (in ECB and CBC modes of operation) require complete plaintext blocks in order to perform encryption and consequently, in cases when the size of the plaintext is not a multiple of the block size, the last message block has to be filled with additional data, called padding string, and the receiver of the ciphertext must know the proper method of removing the padding (see Fig. 11). The padding strings can be composed of fixed values (e.g. 0x00 (zero padding), 0x80 followed by 0x00s [67] or the number or bytes required to fill the block [46]). However, when applied to fixed size messages the ciphertext will leak information about the message, hence in order to disguise the size of the message the padding of the last plaintext block can be done using a random sequence, in which case the number of padding bytes must be communicated to the receiver.

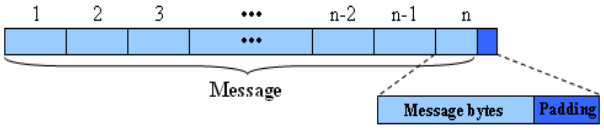

Fig. 11. Padding stream attached to the message before encryption.

Deterministic public encryption schemes can be turned into randomized algorithms by appending an independently generated random padding value to the plaintext

before encryption (sometimes called salting the message) in order to ensure that the same plaintext will not result each time in the same ciphertext under a fixed public key (see subsection 2.2) and that short messages will not be compromised by an adversary trying all the possible alternatives of the small message space. Furthermore, padding schemes associated with asymmetric encryption algorithms, like Optimal Asymmetric Encryption Padding (OAEP) [83] or Probabilistic Signature-Encryption Padding (PSEP) [23] also use random sequences and one of the major goals is to ensure that an attacker manipulating the plaintext cannot exploit the mathematical structure of the used asymmetric cipher, such as the well known property of RSA: $E(M1xM2) = E(M1)xE(M2)$. There are other attacks as well that can be defeated using random padding algorithms as described in [13].

6. Nonces and Challenges

Random *nonce* values (derived from number used once) are generally used in data-origin or entity authentication (Kerberos, challenge handshake authentication protocols) and authenticated key establishment protocols as challenges and have the goal of ensuring that a certain cryptographic operation is performed after the challenge was received so that the answer is not a replay and can check the freshness of the message (a session key) if the authentication mechanism also contains a proper message integrity scheme. A random nonce links two subsequent messages as the following example shows: one entity sends a message which includes a random nonce and the receiver will respond with a message constructed using the nonce received and thus demonstrating the freshness of the message.

Whether used to ensure message freshness and principal liveness, mutual or unilateral authentication with/without the help of a trusted third party, random nonces play a very important role alongside with cryptographic keys shared secret keys (such as in Needham Schroeder symmetric key protocol and Kerberos) or public-private key pairs, providing uniqueness and timeliness assurance. Cryptographically secure random generators have to be selected for producing nonces (or challenges) so that values are not reused and are unpredictable to attackers. Usually block cipher based pseudorandom generators are applied in OFB or counter modes [62].

Random nonce values are very important in another type of cryptographic protocols, named zero knowledge proof protocols that make possible the demonstrations of knowledge of a secret while revealing no information about the secret (other than what the verifier can deduce without the help of the prover). In satisfying this goal, zero knowledge protocols use randomness with two different purposes, firstly as challenges similar to a certain extent to asymmetric challenge-response authentication protocols, and secondly as commitments in order to prevent cheating [62, 79].

7. Blinding Values

Special classes of digital signature algorithms, called blind signature schemes (BSS), can provide additional features to authentication and nonrepudiation, like

blindness – the property of signing a message without exposing the message content to the signer, and untraceability — the property which ensures that the signer of the message is unable to associate signatures to messages even when signatures are revealed to the public. BSS is of great importance in many security sensitive applications especially in those focusing on the privacy of the user, like electronic voting (e-voting) or digital cash (e-cash) protocols [17, 36]. This technique was first introduced by Chaum in 1982 [16] and has known many variations [48] since then like RSA based blind signatures, Elliptic Curve based blind signature, ElGamal based blind signature, etc.

The basic idea in all these variations is that the message is blinded (blinding phase) a priori to signing using a blinding value (or blinding factor), a random sequence that renders the message unintelligible and is known only to the owner of the message. The signing phase is similar to the common digital signing schemes where a third party is requested to sign a message using his private key and anyone can verify the signature using the signers public key, the difference is that the received message is blinded so that the signer does not know what is the content of the message.

Blinding Phase	Alice:	**m** - message, **r** - random value, **(n,e)** public key $\mathbf{x = r^e m \bmod n}$
Signing Phase	Bank:	**x** - blinded message, **(n,d)** private key $\mathbf{y = x^d \bmod n}$
Extracting Phase	Alice:	**y** - signature of blinded message $\mathbf{s = r^{-1} y \bmod n}$

Fig. 12. RSA blind signature scheme.

In case of an RSA blind signature scheme [48] between Alice and a Bank having (n, e) as public key and the (n, d) pair as the private key, Alice first blinds her message m by a random blinding factor r. The three stages of the RSA blind signature scheme are depicted in Fig. 12.

8. Random Number Generator Security Flaws in Real-Life Applications

Goldberg and Wagner presented in [41] the security flaw within Netscape's SSL implementation caused by the weak seeding mechanism of the applied PRNG that led to the compromise of secure transactions on the Internet performed in Netscape's Web browsers. The seed constructed from the weakly random sources such as the current process id, the parent process id and the time of the day, made the attack on revealing the SSL generated secret keys easily realizable.

The security issue of Microsoft Office 97, 2000 and 2003 Word and Excel applications was discovered by Hongjun Wu [96] and it regards the misuse of the RC4 cipher, namely the re-encryption of a modified and saved document is performed using the same initialization vector (IV) and the same secret key. As a result, the two versions of the document are encrypted with identical keystreams producing two ciphertexts which XOR-ed together leak precious information about the plaintexts.

A critical security flaw discovered in 2008 in the Debian OpenSSL random number generator [7] revealed that all key material produced by the generator in more that one and a half years was highly compromised due to a Debian specific change in the OpenSSL package that caused modification in the seeding process and reduced the random input of the seed to the current process id (max 32,768). This made the random number generator very predictable [80] determining an urgent necessity of recreating key materials like SSL keys and their SSL certificates, public key certificates and session keys used in SSL/TLS connections.

Java session IDs play a very important role in providing secure digital transactions using Java Servlets, but due to a design mistake, presented in [43], the PRNG responsible for generating session IDs presented a highly predictable behavior because of the insufficient entropy in the seed value. This weakness could be easily exploited by an adversary who subsequently could impersonate legitimate clients by deducing session ID that in many cases serve as the sole information clients are identified by.

Cryptanalysis of WRNG [25], the pseudorandom number generator (based on RC4 stream cipher and SHA1 hash function) used in Microsoft Windows 2000 operating systems, uncovered deficiencies in the design of the generator resulting in efficient forward and backward security attacks. These attacks succeeded in revealing the current internal state of the generator which determines the output of 128 KB of random values, as the internal state seldom changes and the seed contains predictable values. Therefore it became possible to predict random values generated by WRNG used by different applications for security purposes.

9. Conclusions and Future Work

Random numbers are of crucial importance in almost every aspect of modern digital cryptography, responsible of the strength of cryptographic primitives in securing precious information by rendering it unknown, unguessable, unpredictable and unrepeatable for an adversary.

Our survey highlighted the many practical applications of random numbers in cryptography, starting by a short introduction in the vast domain of randomness and types of random number generators (TRNG – true random number generators, PRNG – pseudorandom number generators and URNG – unpredictable random number generators) and turning to the presentation of the many faces of randomness in cryptography as:

- cryptographic keys which determine the transformation of plaintext into ciphertext and vice versa in both symmetric and asymmetric techniques,
- initialization vectors used in symmetric stream and block ciphers in order to ensure that the ciphers produce a unique output even under the same encryption key, thus avoiding the laborious work of rekeying,
- nonces and challenges used to ensure message freshness, principal liveness, mutual or unilateral authentication and demonstrations of knowledge of a secret while revealing no information about the secret,

- cryptographic salt used generally as one of the inputs for the key derivation functions,
- padding strings which in symmetric block ciphers fill the last block of plaintext in order to bring the message to a length multiple of the block size and in asymmetric techniques serves to turn deterministic public encryption schemes into randomized algorithms,
- blinding factors used in blind signature schemes for ensuring that the message signing process is performed without exposing the message content to the signer.

The many roles randomness plays in cryptography and the importance of each role emphasizes the high degree to which cryptography relies on randomness in providing the security requirements it is designed to deliver.

Random number sequences come in different flavors and the quality and quantity of randomness applied has a great influence on the security properties and performance of the cryptographic techniques, hence choosing a suitable random number generator is a difficult task and, as briefly presented above, security flaws in random number generators can easily compromise the security of the whole system.

Although this survey focuses on the use of random numbers only in digital cryptography, we must mention here that randomness plays a very important role in Quantum Cryptography as well, which will be the object of a future survey.

Acknowledgments. This work was supported by the CNMP funded CryptoRand project, nr. 11-020/2007 and PRODOC POSDRU/6/1.5/S/5 ID 7676.

References

[1] Babbage S., Dodd M., *The stream cipher MICKEY-12*, (ECRYPT) Stream Cipher Project Report 2005/ 016.

[2] Bartosz Z. *VMPC-MAC: A stream cipher based authenticated encryption scheme*, Cryptology ePrint Archive, Report 2004/301.

[3] Bellare M., Rogaway P., *Optimal asymmetric encryption*, in *Advances in Cryptology Eurocrypt* 1994, pp. 92–111.

[4] Bellare B., Canetti R., Krawczyk H., *Message authentication using hash functions: the HMAC construction*, CryptoBytes, vol. **2**, no. 1, pp. 12-15, 1996.

[5] Bellare B., Canetti R., Krawczyk H., *Keyed Hash Functions and Message Authentication*, *Proceedings of Crypto'96*, LNCS 1109, pp. 1–15.

[6] Bellare M., Desai A., Jokipii E., Rogaway P., *A Concrete Security Treetment of Symmetric Encryption: Analysis of the DES Modes of Operation*, *Proceedings of the 38th Symposium on Foundations of Computer Science, IEEE,* 1997, pp. 394–403.

[7] Bello L., *Debian Security Advisory*, 2008, `http://www.debian.org/security/2008/dsa-1571`.

[8] Ben Atti N., Diaz-Toca G. M., Lombardi H., *The Berlekamp-Massey Algorithm revisited*, Appl. Algebra Eng. Commun. Comput, vol. **17**, no. 1, 2006, pp. 75–82.

[9] Berbain C. et al., *DECIM - A New Stream Cipher for Hardware Applications*, in *Proceedings of SKEW - Symmetric Key Encryption Workshop, Network of Excellence in Cryptology ECRYPT*, Aarhus, Danemark, 2005.

[10] Blum M., *Independent unbiased coin flips from a correlated biased source - a finite state Markov chain*, Combinatorica, vol. **6**, no. 2, pp. 97-108, 1986.

[11] Blum L., Blum M., Shub M., *A simple unpredictable pseudo-random number generator*, SIAM J. Computing, vol. **15**, no. 2, 1986, pp. 364–383.

[12] Blumenthal U., *Improvements to SEAL*, In SECURITY, 5th IBM Interdivisional Technical Symposium, 1994, pp. 976–978.

[13] Boneh D., *Twenty years of attacks on the RSA cryptosystems*, Notices of the American Mathematical Society (AMS), vol. **46**, no. 2, 1999, pp. 203–213.

[14] Braeken A., Lano J., Mentens N., Preneel B., Verbauwhede I., *SFINKS : A Synchronous Stream Cipher for Restricted Hardware Environments*, in *Proceedings of SKEW - Symmetric Key Encryption Workshop, Network of Excellence in Cryptology ECRYPT*, Aarhus, Danemark, 2005.

[15] De Cannière C., Preneel B., *Trivium - A Stream Cipher Construction Inspired by Block Cipher Design Principles*, in *Proceedings of the 9th International Conference on Information Security - ISC*, 2006, pp. 171–186.

[16] Chaum D., *Blind signatures for untraceable payment*, in *Proceedings of EUROCRYPT'82*, Advances in Cryptology, Springer-Verlag, 1983, pp. 199–203.

[17] Chaum D., *Blinding for unanticipated signatures*, in *Proceedings of EUROCRYPT'87, Advances in Cryptology*, Springer-Verlag, 1987, pp. 227–233.

[18] Chen K. et al., *Dragon: A Fast Word Based Stream Cipher*, in *Proceedings of the 7th International Conference on Information Security and Cryptology - ICISC 2004*, pp. 33–50.

[19] Chindris G., Muresan M., *Bipolar Junction Effects for High Entropy Data Harvesters*, *Proceedings of the 2008 10th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, 2008, pp. 449–452.

[20] Coppersmith D., Krawczyk H., Mansour Y., *The Shrinking Generator*, *Proceedings of Crypto 93*, Springer-Verlag, 1994, pp. 22–39.

[21] Diffie W., Hellman M. E., *New Directions in Cryptography*, *IEEE Transactions on Information Theory*, vol. **22**, 1976, pp. 644–654.

[22] Dodis Y., Spencer J., *On the (non)Universality of the One-Time Pad*, in *Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science (FOCS 2002)*, pp. 376–388.

[23] Dodis Y., Freedman M. J., Jarecki S., Walfish S., *Versatile padding schemes for joint signature and encryption*, *Proceedings of the 11th ACM conference on Computer and Communications Security*, 2004, pp. 344-353.

[24] Dodis Y., Pietrzak K., Przydatek B., *Separating Sources for Encryption and Secret Sharing*, *Third Theory of Cryptography Conference*, TCC 2006, pp. 601–616.

[25] Dorrendorf L., Gutterman Z., Pinkas B., *Cryptanalysis of the Windows Random Number Generator*, *ACM Conference on Computer and Communications Security 2007*, pp. 476–485.

[26] Durrant S., *Random Numbers in Data Security Systems*, Intel Coropration, 1999, `http://www.intel.com/design/security/rng/WP\_Durrant.htm`

[27] Eastlake D., Schiller J., Crocker S., *Randomness Requirements for Security*, RFC 4086, 2005.

[28] Ekdahl P., Johansson T., *A new version of the stream cipher SNOW*, *Selected Areas in Cryptography* (SAC 2002), Springer-Verlag, 2002, pp. 47–6.

[29] Elias P., *The efficient construction of an unbiased random sequence*, Ann. Math. Stat., vol. **43**, no. 2, 1972, pp. 865–870.

[30] Ellison C., *Cryptographic Random Numbers*, Draft P1363 Appendix E, 1998.

[31] eSTREAM, *ECRYPT Stream Cipher Project*, `http://www.ecrypt.eu.org/stream.`

[32] Feige U., Fiat A., Shamir A., *Zero-knowledge proofs of identity*, ACM Special Interest Group on Algorithms and Computation Theory (SIGACT), 1987.

[33] FIPS-198a, *The Keyed-Hash Message Authentication Code (HMAC)*, U.S. Department of Commerce/N.I.S.T., 2002.

[34] Fluhrer S., McGrew D. A., *Statistical Analysis of the Alleged RC4 Keystream Generator*, *Fast Software Encryption 2000, LNCS 1978*, Springer-Verlag, 2000, pp. 19–30.

[35] Fluhrer S., Lucks S., *Analysis of the E0 Encryption System*, *Selected Areas in Cryptography- SAC 2001*, LNCS 2259, 2001, Springer-Verlag, pp. 38–38.

[36] Franklin M., Yung M., *The Blinding of Weak Signatures*, LNCS 950, in *Proceedings of EUROCRYPT'94, Advances in Cryptology*, Springer-Verlag, 1995, pp. 67–76.

[37] Gammel B. M., Göttfert R., Kniffler O., *The Achterbahn stream cipher*, eSTREAM, the ECRYPT Stream Cipher Project, Report 2005/002, 2005, `http://www.ecrypt.eu.org/stream/`

[38] Geffe P. R., *How to protect data with ciphers that are really hard to break*, Electronics, 1973, pp. 99–101.

[39] Gennaro R., *Randomness in Cryptography*, IEEE Security and Privacy, vol. **4**, no. 2, 2006, pp. 64–67.

[40] Gligor V., Donescu P., *Fast Encryption and Authentication: XCBC Encryption and XECB Authentication Modes*, 2nd NIST Workshop on AES Modes of Operation, Santa Barbara, USA, 2001.

[41] Goldberg I., Wagner D., *Randomness and the Netscape browser*, Dr. Dobb's Journal, 1996, pp. 66–70.

[42] Gutmann P., *Software Generation of Practically Strong Numbers*, 1998 USENIX Security Symposium, 1998, revised June 2000.

[43] Gutterman Z., Malkhi D., *Hold your sessions: An attack on Java session-id generation*, LNCS, 3376, Springer-Verlag, 2005, pp. 44–57.

[44] Gutterman Z., Pinkas B., Reinman T., *Analysis of the Linux Random Number Generator*, Cryptology ePrint Archive, Report 2006/086.

[45] Hell M., Johansson T., Meier W., *Grain - A Stream Cipher for Constrained Environments*, International Journal of Wireless and Mobile Computing, Special Issue on Security of Computer Network and Mobile Systems., 2006.

[46] Housley R., *Cryptographic Message Syntax (CMS)*, RFC 3852, 2004.

[47] Jansen C., Kolosha A., *Cascade Jump Controlled Sequence Generator (CJCSG) (Pomaranch)*, eSTREAM, ECRYPT Stream Cipher Project, Report 2005/022, 2005.

[48] Jena D., Jena S. K., Majhi B., *A Novel Untraceable Blind Signature Based on Elliptic Curve*, International Journal of Computer Science and Network Security (IJCSNS) vol. **7**, no. 6, 2007.

[49] Kaufman C., *The Internet Key Exchange (IKEv2) Protocol*, RFC 2306, 2005.

[50] Black D., McGrew D., *Using Authenticated Encryption Algorithms with the Encrypted Payload of the Internet Key Exchange version 2 (IKEv2) Protocol*, RFC 5282, 2008.

[51] Kelsey J., Schneier B., Wagner D., Hall C., *Cryptanalytic Attacks on Pseudorandom Number Generators*, Fast Software Encryption, LNCS 1372, Springer-Verlag, 1998, pp. 168–188.

[52] Kelsey J., *Entropy and entropy sources in x9.82*, in *Proceeding of the NIST Random Number Generation Workshop*, July 2004.

[53] Kerckhoffs A., *La Cryptographie Militaire*, Journal des Sciences Militaires, vol. **9**, 1883, pp. 5–38.

[54] Knudsen L. R., *Practically secure Feistel ciphers*, Fast Software Encryption, LNCS 809, Springer-Verlag, 1994, pp. 211–221.

[55] Knuth D. E., *The Art of Computer Programming*, vol. **2**, 2nd edition, Addison-Wesley, 1981.

[56] Krawczyk H., Bellare M., Canetti R., *HMAC: Keyed-Hashing for Message Authentication*, RFC 2104, 1997.

[57] Lano J., *Cryptanalysis and Design of Synchronous Stream Ciphers*, PhD thesis, Katholieke Universiteit Leuven, B. Preneel, 2006.

[58] *The LavaRnd Random Number Generator*, http://www.lavarnd.org/

[59] L'Ecuyer P., Simard R., *TestU01: A C Library for Empirical Testing of Random Number Generators*, ACM Transactions on Mathematical Software, vol. **33**, no. 4, Article 22, 2007.

[60] McInnes J. L., Pinkas B., *On the impossibility of private key cryptography with weakly random keys*, in *Proceedings of CRYPTO90*, LNCS 537, Springer-Verlag, 1990, pp. 421–436.

[61] Marsaglia G., *The Diehard test suite*, `http://www.csis.hku.hk/diehard/`, 2003.

[62] Menezes A., Oorschot P. van, Vanstone S., *Handbook of Applied Cryptography*, CRC Press, 1996.

[63] Needham, R., Schroeder, M., *Using encryption for authentication in large networks of computers*, Communications of the ACM, vol. **21**, no. 12, 1978, pp. 993–999.

[64] Neumann J. von, *Various techniques used in connection with random digits*, National Bureau of Standards, Applied Mathematics Series, vol. **12**, 1951, pp. 36–38.

[65] NIST, FIPS 196, *Entity authentication using public key cryptography*, 1997.

[66] NIST, *The NIST test suite*, `http://csrc.nist.gov/groups/ST/toolkit/rng/batteriesstatstest.html`

[67] Dworkin M., *Recommendation for Block Cipher Modes of Operation - Methods and Techniques*, NIST Special Publication 800-38A, 2001.

[68] Dworkin M., *Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication*, NIST Special Publication 800-38B, 2005.

[69] Barker E., Barker W., Burr W., Polk W., Smid M., *Recommendation for Key Management - Part 1: General*, NIST Special Publication 800-57, revised March 2007.

[70] Barker E., Kelsey J., *Recommendation for Random Number Generation Using Deterministic Random Bit Generators*, NIST Special Publication 800-90, 2006, revised 2007.

[71] Chen L., *Recommendation for Key Derivation Using Pseudorandom Functions*, NIST Special Publication 800-108, 2008.

[72] Neuman C., Yu T., Hartman S., Raeburn K., *The Kerberos Network Authentication Service (V5)*, RFC 4120, 2005.

[73] Petit C., Standaert F., Pereira O., Malkin T., Yung M., *A block cipher based pseudo random number generator secure against side-channel key recovery*, in *ASIAN ACM Symposium on Information, Computer and Communications Security* (ASIACCS), pp. 56–65, 2008.

[74] RSA Laboratories. *PKCS #1 v2.1: RSA Encryption Standard*, 2002,

[75] idQuantique White Paper, *Random Numbers Generation using Quantum Physics*, http://www.idquantique.com/products/files/quantis-whitepaper.pdf

[76] RFC 2898, *PKCS #5: Password-Based Cryptography Specification Version 2.0*, `http://www.ietf.org/rfc/rfc2898.txt`

[77] Rivest R. L., *Are 'strong' primes needed for RSA?*, unpublished manuscript, 1991.

[78] Rose G. G., *A Stream Cipher Based on Linear Feedback over GF(28)*, ACISP 1998, LNCS 1438, pp. 135–146.

[79] Schneier B., *Applied Cryptography: Protocols, Algorithms, and Source Code in C*, Second Edition, John Wiley & Sons, 1996.

[80] Schneier B., *Schneier on Security: Random Number Bug in Debian Linux*, `http://www.schneier.com/blog/archives/2008/05/random\_number\_b.html`

[81] Schnorr C. P., *Efficient identification and signatures for smart cards*, Advances in Cryptology - Crypto '89, LNCS 435, Springer-Verlag, 1990, pp. 239–252.

[82] Seznec A., Sendrier N., *HArdware Volatile Entropy Gathering and Expansion: generating unpredictable random number at user level*, INRIA Research Report, 2002.

[83] Shoup V., *OAEP Reconsidered*, Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology, LNCS 2139, 2001, pp. 239–259.

[84] Shannon C. E., *A mathematical theory of communication*, Bell System Technical Journal, vol. **27**, pp. 379–423, 623–656, 1948.

[85] Simka M., Fischer V., Drutarovsky M., Fayolle J., *Model of a True Random Number Generator Aimed at Cryptographic Applications*, *Proceedings of the International Symposium on Circuits and Systems* (ISCAS) 2006, 2006, pp. 5619–5623.

[86] Simpson W., *Challenge Handshake Authentication Protocol (CHAP)*, RFC 1994, 1996.

[87] Soto J., *Randomness testing of the AES candidate algorithms*, NIST IR 6390, September 1999.

[88] Soto J., Bassham L., *Randomness Testing of the Advanced Encryption Standard Finalist Candidates*, in *Proceedings of the 3rd AES Candidate Conference*, 2000.

[89] Stinson D.R., *Cryptography – Theory and Practice*, CRC Press, Boca Raton, 1995.

[90] Stipcevic M., Medved Rogina B., *Quantum random number generator*, Rudjer Boskovic Institute, Bijenicka, Zagreb, Croatia, 2007.

[91] Suciu A., Marton K., Antal Z., *Data Flow Entropy Collector, Proceedings of the 10th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, GTARNS08 workshop, 2008, pp. 445–448.

[92] Yao F. F., Yin Y. L., *Design and Analysis of Password-Based Key Derivation Functions*, LNCS 3376, Springer-Verlag, 2005, pp. 245–26.

[93] Walker J., *Hotbits: Genuine random numbers, generated by radioactive decay*, Fourmilab, `http://www.fourmilab.ch/hotbits/how.htm`, 1996.

[94] Walker J., *ENT - A Pseudorandom Number Sequence Test Program*, Fourmilab, 2008, `http://www.fourmilab.ch/random/`

[95] Whiting D., Schneier B., Lucks S. Muller F., *Phelix - Fast Encryption and Authentication in a Single Cryptographic Primitive*, ECRYPT Stream Cipher Project Report 2005/027.

[96] Wu H., *The Misuse of RC4 in Microsoft Word and Excel*, Institute for Infocomm Research, Singapore, 2005.