Title:

Algorithms and Parameters for Secure Electronic Signatures

Source:

This document is the outcome of the work of the Algorithms group (ALGO) working under the umbrella of EESSI-SG (European Electronic Signature Standardisation Initiative Steering Group).

Background/Content/Remarks:

This document provides for security and interoperability for the application of the underlying mathematical algorithms and related parameters for secure electronic signatures in accordance with the "DIRECTIVE 1999/93/EC OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL of 13 December 1999 on a Community framework for electronic signatures."

This document will be handed to the EC via the EESSI-SG through ICTSB as a substantial contribution to be discussed further at the A9C (Article 9 Committee) level. The final decision on how to respect and how to handle this contribution and its future handling process will be the responsibility of the EC and A9C.

The document defines a list of approved cryptographic algorithms together with the requirements on their parameters, as well as the approved combinations of algorithms in the form of "signature suites". The approved algorithms and parameters shall be referenced in the corresponding Protection Profiles (e.g. for SSCDs or trusted CSP components).

The document contains several informative annexes which provide useful information on a number of subjects mentioned in the text.

ALGORITHMS AND PARAMETERS FOR SECURE ELECTRONIC SIGNATURES

Contents

Foreword

This clause is always the first unnumbered clause.
Normally it is drafted by the CEN/ISSS secretariat, but in some cases, the foreword can also be the subject of some discussion. In any case can the Foreword contain specific elements to the Workshop.
Typically, the Foreword will refer to the other parts in case of a multi-part CWA.

1 Scope

The present document defines an initial set of algorithms and the corresponding parameters to be included in a list of approved methods for producing or verifying Electronic Signatures in Secure Signature-Creating Devices (SSCD) (EESSI-work area F: Security Requirements for Secure Signature Creation Devices), to be referenced in the Certificate Policy documents (EESSI-work area A: ETSI TS 101 465: Policy requirements for certification authorities issuing qualified certificates), during the signature creation and validation process and environment (EESSI-work area G1/2), in trusted CSP components (Certification Service Provider) (EESSI-work-area D: Security Requirements for Trustworthy Systems managing Certificates for Electronic Signatures) and other technical components and related areas.

The document defines a list of approved cryptographic algorithms combined with the requirements on their parameters, as well as the approved combinations of algorithms in the form of “signature suites”. The approved algorithms and parameters shall be referenced in the corresponding Protection Profiles (e.g. for SSCDs or trusted CSP components). To support the management activities, a numbering scheme for cryptographic algorithms and their parameters is defined.

Specifically, the document gives guidance on

- management practices for cryptographic algorithms (see 4.1),
- signature suites (see 4.2),
- cryptographic hash functions (4.3),
- padding methods (see 4.4),
- signature algorithms and the corresponding key generation algorithms (see 4.5), and
- random-number generation (see 4.6).

The document also gives guidance on the management practices that shall be applied to cope with developments such as the examples given in the Annex A. It describes a process for updating the approved algorithms lists and parameters. Annex B contains OIDs assigned to the approved algorithms, Annex C gives more information on the generation of RSA keys for signatures and Annex D addresses the generation of random data.

Patent related issues are out of the scope of this document.

2 Normative References

[1] International Organization for Standardization, “ISO/IEC 9979: Information technology - Security techniques - Procedures for the registration of cryptographic algorithms,” 1999.
[2] Housley,R., et al., “Internet X.509 Public Key Infrastructure. Certificate and CRL Profile,” The Internet Engineering Task Force, RFC 2459, January 1999.
[3] International Organization for Standardization, , “ISO/IEC 10118-3: Information technology - Security techniques - Hash functions - Part 3: Dedicated hash functions,” 1998.
[4] National Institute of Standards and Technology, “NIST: FIPS Publication 180-1: Secure Hash Standard (SHS-1),” May 1995.
[5] Dobbertin, H., A. Bosselaers, and B. Preneel, “RIPEMD-160: A strengthened version of RIPEMD,” In *Fast software encryption, Proc. third International Workshop, Cambridge, UK, February 21 - 23, 1996*, pp. 71-82, D. Gollmann (ed.), LNCS 1039, Springer-Verlag, 1996.
[6] RSA Laboratories, “PKCS #1 v2.0: RSA Cryptography Standard,” October 1998.
[7] ISO, “ISO/IEC 14888-3: Information technology - Security techniques - Digital signatures with appendix - Part 3: Certificate-based mechanisms,” 1999.
[8] Rivest, R.L., “Finding four million random primes,” In *Advances in Cryptology - Crypto '90*, pp. 625-626, A.J. Menezes (ed.), LNCS 537, Springer-Verlag, 1991.
[9] National Institute of Standards and Technology, “NIST: FIPS Publication 140-1: Security requirements for cryptographic modules,” January 1994.
[10] Blum. M., and S. Micali, “How to generate cryptographically strong sequences of pseudo-random bits,” *SIAM Journal on Computing*, Vol. 4, No. 13, 1984, pp. 850-863.
[11] NIST, “NIST: FIPS Publication 186-2: Digital Signature Standard (DSS),” January 2000.
[12] The Institute of Electrical and Electronics Engineers, Inc, “Standard Specifications for Public-Key Cryptography,” IEEE P1363, 2000.
[13] American National Standards Institute, “Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA),” ANSI X9.62-1998, 1998.
[14] International Organization for Standardization, “ISO/IEC 14888-3 Information technology - Security techniques - Digital signatures with appendix - Part 3: Certificate-based mechanisms,” Draft, 2000.
[15] International Organization for Standardization, “ISO/IEC 9796-3: Information technology - Security techniques - Digital signature schemes giving message recovery -- Part 3: Discrete logarithm based mechanisms, ” Draft, 2000.
[16] International Organization for Standardization, “ISO/IEC 15946-2: Information technology - Security techniques - Cryptographic techniques based on elliptic curves - Part 2: Digital signatures,” Draft, 2000.
[17] Rivest, R., A. Shamir, and L. Adleman, “A method for obtaining digital signatures and public key cryptosystems,” *Communications of the ACM*, Vol. 21, No. 2, 1978, pp. 120-126.
[18] International Organization for Standardization, “ISO/IEC 15946-2: Information technology – Security techniques – Cryptographic techniques based on elliptic curves – Part 4: Digital signatures giving message recovery,” Committee Draft 2001-03-08.
[19] RFC 1750: “Randomness Recommendations for Security,” Internet Request for Comment 1750, 1994.
[20] American National Standards Institute, “Financial Institution Key Management (wholesale),” ANSIX9.17-1985, 1985.
[21] AIS 20: “Application Notes and Interpretation of the Scheme : Functionality classes and evaluation methodology for deterministic random number generators.” Available at http://www.bsi.bund.de/aufgaben/ii/zert/jil_ais/ais20e.pdf.
[22] RSA Laboratories, “PKCS #1 v2.1 draft 2: RSA Cryptography Standard,” January 2001.

3 Definitions and Abbreviations

3.1 Definitions

Management activity (MA) – An MA is an action that shall be taken by the electronic signature committee under the circumstances specified in Clause 4 of this document.

Signature suite – A signature suite is a combination of a signature algorithm with its parameters, a key generation algorithm, a padding method, and a cryptographic hash function. The currently approved signature suites are specified in this document.

Bit length – The bit length of an integer p is r if $2^{r-1} = p < 2^r$.

3.2 Abbreviations

A9C	Article 9 Committee
DSA	Digital Signature Algorithm
ECDSA	Elliptic Curve Digital Signature Algorithm
MA	Management Activity
OID	Object Identifier
RSA	Rivest, Shamir, and Adleman
SCD	Signature Creation Data
SSCD	Secure Signature-Creation Device
SVD	Signature Verification Data

4 Algorithms and Parameters for Secure Electronic Signatures

4.1 Management activities

As a response to relevant developments in the area of cryptography and technology, activities for the management of the algorithms and parameters for secure electronic signatures shall enable dynamic updating of the lists of approved algorithms and parameters. The initial lists of approved algorithms and their parameters are given in this document.

The management activities to introduce new algorithms and their parameters and to delete algorithms and their parameters from the list need to respond to the following situations:

1. The need to introduce new algorithms and relevant parameters will call for a mechanism that is rather dynamic. Normal update cycles of this document are not deemed to be fast enough to reflect the market need.

2. Advances in cryptography will call for a phasing out of some algorithms or parameters. Such phasing out will be known well in advance.

3. In the case of new attacks the immediate need to remove an algorithm can arise.

Through A9C, mechanisms shall be put in place that can react within 6 months for cases 1 and 2 and at most 1 month for case 3.

Further information on updating algorithms and parameters is contained in Annex A.

4.2 Signature suites for secure electronic signatures

Due to possible interactions which may influence security of electronic signatures, algorithms and parameters for secure electronic signatures shall be used only in predefined combinations referred to as the signature suites. A signature suite consists of the following components:

- a signature algorithm with parameters,
- a key generation algorithm,
- a padding method, and
- a cryptographic hash function.

If any of the components of a suite has been cancelled, the suite must be cancelled as well. If any of the components of a suite has been updated, the suite must be updated as well.

The list of currently approved signature suites is given in Table 1.

Each entry in the list of signature suites shall contain two expiry dates. The first is the last day the suite can be used for producing signatures. The second is the last day that the suite should be used to verify such signatures, which will typically be one year later, to give some flexibility. A different expiry date for signature verification may be enforced by the signature policy. These dates require revision by the first date for the mentioned suite.

For all components only short names are used. Each component is defined in a separate table in the following sub-clauses. Object Identifiers (OIDs) are specified in Annex B.

Table 1 – The list of approved signature suites

Signature suite entry index	Signature algorithm	Signature algorithm parameters	Key generation algorithm	Padding method	Cryptographic hash function	Expiry date (signing)	Expiry date (verification)
001	rsa	MinModLen=1020	rsagen1	emsa-pkcs-v1_5	sha1	31.12.2005	31.12.2006
002	rsa	MinModLen=1020	rsagen1	emsa-pss [1]	sha1	31.12.2005	31.12.2006
003	rsa	MinModLen=1020	rsagen1	emsa-pkcs-v1_5	ripemd160	31.12.2005	31.12.2006
004	rsa	MinModLen=1020	rsagen1	emsa-pss [1]	ripemd160	31.12.2005	31.12.2006
005	dsa	pMinLen=1024 qMinLen=160	dsagen1	-	sha1	31.12.2005	31.12.2006
006	ecdsa-Fp	qMinLen=160 r0Min=10^4 MinClass=200	ecgen1	-	sha1	31.12.2005	31.12.2006

[1] When finalized.

007	ecdsa-F2m	qMinLen=160 r0Min=10^4 MinClass=200	ecgen1	-	sha1	31.12.2005	31.12.2006

4.3 Cryptographic hash functions

A cryptographic hash function is a preimage resistant and 2nd preimage resistant function with a constant-length output used to compute the hashsum of a document to be signed. A hash function must be collision-resistant which means that it is computationally infeasible to find two different documents yielding the same hashsum (which implies that it is also infeasible to find a different document yielding the same hashsum as a given document).

If due to advances in cryptography or computing technology any of these conditions is no longer met, the hash function is considered insecure and must be removed from the list of approved hash functions (see section 4.1, Management activities).

The list of currently approved hash functions is given in Table 2. Each hash function has a unique entry index represented by a string beginning with "2." followed by a two-digit entry number. The relevant OIDs are given in Annex B.

Table 2 – The list of approved cryptographic hash functions

Hash function entry index	Short hash function entry name	Adoption date	Normative references
2.01	sha1	01.01.2001	[3,4]
2.02	ripemd160	01.01.2001	[3,5]

4.4 Padding methods

Some signature algorithms require a hashsum to be padded to up to a certain block length used with a specific algorithm setting (e.g. RSA modulus length). This document does not specify mandatory padding methods, but requires a padding method, if required by an algorithm (as indicated in 4.5), to meet certain requirements defined in the given normative references.

The list of currently approved padding methods is given in Table 3. Each padding method has a unique entry index represented by a string beginning with "3." followed by a two-digit entry number.

Table 3 – The list of approved padding methods

Padding method entry index	Short padding function entry name	Random number generation method	Random generator parameters	Adoption date	Normative references
3.01	emsa-pkcs-v1_5	-	-	01.01.2001	[6], Section 9.2.1
3.02	emsa-pss	TBD	TBD	TBD	[22], Section 9.2.2

It is intended that further padding schemes will be added as and when they have been fully evaluated. These include several methods based on ISO/IEC 9796-2.

4.5 Signature algorithms

4.5.1 General comments

A signature algorithm is applied to the hashsum of the document to be signed to generate a signature with the SCD. The algorithm to be applied for verification with the SVD must be given. Before a signature algorithm is applicable a complete set of approved parameters must be defined. It must be practically impossible to compute the SCD from the SVD.

The list of currently approved signature algorithms is given in Table 4. Each signature algorithm has a unique entry index represented by a string beginning with "1." followed by a two-digit entry number.

Note that the minimum modulus length for RSA is given as 1020 bits rather than the more natural 1024. This is to allow for implementations that cannot use the topmost bit(s).

Table 4 – The list of approved signature algorithms

Signature algorithm entry index	Short signature algorithm entry name	Signature algorithm parameters	Padding methods	Key generation algorithms	Adoption date	Normative references
1.01	rsa	MinModLen=1020	emsa-pss	rsagen1	TBD	[17,7]
1.02	dsa	pMinLen=1024 qMinLen=160	-	dsagen1	01.01.2001	[11]
1.03	ecdsa-Fp	qMinLen=160 r0Min=10^4 MinClass=200	-	ecgen1	01.01.2001	[11,13]
1.04	ecdsa-F2m	qMinLen=160 r0Min=10^4 MinClass=200	-	ecgen1	01.01.2001	[11,13]

The following subclauses describe the parameters and key generation algorithms for the signature algorithms listed in Table 4. Table 5 summarises the approved key generation algorithms for all signature algorithms considered in this document.

Table 5 – The list of approved key generation algorithms

Key generator entry index	Short key generator entry name	Signature algorithm	Random number generation method	Random generator parameters	Adoption date	Normative references
4.01	rsagen1	rsa	trueran	EntropyBits≥128	01.01.2001	
4.02	dsagen1	dsa	trueran or pseuran (FIPS 186-2 method)	EntropyBits≥128 or SeedLen≥128	01.01.2001	[11]
4.03	ecgen1	ecdsa-Fp or ecdsaF2m	trueran or pseuran	EntropyBits≥128 or SeedLen≥128	01.01.2001	

4.5.2 RSA

4.5.2.1 Parameters

The RSA algorithm's security is based on the difficulty of factoring large integers. To generate SCD and SVD, two prime numbers, p and q, are generated randomly and independently, satisfying the following properties:

- the bit length of the modulus $n = pq$ must be at least MinModLen; its length is also referred to as ModLen;
- p and q should have roughly the same length; e.g. set a range such as $0.5 < | \log_2 p - \log_2 q | < 30$;
- there should be sufficiently many primes to choose from and they should be reasonably uniformly distributed.

The SCD consists of the private exponent d and the modulus n.
The SVD consists of the public exponent e and the modulus n.

4.5.2.2 Key generation algorithm rsagen1

Generate p and q as indicated above by applying a random number generation method satisfying the requirements trueran (see 4.6). The modulus must effectively be influenced by EntropyBits bits of entropy. Random numbers shall be tested for primality until one of them is found to be prime with a probability of error (i.e. of actually being composite) of at most 2^{-60}. For some empirical results on the reliability of simple primality tests, see, for example, [8]. For more detailed information on the bounds of the Miller-Rabin test see [DLP].

Select the public exponent e as an integer between 3 and n-1 so that gcd(e, lcm(p-1, q-1))=1 holds. Compute the private exponent d such that $ed \equiv 1 \pmod{\text{lcm}(p-1, q-1)}$. Note that any solution d to the equation $ed \equiv 1 \pmod{(p-1)(q-1)}$ will automatically satisfy this.

4.5.3 DSA

4.5.3.1 Parameters

The DSA algorithm's security is based on the difficulty of computing the discrete logarithm in the multiplicative group of a prime field F_p. The DSA computations shall be performed as described in [11]. The public parameters p, q and g may be common to a group of users. The prime modulus p shall be at least pMinLen bits long. q, which is a prime divisor of (p-1), shall be at least qMinLen bits long. g shall be computed as indicated in [11].

The SCD consists of

- a randomly or pseudorandomly generated integer x, $0 < x < q$, which is signatory-specific, and
- a randomly or pseudorandomly generated integer k, $0 < k < q$, which must be regenerated for each signature.

If the distribution of k is significantly different from uniform within the interval then there may be weaknesses.

The SVD consists of an integer y computed as $y = g^x \bmod p$.

When computing a signature of a message M, no padding of the hashsum is necessary. However, the hashsum must be converted to an integer by applying the method described in Appendix 2.2 of [11].

4.5.3.2 Key generation algorithm dsagen1

p and q shall be generated as described in Appendix 2.2 of [11].

Generate x by applying a random number generation method satisfying the requirements trueran (see 4.6.2) or using a method satisfying pseuran (see 4.6.3) with an appropriate size seed. The resulting bits shall effectively be influenced by EntropyBits bits of true randomness. Generate k using one of these methods; k does not have to be generated using exactly the same method as x. The pseudo-random number methods of FIPS 186-2 [11] are suitable for pseuran.

4.5.4 Elliptic curve analogue of DSA based on a group $E(F_p)$

4.5.4.1 Parameters

This signature algorithm is referred to as ecdsa-Fp. The algorithm shall be applied as specified in the normative references [11, 13] given in Table 4. The same algorithm is also specified in references [12, 14, 16] which can be used for information. The security of the ecdsa-Fp algorithm is based on the difficulty of computing the elliptic curve discrete logarithm.

The public parameters are as follows:

- p large prime,
- q large prime at least qMinLen bits long, $p \neq q$,
- E elliptic curve over a finite field F_p whose order is divisible by q, and
- P fixed point on E of order q.

The class number of the maximal order of the endomorphism ring of E shall be at least MinClass. The value r_0:=min(r: q divides p^r-1) shall be greater than r0Min.

In FIPS 186-2 [11] five curves over a prime field are defined. All these curves fulfil the above requirements.

The SCD consists of

- a statistically unique and unpredictable integer x, $0 < x < q$, which is signatory-specific, and
- a statistically unique and unpredictable integer k, $0 < k < q$, which must be regenerated for each signature.

The SVD consists of Q, a point of E which is computed as $Q=xP$.

4.5.4.2 Key generation algorithm ecgen1 for ecdsa-Fp

The prime number p. which determines the field size, shall be generated as specified in [13]. The elliptic curve over F_p shall be selected to have an order divisible by a prime q of length ≥ qMinLen ≥ 160 as specified in [13]. x and k shall be generated by a random number generator satisfying the requirements trueran or pseuran with the additional requirement that k practically never repeats. k does not have to be generated using the same method as x.

4.5.5 Elliptic curve analogue of DSA based on a group $E(F_{2^m})$

4.5.5.1 Parameters

This signature algorithm is referred to as ecdsa-F2m. The algorithm shall be applied as specified in the normative references [11, 13] given in Table 4. The same algorithm is also specified in references [12, 14, 16] which can be used for information. The security of the ecdsa-F2m algorithm is based on the difficulty of computing the elliptic curve discrete logarithm.

The public parameters are as follows:

- m prime number,

- q large prime at least qMinLen bits long,
- E elliptic curve over a finite field F_{2^m} whose order is divisible by q;
- it must not be possible to define E over F_2, and
- P fixed point on E of order q.

The class number of the maximal order of the endomorphism ring of E shall be at least MinClass. The value r_0:=min(r: q divides 2^{mr}-1) shall be greater than r0Min.

In FIPS 186-2 [11] five pseudorandomly generated curves over F_{2^m} are defined. All these curves satisfy the above requirements. Note that the Koblitz curves given in [11] are defined over F_2 and hence do not fulfil the fourth requirement.

The SCD consists of

- a statistically unique and unpredictable integer x, $0 < x < q$, which is signatory-specific, and
- a statistically unique and unpredictable integer k, $0 < k < q$, which must be regenerated for each signature.

The SVD consists of Q, a point of E which is computed as $Q=xP$.

4.5.5.2 Key generation algorithm ecgen1 for ecdsa-F2m

The prime number m, which determines the field size, shall be fixed as specified in [13]. The elliptic curve over F_{2^m} shall be selected to have an order divisible by a prime q of length ≥ qMinLen ≥ 160 as specified in [13]. x and k shall be generated by a random number generator satisfying the requirements trueran or pseuran with the additional requirement that k practically never repeats. k does not have to be generated using the same method as x.

4.5.6 Discrete logarithm based signatures giving message recovery

Discrete logarithm based signatures giving message recovery are implemented over the same algebraic structures as dsa and ecdsa algorithms and they make use of the same procedures for generating parameters and keys. Discrete logarithm based signatures giving message recovery can be used to produce very compact signatures in case only short messages are to be signed. Such algorithms are specified in [12], [15] and [18].

4.6 Random number generation

4.6.1 General comments

Random number generation is relevant to generating the SCD or to generating random parameters in some cryptographic algorithms (e.g. DSA). In some cases it may also be relevant to hash function padding. Therefore the possible choices of random number generation requirements are always given in connection with padding methods (Table 3) and key generation algorithms (Table 5).

Table 6 – The list of approved random number generation methods

Random generator entry index	Short random generator entry name	Random generator parameters	Adoption date	Normative references
5.01	trueran	EntropyBits	01.01.2001	
5.02	pseuran	SeedLen	01.01.2001	
5.03	fips186-2-31	SeedLen	01.01.2001	[11]
5.04	fips186-2-32	SeedLen	01.01.2001	[11]

4.6.2 Random generator requirements trueran

A physical random generator based on a physical noise source (primary noise) and a cryptographic or mathematical post-treatment of the primary noise. The primary noise must be subjected to an adapted statistical test on a regular basis (see, e.g. [9], Section 4.11.1, Statistical random number generator tests). See Annex D for more detailed information on generating random numbers. The expected effort of guessing a cryptographic key shall be at least equivalent to guessing a random value which is EntropyBits long.

4.6.3 Random generator requirements pseuran

A pseudo-random generator must be initialised by a genuine random number, called seed, of length SeedLen. The output of the generator must satisfy the following requirements:

- no information is ascertainable a priori as to the output bits which are generated;

- the knowledge of a partial output sequence permits no inferences with regard to the remaining bits with probability higher than 0.5;
- there is no usable method of recovering the seed from a subset of the output of the generator.

A pseudo-random generator satisfying these requirements is described in, for example, [10].

4.6.4 Random number generator fips186-2-31

This random number generator is specified in Appendix 3.1 of [11].

4.6.5 Random number generator fips186-2-32

This random number generator is specified in Appendix 3.2 of [11].

Annex A (normative): Updating algorithms and parameters

A.1 Introduction

Cryptographic algorithms in general do not offer unlimited perfect security in the information-theoretical sense. Their security depends on

- the difficulty of solving a hard mathematical problem that they are based on, and
- the computational infeasibility of solving the problem using the current technology.

This effectively means that a previously secure cryptographic algorithm cannot be considered secure any longer if, for example,

- a mathematical method has been found so that the previously hard problem the algorithm was based on is no longer hard, or
- a new type of attack, which explores an algorithm implementation vulnerability, is developed, or
- the advances in technology make it possible to solve the problem within a significantly shorter period of time.

In any of those cases a specific implementation of a cryptographic algorithm cannot be considered secure any longer. It is therefore of crucial importance to establish suitable management practices to cope with such developments to prevent from use of insecure algorithms.

This Annex defines the management practices to enable fast and technologically appropriate reactions to new developments in computing technology and new findings in the area of cryptography.

A.2 Management Process

As a response to relevant developments in the area of cryptography and technology, it is recommended that the lists of approved algorithms and parameter values be dynamically updated. The initial lists of approved algorithms and their parameter values are given in this document. This annex identifies several cases where an update of the lists is required:

Adopting a new algorithm. As a result of monitoring the relevant developments in cryptography and computing technology, a notified body of a member state may require adoption of a new algorithm. An algorithm can be adopted if at least one complete set of parameters for this algorithm has been adopted. Definitions of the complete sets of parameters for a specific algorithm are given in this document.

Cancelling an algorithm. As a result of monitoring the relevant developments in cryptography and computing technology, a notified body of a member state may require cancellation of a currently approved algorithm. The requirement should contain a clear reasoning about the insufficient security of the algorithm to be cancelled and about the implications for existing products using this algorithm.

Updating parameter values for an approved algorithm. As a result of monitoring the relevant developments in cryptography and computing technology, a notified body of a member state may require an update to the parameter values of an approved algorithm. The requirement should contain clear reasoning about the insufficient security of the parameter values to be updated and about the implications for existing products using these parameter values.

If any of the components of a **signature suite** has been cancelled/updated, the suite must be cancelled/updated as well.

Monitoring the relevant developments in cryptography and computing technology should be a continuous activity performed by the notified bodies of the member states. The activity should be focused on answering the following questions:

- Is a currently approved algorithm still considered secure?
- Are the currently approved parameter values for an algorithm still considered secure?
- Are there new algorithms that should be considered? Is there a market demand for such algorithms?

The cancellation/update date should be set in such a way as to allow some **transition period** within which the algorithm/parameter values to be cancelled/updated may be used in the existing products (e.g. signature devices, digital certificates). It should, however, be indicated that the cancelled/updated items should not be considered for use in new products (e.g. signature devices under development, certificates to be issued). The transition period should allow the vendors using the cancelled/updated items time to alter their production process. If the security implications of a cancellation/update are considered very serious, it should be recommended to withdraw the products using the cancelled/updated item before their planned expiry date.

For example, if a signature key length is no longer considered secure under a low or medium attack potential, digital certificates containing signature keys of this length should be revoked as soon as possible. In addition, a transition period should be recommended within which the existing signatures generated with keys of insufficient length can be verified as being valid. After the transition period the corresponding documents should be re-signed with a longer key. When issuing a certificate, a CSP should be careful not to define the validity period of the certificate to be much longer than the presumed validity period of the applied cryptographic hash functions, signature algorithms, and their parameter values. It is recommended that the length of the CSP's signature keys (for issuing certificates and CRLs) is always chosen to be significantly longer than the currently recommended key length for the signature algorithm used.

Annex B (informative): Algorithm Object Identifiers

The cryptographic algorithm objects listed in the main body of this document are specified using their short name and the normative reference where the exact specification of the algorithm is to be found.

In most cases the cryptographic objects have been assigned a full Object Identifier (OID) by their owner organisations to ensure interoperability of different implementations. In [1] the tree of algorithm object identifiers is depicted, and a list is given of cryptographic algorithms that can be identified using OID.

Table 7 relates the short name of the algorithms, as they appear in this document, to their respective object identifiers.

Table 7 – Object Identifiers

Short name	OID
rsa	{ joint-iso-ccitt(2) ds(5) module(1) algorithm(8) encryptionAlgorithm(1) 1 }
rsaEncryption	{ iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-1(1) 1 }
sha-1WithRSAEncryption	{ iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-1(1) 5 }
id-dsa	{ iso(1) member-body(2) us(840) x9-57(10040) x9cm(4) 1 }
id-dsa-with-sha1	{ iso(1) member-body(2) us(840) x9-57 (10040) x9cm(4) 3 }
sha1	{ iso(1) identifiedOrganization(3) oIW(14) oIWSecSig(3) oIWSecAlgorithm(2) 26 }
ripemd160	{ iso(1) identifiedOrganization(3) teletrust(36) algorithm(3) hashAlgorithm(2) 1 }

The object identifiers are for information only. The user of this document is encouraged to consult the owner organisation for updates or modifications.

See [SECOID] for further details.

Annex C (informative): Generation of RSA keys for signatures

C.1 Generation of random prime numbers

C.1.1 Probabilistic primality test

The generation of large prime numbers for cryptographic applications is usually done using probabilistic primality tests. These algorithms are very efficient but may declare that a composite number is prime with some (small) probability. The Miller-Rabin test is the best such algorithm. It is known that the error probability is less than $(¼)^k$ after k iterations of the elementary algorithm. Consequently, the probability for a random number to be wrongly declared prime after 40 iterations is less than $½^{80}$. More precise analysis of the Miller-Rabin algorithm enables a reduction in the number of iterations required; see for example [xxx].

The generation of random prime numbers in the range $[a,b]$ is done using the following algorithm:

- randomly choose an odd integer x in the range $[a,b]$,
- try to divide x by all the prime numbers smaller than a (small) bound B; if x is divisible, go back to the first step.
- Using the Miller-Rabin probabilistic primality algorithm, test if x is probably prime; if this is not the case, go back to the first step.

Note that the second step just makes the generation faster since the small prime numbers are the most probable prime factors. Furthermore, this test can be done in a single operation by just testing if x is relatively prime with the precomputed product of all the prime numbers smaller than B.

The probability for an odd integer randomly chosen in the range $[a,b]$ to be prime is about $2/\ln(b)$ so the number of repetitions of the algorithm will be less than $\ln(b)/2$. Furthermore, the entropy of k-bit primes generated with this method is about k-$\ln(k)$.

From a practical point of view, the time needed to generate a prime number can be slightly reduced if, instead of choosing a new random integer x for each test, we look for the smallest prime number larger than a random integer x. From a theoretical point of view, even if it does not enable any known attack, the problem is that the probability of finding a given prime number is no longer uniform.

C.1.2 Safe prime numbers

For some cryptographic applications, it is sometimes advised to use so-called *safe* prime numbers. A (probable) prime p is said to be safe if

- p-1 has a large prime factor r,
- p+1 has a large prime factor,
- r-1 has a large prime factor.

Those additional requirements are made to avoid certain factoring algorithms (Pollard p-1 algorithm and its generalizations). If such verifications can be easily done and if the efficiency of prime number generation is not a critical issue, they can be added to the prime number generation procedure. However, it has been proved that the probability for a randomly chosen prime number to fulfill those requirements is overwhelming for the current parameter sizes. Furthermore, the Pollard method is generalized by the elliptic curve factoring method so it is not possible to make an exhaustive list of weak forms of prime numbers.

In conclusion, the prime numbers much be randomly chosen and must not have any kind of special form; if this is done, additional tests can be added.

C.2 Generation of RSA modulus

An RSA modulus is obtained by multiplying two prime numbers of roughly the same size. Furthermore, the two factors must not be too close in order to be far enough from the square root of the modulus.

If we let p and q be the two prime factors of the modulus n, we can require that

$$0.5 < |\log_2(p)-\log_2(q)| < 30$$

This implies that

$$\log_2(n)/2\text{-}15 < \log_2(p), \log_2(q) < \log_2(n)/2+15$$

The generation of an RSA modulus of exactly k bits could be done with the following algorithm:

- Choose a random prime number p in the range $[2^{k/2-15}, 2^{k/2+15}]$;
- Choose a random prime number q in the range $[2^{k-1}/p, 2^k/p]$;
- If the condition $0.5 < |\log_2(p)-\log_2(q)| < 30$ is not satisfied, go back to the first step;
- Let n be the product of p and q.

A more complicated method that avoids the third step altogether but produces differently distributed primes is:

- Choose a random prime number p in the range $[2^{k/2-1/4}, 2^{k/2+15}]$;
- Choose a random prime number q in the range $[a,b]$ where $a=\max(2^{k-1}/p, p.2^{-30})$ and $b=\min(2^k/p, p.2^{-1/2})$;
- Let n be the product of p and q.

C.3 Generation of RSA keys

An RSA public key is made of an RSA modulus $n=p.q$ generated as explained in the previous section and of a public exponent e. The only requirement for e is to be relatively prime to $\text{lcm}(p\text{-}1,q\text{-}1)$. This public exponent may be chosen as small as $e=3$.

The related RSA secret key d (or signature generation key) is computed using the extended Euclidian algorithm:

$$e.d \equiv 1 \bmod \text{lcm}(p\text{-}1,q\text{-}1)$$

The optimization that consists of first choosing the secret exponent d and then computing the public exponent e can be used but in this case d must be randomly chosen in the set $[3,n]$; the use of small decryption exponents must be avoided since there exist some attacks that can factor the modulus in this case.

Let us also remember that a new modulus n must be used for each user of the signature scheme. A modulus must not be shared by some users, even if different public exponents are used. Furthermore, notice that if the RSA keys are generated as explained above, the probability of generating two keys with the same modulus or the same secret exponent is totally negligible, even if many keys are computed.

Annex D (informative): On the generation of random data

D.1 Why cryptography needs random numbers

Random data is commonly used in many cryptographic systems. However, different kinds of randomness have to be considered according to the application.

Randomness to avoid replay and collisions. Firstly, in some cases we just need to generate numbers that are distinct. For example, when some challenges are used for authentication, we only require that the same challenge is not generated twice. A solution is to store already used data or to use a counter but a much simpler solution is to generate enough random data. Consequently, as one of the applications of the birthday paradox, we know that, if the challenges are chosen in a large enough set, they will all be different with overwhelming probability. In this case, we see that the statistical properties of "random" data is unimportant; we can tolerate some bias without compromising the security.

Randomness to generate cryptographic data.
Secret keys. A stronger notion of random data must be used for applications such as the generation of secret keys for asymmetric algorithms such as signature. In this case, we need data with strong statistical properties in order to avoid attacks that may use known bias in secret key generation to increase their efficiency. A consequence is that the probability of generating the same secret key twice is negligible.

Public parameters. In addition to secret keys, public parameters, such as the prime numbers in DSA [11], are also generated using a non-deterministic procedure. Furthermore, in this case the randomness of those parameters may by proved to users, just by giving the initial random data and the algorithm used to derive public parameters from the data. Consequently, people are quite sure that the system wide parameters have not been chosen in a very specific way, with some properties that may lower the security.

Temporary secret data. Random bits are also necessary in many protocols for temporary use. For example, the DSA signature scheme (see FIPS 186-2 [11]) uses the secret key of the user who signs the message and also a one-time secret key that has to be regenerated for each signature (the signature of two messages with the same temporary key reveals both the key and the user's signature secret key!). The security analysis of DSA shows that both secrets have the same importance so it does not have any sense to carefully generate users secret keys if temporary keys are not good enough.

Stream ciphers. Finally, in applications such as the generation of secret keys for symmetric algorithms, the random data must be random in a very strong sense; for example, it should not be possible to derive any knowledge of generated data from previously generated data, even if the previously generated data is known. This situation may also occur in the context of signatures, for example if an authority generates secret keys and an attacker tries to gain information on some of those keys after having obtained some others. Consequently, there must not be any usable link between different keys.

Truly random bits vs. pseudorandom bits. The generation of random bits appears to be a difficult task. Consequently, pseudorandom bit generators often have to be used in many applications. They are not generators of truly random bits since they are deterministic but, starting with a random seed, they are able to generate sequences of bits that are random looking. In the next section we describe some ways of generating truly random bits. Then we will focus of pseudorandom bit generation.

D.2 Generation of truly random bits

Let us first understand what we mean by a true random number generator (TRNG); the aim of a TRNG is to generate individual bits, with uniform probability and without any correlation between those bits. Consequently, the knowledge of some bits does not give any information (in a strong information theoretic sense) about the other generated bits.

Notice that for cryptographic applications, public sources of (almost) random bits such as the digits of p cannot be used even if they are random-looking since they are not secret at all!

In practice, perfect random sources are not available, so the idea is to use a source of bits that may not be perfectly random and then to hash the bits in order to obtain really random bits. In this process, we assume that a hash function, such as SHA-1, is able to "extract" the randomness from a biased bit string. Formally, the amount of

randomness of such a string is measured by its entropy. The notion of entropy has to be used very carefully since it applies to probability distributions and not to actual bit strings. From a more practical point of view, it is useful to consider that a random source generates sequences of bits and that only a ratio is random. For example, on evaluation we may find that for eight generated bits we have one bit of randomness; consequently, hashing 1280 generated bits with SHA-1 will hopefully produce 160 truly random bits.

Randomness sources. The available random sources can vary a lot from an application to another. A computer specialised in the generation of secret keys may, for example, use specific devices based on quantum effects to generate high quality random bits (see IEEE P1363 Annex D [12] for a list of such sources).

More generally, a normal computer can measure the variations in mouse movements or keystrokes. It can also measure the timing of some tasks that are not achieved in constant time such as hard disk access. The randomness of such sources and an estimation of their entropy may be tested using statistical tests (see section D.3). In addition the source should be described by a stochastic model. This model should produce similar biases and dependencies to the source taking into account the physical effect that the source is based on. The use of a stochastic model allows the notion of entropy. With the help of the model a sufficient loss rate can be approximated.

Ideally, random data should come from a few different sources and hashed with a high loss rate (for example hash 1 Kbytes to obtain 160 bits of random bits); see [19] for further details.

Be aware that the functions that generate random numbers that are implemented in programming libraries do not always output cryptographically secure bits! It is, for example, well known that in the old versions of the C *rand* function, the least significant bits were not very random at all. Furthermore, remember that good statistical properties of a sequence, for example the digits of p or linear congruantial generators, are not sufficient to make them useful and secure for cryptographical applications.

Random number generation is very difficult in simple environments such as smart cards. Random sources are not directly available since there is no physical device and everything is very deterministic. Usually simple mechanisms such as pairs of asynchronous clocks are implemented but the quality of such randomness generators has to be analysed very carefully using mathematical modelling and statistical tests.

Algorithmic countermeasures to improve security. Some general algorithmic solutions may be used to increase the security if a good source of randomness is not available. For example, consider the DSA signature scheme; the signature algorithm involves a secret key x related to the public verification key $g^x \bmod p$ and a temporary secret key k that has to be refreshed for each signature. FIPS 186-2 [11] says that k may be randomly or pseudo-randomly generated. This means that the values of k do not have to be perfectly independent (note that if the discrete logarithm problem is hard, k is never revealed).

The secret key x is generated once, usually outside of the smart card, so we can assume it has good randomness properties. Consequently, when a bit string is generated from the available source, the following transformations may be used to increase the security:

- encrypt the bit strings with a stream cipher in order to hide possible repetitions while keeping the same number of available bits,
- combine the obtained bits with the secret key x (using a hash function or a block cipher for example),
- other data, such as a counter and/or a smart card unique serial number, can be added to increase security.

D.3 Statistical tests

Some statistical tests can be used to verify the quality of a random source. **It should be clear that passing those tests is not a sufficient guarantee of randomness!** Consider for example the p digits or the bit string obtained by the concatenation of SHA-1(0), SHA-1(1),… They are bad cryptographic bit strings but they pass all the tests we may imagine.

FIPS 140-1 [9] describes four very simple tests. They can be easily implemented and consequently can be used as a way to check obvious failures of a TRNG.

Let us consider a string of 20000 bits output for a generator. If all the four following tests are passed, the generator passes the test (we insist on the fact that this only means that the generator is not out of order but it does not imply anything on the randomness of generated bits).

- **Test 1**: the number *n* of '1's should satisfy $9654<n<10346$,
- **Test 2**: divide the 20000 bits into 5000 contiguous 4 bit segments. Let *f(i)* be the number of occurrences of *i* interpreted has a segment of 4 bits ($0<=i<=15$). Compute $X=16/5000(f(0)^2+f(1)^2+\ldots f(15)^2)-5000$. The test is passed if $1.03<X<57.4$,
- **Test 3**: a run is a maximal sequence of consecutive bits of either all '1's or all '0's. The number of runs of length 1 of 0's or 1's must be between 2267 and 2733 ; for length 2 between 1079 and 1421, for length 3 between 502 and 748, for length 4 between 223 and 402, for length 5 and for length larger or equal to 6 between 90 and 223,
- **Test 4**: there is no run of length 34 or more.

These tests are described and analysed more precisely in chapter 5 of [HAC]. They are tuned in such a way that a truly random bit string fails any of the tests with probability less than 10^{-6}.

Some publicly available programs implement much more sophisticated tests such as Maurer's universal test [Maur] based on compression techniques. One may for example refer to the NIST test suite [STS].

D.4 Pseudorandom bit generation

D.4.1 General

In applications that require many random bits or when random sources are not available, random-looking bits are generated from an initial random seed using a deterministic algorithm. It should be clear that this does not "create" additional randomness but only derives random looking bits from the randomness initially present in the seed.

The differences between the security levels of PRNGs are determined by the assumptions that are made on the availability of generated bits. Weak PRNGs may generate random-looking bits but the knowledge of a sequence of bits enables the computation of following ones. In strong PRNGs, the knowledge of a sequence of bits does not give information on bits that are subsequently (and even previously) generated. Furthermore, some PRNGs are provably secure based on number theoretical assumptions. A formal classification of PRNGs and their suitability for different cryptographic applications is defined in [21]. Here is a short list of the most well known PRNGs.

D.4.2 ANSI X9.17 generator [20].

This PRNG is designed to pseudorandomly generate keys and initialisation vectors for use of DES. It uses the triple-DES algorithm with a fixed key to mix a 64-bit seed with the current date. Iterated encryption enables to generate as many output bits as needed.

D.4.3 FIPS 186 generator [11].

The digital signature standard proposes two kinds of PRNGs to generate public parameters, secret keys and temporary secret keys. The first one is based on a hash function such as SHA-1 and the second one uses a block cipher such as DES.

D.4.4 RSA PRNG and Blum-Blum-Shub PRNG [HAC].

Those PRNGs are based on iterated exponentiation modulo a composite modulus. The advantage is to base the security on the intractability of number theoretic problem (respectively RSA and the factorisation problem) but the main drawback is the poor efficiency in comparison with the other PRNGs described above, the security of which is only heuristic.

A security analysis of PRNGs appeared in the paper of Kelsey et al. at FSE '98 [KSWH]. Let us remind ourselves of some simple ways they propose to increase the security of existing PRNGs:

- the output may be hashed, using SHA-1 for example, to make direct cryptanalysis of the PRNG much more difficult,
- the initial seed may be hashed with a counter and/or the date to avoid replay attacks,
- the same seed may not be used to generate too many output bits.

D.5 Conclusion

In conclusion, cryptographically secure generation of random bits usually requires the use of a good pseudorandom bit generator initially fed with a good truly generated seed. Both of those primitives (the TRNG and the PRNG) have to be carefully chosen according to the application and the environment in which it is implemented.

Bibliography

The following material, though not specifically referenced in the body of the present document (or not publicly available), gives supporting information.

[BSIK] BSI, "Geeignete Kryptoalgorithmen gemäß § 17 (2) SigV," Entwurf, 27.6.2000.
[AFGS] The Austrian Federal Government, "Signaturverordnung – SigV, " BGBl. II Nr. 30/2000.
[RSAFAQ] RSA Security, Inc., "RSA Laboratories' Frequently Asked Questions About Today's Cryptography," Version 4.1, July 2000, http://www.rsasecurity.com/rsalabs/faq/index.html.
[SECAlg] GMD, "SECUDE: Algorithms," SECUDE-5.1, November 1997 http://www.darmstadt.gmd.de/secude/Doc/htm/algs.htm.
[GaSm] Galbraith, S.D., and N.P. Smart, "A Cryptographic Application of Weil Descent," In *Cryptography and Coding*, M. Walker (ed.), LNCS 1746, Springer-Verlag, 1999.
[JoMe] Johnson, D.B., and A.J. Menezes, "Elliptic Curve DSA (ECDSA): An Enhanced DSA, " at http://www.certicom.com/research/wecdsa.html.
[SECOID] OID tree structure, http://www.darmstadt.gmd.de/secude/Doc/htm/oidgraph.htm
[DLP] I. Damgard, P. Landrock and C. Pomerance, "Average case error estimates for the strong probable prime test," Math. Comp., 61:177-194, 1993.
[STS] "A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications," NIST Special Publication 800-22, 2000, http://csrc.nist.gov/rng.
[HAC] "Handbook of Applied Cryptography (chapter 5)," A. Menezes, P. van Oorschot and S. Vanstone. CRC Press, 1997, http://www.cacr.math.uwaterloo.ca/hac.
[Knuth2] "The art of computer programming – Volume 2 / Seminumerical algorithms (chapter 3)," D. Knuth. Addison-Wesley, 1981.
[Maur] "A universal statistical test for random bit generators," U. Maurer, Advances in Cryptology-CRYPTO '90 (LNCS 537), 409-420, 1991.
[KSWH] "Cryptanalytic Attacks on Pseudorandom Number Generators," J. Kelsey, B. Schneier, D. Wagner and C. Hall, Fast Software Encryption '98 (LNCS 1372), 168-188, 1998.