IEEE P1363 / D13 (Draft Version 13). Standard Specifications for Public Key Cryptography

Annex D (Informative). Security Considerations.

Comments and suggestions are welcome. Please contact the chair, Ari Singer, at singerar@pb.com.

ANNEX D (informative). Security Considerations

D.1 Introduction

This annex addresses security considerations for the cryptographic techniques that are defined in this standard. It is not the intent of this annex to teach everything about security or cover all aspects of security for public-key cryptography. Rather, the goal of this annex is to provide guidelines for implementing the techniques specified in the standard. Moreover, since cryptography is a rapidly changing field, the information provided here is necessarily limited to the published state of the art as of the time the standard was drafted, August 1998. Implementers should therefore review the information against more recent results at the time of implementation; the working group Web page may contain additional relevant information (see http://grouper.ieee.org/groups/1363/index.html).

Security recommendations (in the form of "should") are given throughout this annex. It should be understood, however, that there may be choices other than the ones recommended here that achieve a given level of security. Furthermore, as discussed in D.2, the definition of security depends on the types of attack that are relevant to an implementation. If an attack is not relevant, then some recommendations may not apply. Thus, while the recommendations given here enable security, they should not necessarily be taken as requirements. Nevertheless, it is expected that other standards based on this standard may upgrade some of the recommendations to requirements.

The considerations are presented in six parts. General security principles applying to all the families and schemes are presented in D.2. Key management considerations that also apply to all the families and schemes are summarized in D.3. Family-specific considerations are given in D.4, and scheme-specific considerations are given in D.5. As generation of random numbers is a common tool needed to implement this standard, random number generation considerations are summarized in D.6. Implementation considerations, relevant to all the families and schemes, are given in D.7.

For readers who are interested in extensive and in-depth discussions on security and cryptography, some reference books include [MOV96], [Sch95], [Sta98] and [Sti95].

D.2 General Principles

The storage and transmission of data in modern computer and communications systems is vulnerable to a number of threats, including unauthorized disclosure and modification, as well as impersonation of parties in the system. Parties' assurances of where data is from, whether data is in its original form, who has access to data, and whether a party claiming a certain identity is authentic, all depend on appropriate protection of data.

Policy measures such as requiring a password for access to a system, physical measures such as concealing wires, and legal measures such as prohibiting eavesdropping by statute, all provide a degree of defense against the threats just mentioned. But such approaches are all indirect, as they address only the implementation or use of a system—not the data itself. Direct protection, involving operations on the data, is the purpose of cryptography.

As noted in Section 4 of this standard, three common means of data protection provided by public-key cryptography are key agreement schemes, encryption schemes, and digital signature schemes. Implemented and combined appropriately, they can address the threats just mentioned. With either a key agreement scheme or an encryption scheme, for instance, parties can establish a shared secret key. The parties can then prevent unauthorized disclosure by applying a symmetric encryption algorithm to the data using the shared secret key, so that parties not possessing the key cannot recover the data. A party can also encrypt data directly with an encryption scheme. Similarly, with a signature scheme, parties can detect unauthorized modification as well as impersonation.

The need for the protection of data and identities against the threats mentioned above leads to consideration of the following question: "What is a secure system?" There is no easy answer to this question. As the *Handbook of Applied Cryptography* states, "… security manifests itself in many ways according to the situation and requirement" [MOV96, Section 1.2]. The following paragraphs explore some of the general principles that help answer this question.

In a typical security analysis, an *opponent* (also called an *adversary*)—the source of the threats mentioned above—is assumed to have certain abilities. Under the usual model, called Kerckhoffs' assumption [Ker83], an opponent will have full knowledge of system design, including the specification of all cryptographic operations (as, for instance, this standard would provide). Usually, the opponent will be able to use more tools than just "passive wiretapping" in order to read messages exchanged between parties. For example, the opponent may also be able to modify the message between parties. The opponent should also be considered to have some computational resources. The opponent may also be trusted by other parties to some extent. For instance, legitimate parties may be willing to perform selected cryptographic operations requested by the opponent, or the opponent's keys may be accepted as legitimate by other parties. In other words, the opponent may appear to other parties as a legitimate party. However, the opponent generally will not know all of the secrets in the system, such as long-term private keys, and it is this knowledge that distinguishes the opponent from the legitimate parties. Any attempt by the opponent to violate the security of a system in some manner is called an *attack*.

Under the assumption that such an opponent is present, the legitimate parties need to ensure that certain *security properties* (such as, for example, data confidentiality or data integrity) are satisfied. Just as there are different classes of opponent, there are different desirable security properties. A higher level of security is likely to be desirable for more valuable data, for instance, or for data with a long lifetime. Thus, as a general principle, security properties should be selected consistent with the value of the data being protected and the set of opponents envisioned. Since the cost of providing security generally varies according to the security properties, decisions related to security are appropriately framed as cost-benefit analyses.

In light of the above discussion, security may be defined as the *assurance of trust in the face of opponents with certain knowledge and resources*. Thus, the opponent, and the types of attack it can mount, is a required component of the definition of security.

A designer's objective, then, is to choose security techniques that provide certain desired security properties when faced with an opponent considered reasonable for a particular system. The cost-benefit analysis that can be applied to make decisions related to security can also be applied when evaluating the capabilities of a potential opponent. For example, an opponent is unlikely to spend more on breaking the system than it will obtain by doing so. However, it is important not to overestimate the opponent's costs, because an opponent may be using stolen, borrowed or free resources (such as spare cycles of multiple computers). Indeed, in the case of free resources, the attack is always worth doing in a cost-benefit analysis, assuming the cost is really zero. Also, a large one-time investment for an opponent may pay off over time if the opponent breaks multiple systems. For example, if many DL-based systems use one particular finite field, a large investment in accelerated hardware modules for solving the DL problem in that specific field may pay off for the opponent, especially because the computation time per logarithm goes down if multiple discrete logarithms need to be computed. It is also important not to underestimate the opponent's benefit—it may be unclear how much publicity, personal satisfaction, or furthering of political goals may be worth to a potential opponent. Finally, one must also consider risk in terms of an opponent's probability of success. Even a chance of one in one million may be unacceptably large in some situations, so it is not only the cost of guaranteed success that is relevant.

In support of this process of choosing appropriate techniques when faced with a potential opponent, this annex summarizes the choices related to cryptographic techniques in the standard. Note, however, that this standard (and this Annex in particular) focuses on individual implementation of individual cryptographic techniques, and not on entire systems. In particular, this standard defines cryptographic techniques from the point of view of a single party, rather than protocols in which multiple parties participate and multiple cryptographic operations are performed. It is expected that techniques in this standard will be combined into protocols by other standards or by implementers. However, such combinations, especially ones where multiple cryptographic operations (e.g., both encryption and signature) on the same data are performed require additional security analyses.

D.3 Key Management Considerations

This section provides general information on key management. Proper key management is an essential component of cryptographic security. It is needed, in particular to ensure integrity, authenticity, and where appropriate, secrecy of domain parameters and keys.

D.3.1 Generation of Domain Parameters and Keys

A set of domain parameters may be generated by one of the parties that intend to use it, by a third party, or jointly by any subset of these. A capability to audit the domain parameter generation may be provided to other parties by generating random components of the domain parameters using one-way function(s) of random seed(s), and publishing the seed(s). Any party can then verify that the domain parameters indeed correspond to the seed(s). This provides some degree of assurance that the party generating the parameters did not pick them specifically to possess some particular special property. The property in question should be rare enough so that it is infeasible to obtain it by simply trying different seeds. The one-way function should be such that it is hard to find a seed that produces parameters with that property (for example, the property should not be closely related to the one-way function). See Annex A.12.4-A.12.7 for an example of parameter generation that provides for auditing. Additional information for auditing domain parameter generation is provided by family in D.4.

A public-private key pair may be generated as follows:

— by the owner of the keys (this is the most commonly used method)
— jointly by the owner of the keys and by another party, such as a certifying authority. Depending on the technique used, the other party may not need to be entrusted with any secrets by the owner of the key pair. This method may be used to ensure that the key pair is not picked by the owner to have some particular special property, and may eliminate the need for a separate verification that the party possesses its private key (see Annex D.3.1). Chen and Williams [CW98] provide an example of this method.
— by a third party that can be trusted with the private key (in particular, it should be trusted to keep the private key secret and not use its knowledge of the private key to its advantage). If the third party is trusted to perform the key generation properly, this method may be used to ensure that the key pair is not picked by the owner to have some particular special property, and may eliminate the need for a separate verification that the party possesses its private key (see Annex D.3.1).

A capability to audit the key generation may be provided using methods similar to those for the parameter generation, as described above. However, because, as opposed to parameters, the private key is meant to be a secret, the random seed should be kept secret, as securely as the private key, because revealing it would reveal the private key. Therefore, the actual auditing must be done only by trusted parties or in situations when it is appropriate to reveal the private key.

Private keys should never be shared among parties. Every private key should be generated independently of other private keys. If random numbers for use in producing the private keys are generated properly, accidental sharing of private keys is extremely unlikely to occur (see Annex D.6 for more on random number generation). Two or more dishonest parties, however, may collude in order to make their private keys be the same (or stand in some relation known to the colluding parties). Dishonest parties may do so, for example, for the purposes of claiming the failure of a key generation procedure and for repudiating signatures produced with those keys (see Annex D.5.2.3 for more on repudiation). Such concerns may be addressed by establishing trust in the implementation of key generation, or by involving a trusted party in the key generation process as described above. It may be possible for an authority to check that no two public keys (and, hence, private keys) are equal in a limited system. However, this check will not reveal a pair of keys that stand in some special relation to each other and may be impractical to implement in some systems.

More considerations for key and parameter generation are given by family in D.4.

D.3.2 Authentication of Ownership

Authentication of ownership of a public key is the assurance that a given, identified party intends to be associated with the public key (and the corresponding set of domain parameters, if any). It may also include assurance that the party possesses the corresponding private key (this is commonly known as Proof of Possession or POP). The latter assurance is stronger in the sense that it prevents an opponent from misleading other parties into believing that the results of operations performed with the private key are associated with the opponent, rather than with the actual owner of the private key.

Further considerations related to authentication are given by type of scheme in D.5.

Authentication of ownership may be conveyed as part of the supporting key management, for instance in a *certificate*, which is a message signed by a certifying authority indicating that a particular party, typically identified by a name, owns a given public key (see [MOV96, Section 13.4.2]). Anyone who knows the certifying authority's public key and trusts the certifying authority can verify the signature on the certificate and thereby authenticate the party's ownership of the public key. Certifying authorities may issue certificates to other certifying authorities, so that trust in a single "root" certifying authority's public key can extend through a chain of certificates to the public keys of a large community. Other attributes, such as key cryptoperiod or usage restrictions, may also be bound to a public key in this manner (see Annex D.3.6, [ITU97]). To gain assurance that a party possesses a private key, and thereby to convey this assurance, a certifying authority should perform an appropriate protocol verifying the possession before issuing the certificate. Such methods may be found in [MOV96, Sections 10.3.3, 10.4 and 13.4.2], [AM98], [SAK98].

Another way of conveying authentication of ownership is for a party to issue its own certificate, i.e., to sign the message containing the public key with a separate private key. This approach is helpful when the public key has a short cryptoperiod and it is impractical to obtain a certificate from a certifying authority. However, this approach does not necessarily provide assurance that the party possesses the corresponding private key.

A public key should be securely associated with its set of domain parameters (if any). This may be accomplished by including the set of domain parameters into the certificate. The set of domain parameters may also be embedded into the system if, for example, the system implementation only performs operations with a single set of domain parameters. Domain parameters should be authenticated and protected from unauthorized modification in the same manner a public key is.

In general, a party's possession of a private key should be authenticated as part of authenticating the party's association with a public key. Situations in which it may be acceptable not to do so are given by type of scheme in D.5.

D.3.3 Validation of Domain Parameters and Keys

Most primitives specified in this standard are not defined when an input set of domain parameters or key is invalid (see Section 4.2 and Annex B). Implementations should therefore appropriately address this possibility. The risks of operating on invalid domain parameters or keys vary depending on the scheme used and the particular implementation, and are addressed by type of scheme in D.5.

DL and EC domain parameters and keys may be validated explicitly before being input to a primitive using, for example, the techniques given in A.16. Note that no techniques for private key validation are provided, because, generally, a party controls its own private key and need not validate it. However, private key validation may be performed if desired. Also note that no techniques for IF public key validation are provided in this standard; however, see [LS98] and [GMR98] for some proposed techniques.

Alternatively, domain parameters may be validated within the infrastructure by an authority. For example, a certifying authority may validate domain parameters and keys as part of issuing a certificate; certificate verification will then imply validity of the domain parameters or keys it contains. Generation of keys or parameters by a trusted authority or jointly with it (see Annex D.3.2) may provide assurance of their validity. As another means of domain parameter validation, a standards body may publish a set of domain parameters, in effect serving as a trusted third party assuring their validity. Such domain parameters have been published, for example, in FIPS 186-1 [FIP94b], ANSI X9.30 [ANS97a] and ANSI X9.42 [ANS98b] for the DL family and ANSI X9.62 [ANS98e], ANSI X9.63 [ANS98f], GEC1 [SEC99] and [NIS99] for the EC family.

The above methods ensure that keys satisfy their definitions (i.e., are valid). They do not, however, provide an assurance that the keys were generated in a secure manner. Such assurance, in addition to the assurance of validity, may be provided by ensuring that only appropriately validated modules can be used for generating keys (see Annex D.7). (In contrast, domain parameters may potentially be generated in unvalidated modules, provided that the parameters are validated, since no secrets are involved.)

Public key validation (PKV) and POP (see D.3.2) can be considered as duals. PKV shows that the party's public key is valid, but not necessarily that the party possesses the corresponding private key. POP demonstrates that the party possesses the private key, but not necessarily that the public key is valid (though certain methods for POP may demonstrate both). Both PKV and POP should be considered in high assurance applications, unless the risk of operating with invalid keys or without assurance that a party possesses a private key is mitigated by other means.

Some additional ways to address the risks of operating on invalid domain parameters and keys are given by type of scheme in D.5.

D.3.4 Cryptoperiod and Protection Lifetime of Domain Parameters and Keys

The *cryptoperiod* of a set of domain parameters or a key is the period during which operations may be performed with the set of domain parameters or the key. A set of domain parameters or a key should have an appropriate cryptoperiod to limit the amount of data whose protection is at risk (in the event of cryptanalytic attack or physical compromise of a private key) and the amount of data available for cryptanalysis. The cryptoperiod of a set of domain parameters should be at least as long as the cryptoperiod of keys associated with the set of domain parameters. The cryptoperiod of a public key used for signature verification should be at least as long as the cryptoperiod of the corresponding private key for signature generation; the cryptoperiod of a private key used for decryption should be at least as long as the cryptoperiod of the corresponding public key for encryption.

Keys with long cryptoperiods are known as *long-term* or *static*; keys with short cryptoperiods are known as *short-term* or *ephemeral*. These terms are most commonly used in the context of key agreement schemes (see Annex D.5.1). Note that whether a key is static or ephemeral is independent of whether or not it is authenticated.

The *protection lifetime* of a key is the amount of time that the key is needed to protect data, and is generally longer than the cryptoperiod of a key. For example, even after the cryptoperiod of an encrypting key expires, the data protected with it may still be sensitive; similarly, the data signed by a signing key may need to be protected from unauthorized modification long after the cryptoperiod of the signing key has expired. Hence, the security measures for a key (including the appropriate key sizes) should be picked primarily according to its protection lifetime, rather than its cryptoperiod.

In general, there is a distinction between the risk of an opponent learning a private key through physical compromise, versus an opponent learning a private key through cryptanalysis. Cryptanalysis may be defended against by appropriate key sizes and by limiting the number of operations performed with a given key and thus the availability of data to the cryptanalyst. The risk of physical compromise of a given key and the value of the key to the opponent may be reduced by giving a private key a short cryptoperiod and erasing the key thereafter.

Implications of domain parameter and key cryptoperiods are described further by type of scheme in D.5.

The cryptoperiod of domain parameters and public keys may be conveyed as part of the supporting key management, for instance as certificate attributes. The cryptoperiod should be securely associated with the parameters and keys and protected from unauthorized modification (see Annex D.3.6). To limit the impact of a successful cryptanalytic attack or a physical compromise, provision should also be made in supporting key management for early termination of a set of domain parameters or a key, for instance through certificate revocation (see, e.g., [MOV96, Section 13.6.3], [ITU97], [SHW98]).

D.3.5 Usage Restrictions

A set of domain parameters or a key should have appropriate restrictions on its use to prevent misapplication of the set of domain parameters or the key as input to an operation, as well as misinterpretation of the results of an operation.

Examples of usage restrictions include:

- restrictions on the type of scheme—for instance, only a signature scheme, or only an encryption scheme
- restrictions on scheme options—e.g., only a particular encoding method or hash function for a signature scheme
- restrictions on messages processed—for instance, only payment orders of a certain format, up to a certain value

As a prudent security practice, the use of a particular key should be restricted to a single scheme with a single specific set of scheme options. Otherwise, the variability of data associated with the key may aid an opponent. If a single key is to be used for a variety of schemes or scheme options, additional security analysis is required. A set of domain parameters may be identified as intended to be used with a single specified scheme, a set of specified schemes, or with all schemes to which it applies in a given family. (The granularity of what constitutes a "scheme", e.g., in terms of scheme options, depends on the implementation and on what is considered to constitute an attack.)

Usage restrictions should be securely associated with the parameters and keys and protected from unauthorized modification (see Annex D.3.6). To accomplish this, they may be conveyed as part of the supporting key management, for instance as certificate attributes (see, e.g., [ITU97]). They may also be embedded (explicitly or implicitly) into a system: for example, in a closed system that is only capable of producing and verifying one specific type of signatures, there may be no need to convey restrictions on the type of scheme or scheme options as part of key management.

D.3.6 Storage and Distribution Methods

The storage and distribution methods for domain parameters and keys should provide appropriate protection.

It is presumed that domain parameters and keys will be identified somehow (perhaps by the owner's identity) and possibly associated with certain attributes, such as ownership, cryptoperiod, and usage restrictions. It is important to protect the integrity of this identification and association when keys or domain parameters are stored and distributed. Specifically, it should be difficult for an opponent to modify the key or set of domain parameters, its identification, or any attributes that are associated with the key or set of domain parameters. The specific protections depend on the component:

- a set of domain parameters should be protected from unauthorized modification, and generally need not be protected from disclosure
- a public key should be protected from unauthorized modification, and generally need not be protected from disclosure
- a private key should be protected from unauthorized modification and, with the possible exception of its associated set of domain parameters, if any, protected from unauthorized disclosure
- identification information and attributes associated with a key or set of domain parameters should be protected from unauthorized modification

The security properties for a storage or distribution method should be commensurate with the security properties intended to be provided by a set of domain parameters or key. A typical protection method for domain parameters and public keys is to bind them with identification information and attributes in a certificate, which may be stored or distributed; the signature on the certificate protects against unauthorized modification. Another method, commonly used for "root" public keys and domain parameters (such as public keys of root certifying authorities—see Annex D.3.1) is to embed them in software or hardware in such a way that they become an inherent part of the system and are protected from unauthorized modification by the same mechanisms as the system itself (see Annex D.7).

When a key is associated with set of domain parameters, the set of domain parameters may either be stored and distributed explicitly with the key, or referenced implicitly. When the set of domain parameters is referenced implicitly, the reference and the set itself should be protected from unauthorized modification. For instance, suppose a key references a shared set of domain parameters. The reference to the shared set should be unambiguous, and the set itself should be protected from unauthorized modification.

D.4 Family-Specific Considerations

This section gives further information on security parameters for each of the three families, as well as generation of domain parameters (if any) and key pairs.

D.4.1 DL Family

D.4.1.1 Security Parameters

The primary security parameters for the DL family are the length in bits of the field order q and the length in bits of the subgroup order r. A common minimum field order length is 1024 bits and a common minimum subgroup order length is 160 bits (see ANSI X9.42 [ANS98b]). (Note 1.)

The field type (prime vs. binary) may affect the security of the schemes. (Note 2.)

D.4.1.2 Generation Method

Considerations for generating domain parameters in the DL family include the following:

— *(Prime case.)* The field order q (i.e., the prime p) should be generated randomly or pseudorandomly among those field orders with a sufficiently large subgroup, with a prime generation method that has a sufficiently low probability of producing a non-prime. For a canonical method of domain parameter generation that can be audited, the field order should be generated as a one-way function of a random seed. (Note 3.)
— *(Binary case.)* The field order q may be predetermined. In a canonical method of domain parameter generation that can be audited, the field order should be selected from a set of allowed values. (See Annex A.16.3–A.16.4 for one such method.) (Note 4.)
— The subgroup order r may have any value consistent with the other parameters, provided that it is sufficiently large and prime (and primitive in the case of binary fields—see Note 5); it may be predetermined, generated randomly or pseudorandomly, or derived from the field order. Prime generation or primality testing for the subgroup order may admit the possibility of error, provided the probability of error is sufficiently low. (Note 5.)
— The base g may have any value consistent with the other domain parameters. (Note 6.)

Considerations for generating key pairs in the DL family include the following:

— The private key s should be generated at random from the range $[1, r - 1]$ (or a sufficiently large subset of it). (Note 7.)

The domain parameters q, r, and g (and the field representation in the binary case) may be shared among key pairs. The private key s should not be shared. (Note 8.)

D.4.1.3 Other Considerations

The field representation (in the binary case) is not known to affect security, although since the field representation is a domain parameter, it should be protected from unauthorized modification, along with the other domain parameters. (Note 9.)

D.4.1.4 Notes

1. *(Security parameters.)* The security of schemes in the DL family against attacks whose goal is to solve the discrete logarithm problem depends on the difficulties of general-purpose discrete logarithm methods and of collision-search methods. In turn, the difficulty of general-purpose discrete logarithm methods depends on the length in bits of the field order q and the difficulty of collision-search methods depends on the length in bits of the subgroup order r.

   For large prime fields, the fastest general-purpose discrete logarithm method today is the generalized number field sieve (GNFS) (see [MOV96, Section 3.6.5], [Gor93b], [Sch93]), which has asymptotic running time $\exp(((64/9)^{1/3} + o(1)) (\ln q)^{1/3} (\ln \ln q)^{2/3})$, where $o(1)$ denotes a number that goes to zero as q grows. For more on estimating the complexity of GNFS for large prime fields, see Note 1 in D.4.3.4; the table given there also applies except that the memory requirements for the linear algebra are $\log_2 q$ times greater for the discrete logarithm problem than they are for the integer factorization problem. (See also the web site `http://www.rsa.com/rsalabs/html/factoring.html` on the status of solving the integer factorization problem. The result can be used as an estimate for an appropriate field order in the discrete logarithm system.) For large binary fields, a similar method (see [MOV96, Section

3.6.5], [GM93]) has asymptotic running time within a constant factor of $\exp(1.587 (\ln q)^{1/3} (\ln \ln q)^{2/3})$. In particular, the method is faster for binary fields than for prime fields.

For both types of field, the Pollard rho method ([Pol78], [MOV96, Section 3.6.3]), and the related Pollard lambda and other collision-search methods (see, e.g., [OW94]) can compute discrete logarithms after, on average, $(p\, r/4)^{1/2}$ field multiplications. The memory requirements of such methods are relatively small. (See also `http://www.certicom.ca/chal/index.htm` on the status of solving the elliptic curve discrete logarithm problem. The result can be used as an estimate for an appropriate subgroup order in the discrete logarithm system.)

The length of the field order and the length of the subgroup order should be selected so that both the general-purpose methods and the collision-search methods have sufficient running time. Often, the parameters are selected so that the difficulty of both types of method is about the same. It does not have be the same, however. For a variety of reasons, such as availability of hardware, for example, an implementation may choose a larger field or subgroup. As noted, a common minimum field order length is 1024 bits, and a common minimum subgroup order length is 160 bits.

When a set of domain parameters is shared among parties, the size should also take into account the number of key pairs associated with the set, since the total running time for computing k discrete logarithms with the same set of domain parameters is only about $k^{1/2}$ times the running time of computing a single discrete logarithm, provided that more memory is available [OW94]. In general, a set of domain parameters that is shared among a large number of key pairs should have larger security parameters than one that is shared among a small number of key pairs. A prudent practice is to pick security parameters such that computing even a single discrete logarithm is considered infeasible.

2. *(Field type: prime vs. binary.)* The field type may affect the security of the schemes, since the running time of general-purpose discrete logarithm methods differs between prime fields and binary fields. Also, since the cost of finite field operations may differ between the two types of field, the actual running time of collision-search methods may vary by a small factor. Furthermore, since there is only one binary field for each field order length, there are fewer alternatives to choose from, should some binary field be cryptanalyzed, than should a single prime field be cryptanalyzed.

3. *(Prime generation.)* The field order q (i.e., the prime p) should be generated randomly or pseudorandomly, since this provides resistance to special-purpose discrete logarithm methods such as the Special Number Field Sieve, or those given in [Gor93a]. The prime generation method should have a sufficiently low probability that a non-prime is generated, so that the probability of generating an invalid set of domain parameters is small. A small but nonzero probability of error (say, $< 2^{-100}$) is acceptable since it makes no difference in practice; methods with a small probability of error are generally simpler than those with zero error. See Annex D.6 for more on generating random numbers and A.15 for more on generating primes.

A desired security level can also be provided when the field order has a special form; such a choice requires further security analysis by the implementer.

Computing the field order as a one-way function of a random seed provides a degree of auditing since it makes it difficult to select a prime with some predetermined rare property. It may not entirely prevent the party generating the prime from producing primes with special properties, since the party can try many different seeds, looking for one that yields the desired property, if the property is likely enough to occur. However, it will make it difficult for a party to produce primes that are vulnerable to a special-purpose discrete logarithm method such as the Special Number Field Sieve, provided that the prime is large enough. (A party might seek to introduce such a vulnerability in the interest of determining users' private keys.) It will also make it difficult to mount an attack on DSA in which the opponent picks a particular subgroup order r in order for two messages to have the same signature (see [Vau96]). (This attack is of concern if the message representative is not shorter than the subgroup order r.) FIPS 186 [FIP94b], ANSI X9.30:1 [ANS97a] and X9.42 [ANS98b] give an auditable method of generating primes by incremental search.

4. *(Field order generation, binary case.)* The field order q may be predetermined, since there is only one binary field for each field order length, and random generation is not meaningful. For the same reason, in canonical domain parameter generation, the field order is selected from a list of allowed values, not generated at random.

5. *(Subgroup order.)* The subgroup order r may have any value consistent with the other domain parameters (provided it is a primitive divisor of $q - 1$ in the case of binary fields), since the difficulty of the collision-search methods for the discrete logarithm problem depends only on the size of the subgroup order, not on its particular value, as long as it is prime. Selecting a field order that is larger than the size of the hash value employed in a signature scheme, as is done in ANSI X9.62 [ANS98e] has the benefit of thwarting Vaudenay's attack [Vau96]. A small but nonzero probability of error in prime generation or primality testing (say, $< 2^{-100}$) is acceptable since it makes no difference in practice. The subgroup order r should be a primitive divisor of $q - 1$ in the case of binary fields (see Annex A.3.9), because otherwise the subgroup is contained entirely in a proper subfield $GF(2^d)$ of $GF(2^m)$, in which case the opponent need only solve the DL problem in the smaller field $GF(2^d)$.

6. *(Base.)* The base g may have any value consistent with the other domain parameters since the difficulty of the discrete logarithm problem does not depend on which base is employed for a given subgroup. All bases are equivalent in the sense that if an opponent can compute discrete logarithms with respect to one base, say h, the opponent can compute discrete logarithms with respect to any other base for the same subgroup, say g, by taking a ratio:

$$\log_g w = \log_h w / \log_h g \bmod r .$$

Computing the base g as a one-way function of a random seed provides a degree of auditing since it makes it difficult to select a predetermined base, such as one that depends on another user's public key, as in [Vau96]. In particular, it thwarts attacks in which the opponent ensures that two parties have different (but related) bases while thinking that they have the same set of parameters ([Vau96]). Such attacks may also be prevented by ensuring that both parties have the same set of parameters, for example, by verifying the authenticity of the parameters, as described in D.3.1.

7. *(Private keys.)* The private key should be generated at random from the range $[1, r-1]$ since this maximizes the difficulty of recovering the private key by collision-search methods. A desired level of security can also be provided when the private key is restricted to a large enough subset of the range, e.g., is shorter than the subgroup order, has low weight, or has some other structure. Such choices require further security analysis by the implementer. See also D.6 for more on random number generation.

8. *(Domain parameters, public key and private key sharing.)* A set of domain parameters may be shared, since by definition, the set of domain parameters may be common to any number of keys. A private key should not be shared.

9. *(Field representation.)* The field representation is not known to affect security; its only cryptographic roles are in the conversion of field elements to integers in the signature and verification primitives, and to octet strings in the key agreement schemes. The choice of field representation for domain parameters and keys is otherwise a matter of data formatting. In the two

cryptographic roles mentioned, no reason is currently known why the field representation should have any impact on security. However, it is important that the parties to a scheme agree on the representation in which conversion is to take place, since otherwise an opponent may be able to trick the parties into operating with a different representation for conversion in an attempt to forge signatures or obtain information about keys.

D.4.2 EC Family

D.4.2.1 Security Parameters

The primary security parameter for the EC family is the length in bits of the subgroup order r. A common minimum subgroup order length is 161 bits (see ANSI X9.62 [ANS98e]). (Note 1.)

The field type (prime vs. binary) may slightly affect the security of the schemes. (Note 2.)

D.4.2.2 Generation Method

Considerations for generating domain parameters in the EC family include the following:

— *(Prime case.)* The field order q (i.e., the prime p) may have any value that provides a sufficiently large subgroup; it may be predetermined, or generated randomly or pseudorandomly. Prime generation for the field order may admit the possibility of error, provided the probability of error is sufficiently low. (Note 3.)
— *(Binary case.)* The field order q may be predetermined. (Note 4.)
— The elliptic curve may be any curve satisfying the definition in Section 5.4 as well as the MOV and anomalous criteria (see Note 1) that provides a sufficiently large prime-order subgroup. It may be predetermined, or generated randomly or pseudorandomly, perhaps subject to certain conditions that simplify the computation of the number of points on the elliptic curve. For example, a 169-bit Koblitz curve may be estimated to be about 13 times faster to solve than a canonical random 169-bit curve, though in practice the additional calculations may result in a smaller speedup. This corresponds to a decrease of about 3.5 bits of symmetric key exhaustion security, or 7 bits of elliptic curve security. For a canonical random method of domain parameter generation that can be audited, the elliptic curve coefficients should be generated as a one-way function of a random seed (see Annex A.12.4-A.12.7 for one such method). (Note 5.)
— The subgroup order r may have any value consistent with the other domain parameters, provided that it is sufficiently large and prime. Prime generation or primality testing for the subgroup order may admit the possibility of error, provided the probability of error is sufficiently low. (Note 6.)
— The base point G may have any value consistent with the other domain parameters. (Note 7.)

Considerations for generating key pairs in the EC family include the following:

— The private key s should be generated at random from the range $[1, r-1]$ (or a sufficiently large subset of it). (Note 8.)

The domain parameters may be shared among key pairs. The private key s should not be shared. (Note 9.)

D.4.2.3 Other Considerations

The field representation (in the binary case) is not known to affect security, although since the field representation is a domain parameter, it should be protected from unauthorized modification, along with the other domain parameters. (Note 10.)

D.4.2.4 Notes

1. *(Security parameters.)* The security of schemes in the EC family against many attacks whose goal is to solve the elliptic curve discrete logarithm problem depends on the difficulty of collision-search methods, which are the fastest methods known today for computing discrete logarithms on an arbitrary elliptic curve. The difficulty of collision search methods depends, in turn, on the length in bits of the subgroup order r. The Pollard rho [Pol78], and the related Pollard lambda and other collision-search methods (see, e.g., [OW94]) can compute elliptic curve discrete logarithms after, on average, $(p\ r/4)^{1/2}$ elliptic curve additions. The memory requirements of such methods are relatively small. For special classes of curves, improved collision-search methods are available. In particular, curves not satisfying the MOV condition (see Annex A.12.1) and anomalous curves over prime fields (those where the order r of the base point is equal to the field order q, and, hence, by the Hasse bound (A.9.5), equal to the curve order—see [Sma98] and [SA98]) are considered insecure and are thus avoided by this standard. Collision-search methods may also be sped-up by a factor of about $(m/e)^{1/2}$ for curves over a binary field $GF(2^m)$ whose coefficients lie in a smaller subfield $GF(2^e)$ (see [GLV98] and [WZ98]); this improvement is not considered significant enough to warrant avoidance of such curves, but it may suggest higher key sizes than for general curves. (See also `http://www.certicom.ca/chal/index.htm` for more information on the status of solving the elliptic curve discrete logarithm problem.)

   The following table provides an estimate for the running time of collisions search methods for different sizes of r. The estimate is given in MIPS-Years, where a MIPS-Year is an approximate amount of computation that a machine capable of performing one million arithmetic instructions per second would perform in one year (about 3×10^{13} arithmetic instructions).

Size of the generator order *r* (Bits)	**Processing Time (MIPS-Years)**
128	4.0×10^5
172	3×10^{12}
234	3×10^{21}
314	2×10^{33}

Table D.1: Estimated Cryptographic Strength for EC Generator Order Sizes

   There is some variation among published estimates of running time due to the particular definition of a MIPS-Year and to the difficulty of estimating actual processor utilization. (How many arithmetic instructions a modern processor performs in a second when running an actual piece of code depends heavily not only on the clock rate, but also on the processor architecture, the amount and speeds of caches and RAM, and the particular piece of code.) Thus, the estimates given here may differ from others in the literature, although the relative order of growth remains the same. The length of the field order and the length of the subgroup order should be selected so that the collision-search methods have sufficient difficulty. A common minimum subgroup order length is 161 bits.

   When a set of domain parameters is shared among parties, the size should also take into account the number of key pairs associated with the set, since the total running time for computing k elliptic curve discrete logarithms with the same set of domain parameters is only about $k^{1/2}$ times the running time of computing a single elliptic curve discrete logarithm, provided that more memory is available [OW94]. In general, a set of domain parameters that is shared among a large number of key pairs should have larger security parameters than one that is shared among a small number of key pairs. A prudent practice is to pick security parameters such that computing even a single elliptic curve discrete logarithm is considered infeasible.

2. *(Field type: prime vs. binary.)* The field type may slightly affect the security of the schemes, since the cost of finite field operations may differ between the two types of field, and the actual running time of collision-search methods may vary by a small factor. Furthermore, since there is only one binary field for each field order length, there are fewer alternatives to choose from should some binary field be cryptanalyzed, than should a single prime field be cryptanalyzed; however, in either case there are many elliptic curves to choose from, and no method is known that cryptanalyzes all elliptic curves over a given field at the same time.

3. *(Prime generation.)* No special primes are known where computing elliptic curve discrete logarithms might be substantially easier. A small but nonzero probability of error in prime generation or primality testing (say, $< 2^{-100}$) is acceptable since it makes no difference in practice; methods with a small probability of error are generally simpler than those with zero error. See Annex A.15 for more on generating primes.

4. *(Field order generation, binary case.)* The field order q may be predetermined, since there is only one binary field for each field order length.

5. *(Elliptic curve.)* The elliptic curve may be any curve satisfying the definition of Section 5.4 as well as MOV and anomalous criteria (see Note 1) that provides a sufficiently large prime-order subgroup, as no other special classes of elliptic curves are known where computing elliptic curve discrete logarithms might be substantially easier. The elliptic curve may be predetermined, or generated randomly or pseudorandomly, perhaps subject to certain conditions that simplify the computation of the number of points on the elliptic curve. Of the methods for curve generation described in A.9.5, method 1 (selecting a random curve and computing its order) imposes no additional conditions on the curve, and the other three methods impose additional conditions. In particular, method 3 (selecting a curve over a subfield) imposes additional conditions that make the discrete logarithm computation on the curve subject to the speed-up due to [GLV98] and [WZ98], as further described in Note 1 above.

   Computing the elliptic curve coefficients (or other inputs to elliptic curve generation) as a one-way function of a random seed provides a degree of auditing since it makes it difficult to select an elliptic curve with some predetermined rare property. It may not entirely prevent the party generating the elliptic curve from producing curves with special properties, since the party can try many different seeds, looking for one that yields the desired property, if the property is likely enough to occur. However, it will make it difficult to mount an attack in which the opponent picks a particular subgroup order r in order for two messages to have the same signature (see [Vau96] for such an attack on DSA). (See also Note 6.) See Annex A.12.4-A.12.7 for an example of such a method.

6. *(Subgroup order.)* The subgroup order r may have any value consistent with the other domain parameters, since the difficulty of the collision-search methods for the elliptic curve discrete logarithm problem depends only on the size of the subgroup order, not on its particular value, provided that the subgroup order is prime. Selecting a field order that is larger than the size of the hash value employed in a signature scheme, as is done in ANSI X9.62 [ANS98e] has the benefit of thwarting Vaudenay's attack [Vau96]. A small but nonzero probability of error in prime generation or primality testing (say, $< 2^{-100}$) is acceptable since it makes no difference in practice.

7. *(Base.)* See Note 6 of D.4.1.4, replacing “base g” with “base point G”.

8. *(Private keys.)* See Note 7 of D.4.1.4.

9. *(Domain parameters, public key and private key sharing.)* See Note 8 of D.4.1.4.

10. *(Field representation.)* See Note 9 of D.4.1.4.

D.4.3 IF Family

D.4.3.1 Security Parameter

The primary security parameter for the IF family is the length in bits of the modulus n. A common minimum length is 1024 bits (see ANSI X9.31 [ANS98a]). (Note 1.)

The key type (RSA vs. RW) is not known to affect the security of the schemes with the recommended encoding methods. (Note 2.)

D.4.3.2 Generation Method

Considerations for generating keys in the IF family include the following:

— The lengths of the primes p and q should be approximately half the length of the modulus n, although other lengths may also provide sufficient security. See Annex A.16.11 and A.16.12 for examples. (Note 3.)
— The primes p and q should be generated randomly or pseudorandomly with a prime generation method that has a sufficiently low probability of producing a non-prime. If auditing of key generation is required, then the primes should be generated as a one-way function of a random secret seed. (Note 4.)
— The public exponent e may have any value consistent with the modulus and key type; it may be predetermined, or generated randomly or pseudorandomly. (Note 5.)
— The private exponent d should be derived from the public exponent e or generated at random. (Note 6.)

The public exponent e may be shared among keys. The modulus n, the primes p and q, and the private exponent d should not be shared. (Note 7.)

D.4.3.3 Other Considerations

The private-key representation does not affect security in general, although the effectiveness of certain physical attacks may vary according to the representation. The private key should be stored securely regardless of the representation, as discussed in D.7. (Note 8.)

D.4.3.4 Notes

1. *(Modulus size.)* The security of schemes in the IF family against attacks whose goal is to recover the private key depends on the difficulty of integer factorization by general-purpose methods, which, in turn, depends on the length in bits of the modulus n. For large moduli, the fastest general-purpose factoring method today is the generalized number field sieve (GNFS) (see [BLP93], [BLZ94]), which has asymptotic running time $\exp(((64/9)^{1/3} + o(1)) (\ln n)^{1/3} (\ln \ln n)^{2/3})$, where $o(1)$ denotes a number that goes to zero as n grows. Previously, the fastest general-purpose method was the multiple polynomial quadratic sieve (MPQS) (see [Sil87]), which has running time of about $O(\exp(\ln n \ln \ln n))$. See [MOV96, Section 3.2] and [Odl95] for more discussion on integer factorization. (See also `http://www.rsa.com/rsalabs/html/factoring.html` for more information on the status of solving the integer factorization problem.)

In order to compute the complexity of the GNFS accurately for a particular n, one needs to know the precise value of the $o(1)$ term for that n. In particular, the complexity of the GNFS is difficult to measure accurately for numbers of cryptographic interest, because they are larger than GNFS can currently factor. The standard practice is to estimate the complexity by extrapolating from running times observed for factoring smaller numbers. Following this, ANSI X9.31 [ANS98a] gives the following estimates for GNFS factorization at various modulus sizes. *Processing time* is the total computational effort, given in MIPS-Years (a MIPS-Year is an approximate amount of computation that a machine capable of performing one million arithmetic instructions per second would perform in one year [about 3×10^{13} arithmetic instructions]); *memory requirements* consider the amount of processor memory for a sieving process, which may be distributed among many processors, and for linear algebra operations, which would typically be on a single processor; and *storage requirements* is the amount of disk space.

Modulus Size (Bits)	**Processing Time (MIPS-Years)**	**Memory Requirements (Bytes)**		**Storage Requirements (Bytes)**
		Sieving process	*Linear Algebra*	
512	4.0×10^5	1.3×10^8	2.0×10^{10}	5.0×10^{10}
1024	3×10^{12}	3×10^{11}	1×10^{14}	2×10^{14}
2048	3×10^{21}	8×10^{15}	3×10^{18}	7×10^{18}
4096	2×10^{33}	8×10^{21}	3×10^{24}	7×10^{24}

Table D.2: Estimated Cryptographic Strength for IF Modulus Sizes

There is some variation among published estimates of running time due to the particular definition of a MIPS-Year and to the difficulty of estimating actual processor utilization. (How many arithmetic instructions a modern processor performs in a second when running an actual piece of code depends heavily not only on the clock rate, but also on the processor architecture, the amount and speeds of caches and RAM, and the particular piece of code.) Thus, the estimates given here may differ from others in the literature, although the relative order of growth remains the same. The length of the modulus should be selected so that integer factorization methods have sufficient difficulty; as mentioned, a common minimum length is 1024 bits.

2. *(Key type: RSA vs. RW.)* The key type (which varies only for the signature schemes) is not known to affect the security of the schemes with recommended and correctly implemented encoding methods. Although root extraction for an even exponent is provably as difficult as integer factorization, only certain encoding methods (e.g., PSS [BR96]) result in schemes whose security can be proven equivalent to integer factorization; the ones recommended here do not. However, certain attack strategies on signature schemes (see, e.g., [JQ99]) have been shown to yield the factorization of the modulus in the RW case but not in the RSA case. Root extraction for an odd exponent is not known to be equivalent to integer factorization. (Boneh and Venkatesan [BV98] have recently shown strong evidence that the problems are not equivalent for low-exponent RSA.) Nevertheless, integer factorization is the only known approach for forging signatures or recovering messages for the schemes with recommended encoding methods, regardless of key type.

3. *(Prime size.)* The lengths in bits of the primes p and q should be approximately half the length of the modulus n, since this maximizes the difficulty of certain special-purpose integer factorization

methods. Such methods include the Pollard rho method [Pol75], with asymptotic expected running time $O(p^{1/2})$; the Pollard $p - 1$ method [Pol74], with running time $O(p')$, where p' is the largest prime factor of $p - 1$; and the $p + 1$ method [Wil82], with running time $O(p')$, where p' is the largest prime factor of $p + 1$. The elliptic curve method (ECM) [Len87] is superior to these; its asymptotic running time is $O(\exp(2\ln p \ln \ln p)^{1/2})$. All these methods are slower than GNFS (see Note 1 above) for factoring a large IF modulus with prime factors that are approximately half the length of the modulus. These methods may be effective, however, when one of the prime factors is small.

A desired security level can also be provided when the lengths of the primes are not approximately half the length of the modulus, so long as the primes are large enough to resist the special-purpose methods just mentioned. Such a choice requires further security analysis by the implementer. For some further discussion, see [Sha95] (see also [GGO96] for a security analysis).

4. *(Prime generation.)* The primes should be generated randomly or pseudorandomly since this provides resistance to exhaustive search and other special-purpose attacks (see Annex D.6 for more on random number generation). The prime generation method should have a sufficiently low probability that a non-prime is generated, so that the probability of generating an invalid key is small. A small but nonzero probability of error (say, $< 2^{-100}$) is acceptable since it makes no difference in practice; methods with a small probability of error are generally simpler than those with zero error. Other criteria may be added to prime generation to ensure that primes meet certain properties. Acceptable prime-generation methods include incremental search from a random starting point (see Annex A.15.6 and A.15.8) with either probabilistic primality testing (probability of nonprime output of less than 2^{-100}; see Annex A.15.1-A.15.3) or primality proving (see Annex A.15.4); and recursive construction of primes with an implicit proof of primality (such as [Sha86], [Mau95], [Mih94]). See also [MOV96, Sections 4.1-4.4], ANSI X9.31[ANS98a] and ANSI X9.80 [ANS98g].

   Computing primes as a one-way function of a random seed provides a degree of auditing since it makes it difficult to select a prime with some predetermined rare property. It may not entirely prevent a user from producing primes with special properties, since a user can try many different seeds, looking for one that yields the desired property, if the property is likely enough to occur. However, it will make it difficult for a user to produce primes that are vulnerable to a special-purpose factoring method such as the $p - 1$ method, provided that the prime is large enough. (A user may wish to do this in the interest of later disavowing a signature.) The seed should be kept secret, because revealing it would reveal p and q and thus the private key. Therefore, the actual auditing must be done only by trusted parties or in situations when it is appropriate to reveal the private key. ANSI X9.31 [ANS98a] gives an auditable method of generating primes by incremental search.

5. *(Public exponent selection.)* The public exponent may have any value consistent with the modulus and key type, since the value of the public exponent is not known to affect the security of the schemes with recommended and correctly implemented encoding methods. IF schemes with non-recommended (or incorrectly implemented) encoding methods or implementations that leak a fraction of the bits of the private exponent or the primes may be vulnerable to certain attacks when the public exponent is small (see [CFP96], [Has88], and [BDF98]). Moreover, it has been shown that for very small public exponents such as $e = 3$, the RSA problem may not be equivalent to the integer factorization problem [BV98] (see note 2). However, the recommended encoding methods and appropriate protection of the private key (see Annex D.7) prevent these attacks. Typical public exponent values are $e = 2$ for RW keys and $e = 3$ or $2^{16}+1$ for RSA keys. A larger public exponent may nevertheless offer an additional line of defense, as it can mitigate concerns about implementation failures in the encoding methods or in the underlying random number generation for an IF encryption scheme.

Restricting the allowed set of public exponents system-wide, and ensuring that system components refuse to perform operations with public exponents that do not satisfy the restriction, may provide a measure of protection against protocol attacks that exploit the ability of an opponent to pick an arbitrary public exponent (see, e.g., [CH98]). However, such attacks are generally better defended against by picking appropriate protocols.

A more common use of a system-wide restriction is picking a specific public exponent system-wide in order to avoid having to store and transmit it with the public key. For example, a system using RW keys may have a system-wide policy that $e = 2$ for all keys, in which case just n, not (n, e), needs to be transmitted and stored for public keys. The system-wide e should be protected from unauthorized modification the same way any public key would be.

6. *(Private exponent selection.)* The private exponent should be derived from the public exponent or generated at random (see Annex D.6) so that with high probability it will have approximately the same size as the modulus. In particular, the private exponent should be substantially longer than one quarter of the modulus length. This makes it difficult to recover the private exponent by certain methods, such as [Wie90] and [BD99]. (Note that ANSI X9.31 requires that for a 1024-bit modulus the private exponent is at least $2^{512+128s}$ where s is an integer ≥ 0.) A desired difficulty can also be provided when the private exponent is significantly shorter than the modulus or has some other structure, such as low weight; such choices require further security analysis by the implementer.

7. *(Modulus, prime, and exponent sharing.)* A modulus should not be shared since it is possible, given two public keys with the same modulus and one of the corresponding private keys, to determine the other private key. A prime should not be shared since moduli with one prime factor in common can be factored by taking their GCD. A private exponent should not be shared since it is the only private component of the private key. Sharing of any of these quantities is unlikely to occur if primes are generated at random and the private exponent is derived from the public exponent or generated at random. A public exponent may be shared since it is public and may have any value consistent with the modulus and key type.

8. *(Private-key representation.)* The choice of private-key representation does not affect security against cryptanalytic attack, although security against certain implementation attacks may vary according to the representation. For instance, the Bellcore fault-analysis attack [BDL97] is more feasible if the private key contains the prime factors of the modulus, since a single undetected bit error in an exponentiation modulo one of the primes will compromise the private key (see Annex D.7 for more on implementation attacks).

D.5 Scheme-Specific Considerations

This section gives further information on security implications of choices for each of the three types of scheme, including primitives, key derivation function or encoding method, key derivation or encoding parameters, and authentication, validation, and cryptoperiod specifics for keys in the schemes.

D.5.1 Key Agreement Schemes

Key agreement schemes are generally used to provide assurance that nobody but the two legitimate parties involved in the protocol can compute the shared secret key. Note that a basic key agreement scheme does not provide assurances that the parties are correct about each other’s identities, or that both parties can actually compute their shared secret keys. Key confirmation (see Annex D.5.1.3) is often used to provide such additional assurances.

In addition, key agreement schemes are sometimes used to provide forward secrecy. See Annex D.5.1.7

for more on forward secrecy and how the key agreement schemes in this standard can provide it.

The application of key agreement schemes in a key establishment protocol is a particularly challenging task, as illustrated by the various attacks on key agreement protocols that have been observed over the years. Apparently slight changes, such as the ordering of messages, may well introduce unintended security weaknesses. Conversely, slight differences may yield security benefits. For instance, requiring each party to commit to its short-term key by sending a hash of the short-term key, prior to exchanging the short-term keys, can remove some potential for an opponent to manipulate a protocol. Although the recommendations in this section cover many of the general principles of key agreement, the implementer is encouraged to consult recent literature on key agreement protocols for further information.

D.5.1.1 Primitives

Secret value derivation primitive choices include the following (the particular choices vary among the three key agreement schemes):

— DLSVDP-DH, DLSVDP-DHC, DLSVDP-MQV, DLSVDP-MQVC
— ECSVDP-DH, ECSVDP-DHC, ECSVDP-MQV, ECSVDP-MQVC

The choice between DL and EC affects security to the extent that the difficulties of the underlying problem differ (see Annex D.4). Attacking the underlying problem is the best currently known method for attacking the primitives. (For ECSVDP-DH and ECSVDP-DHC, attacking the primitive is equivalent to attacking the underlying problem in the sense that if one takes exponential time, then the other takes exponential time as well [BL96].) The choice of -DH vs. –DHC or -MQV vs. -MQVC affects security with respect to key validation (see Annex D.5.1.6 below).

D.5.1.2 Key Derivation Function

The only recommended key derivation function is KDF1.

A key derivation function for the key agreement schemes in this standard (i.e., the three DL/ECKAS schemes) should produce keys that are computationally indistinguishable from randomly generated keys (see Annex D.6 for more on computational indistinguishability). KDF1 is considered to have this property, provided that the length of the key derivation parameters is the same for all keys computed from a given shared secret value.

The security of KDF1 depends on the underlying hash function; SHA-1 and RIPEMD-160 are allowed. The length of the hash function’s intermediate chaining value (for iterative constructions such as SHA-1 and RIPEMD-160) governs the security of KDF1 against certain forms of attack. Typical theoretical attacks for distinguishing KDF1 from a random source involve on the order of 2^{80} operations assuming 160-bit hash function, and these attacks do not necessarily yield any particular key.

D.5.1.3 Key Confirmation

Key confirmation is the assurance provided to each party participating in the key agreement protocol that the other party is actually capable of computing the agreed-upon key, and that it is the same for both parties. Note that key agreement does not necessarily, by itself, provide key confirmation. For example, an opponent may present a public key that is identical or related to some other party’s public key, to make it appear as though a derived key is shared with the opponent, when in fact it is shared with the other party. This is known as the *unknown key-share* attack (see [BM98], [DOW92], [MQV95], [Kal98b], [LMQ98]).

An unknown key share attack may be a concern in situations where a party assumes that another party has certain privileges as a result of a successful key agreement protocol, and the other party is not further identified in the protocol. If an opponent can make it appear as though a derived key is shared with the opponent, when in fact it is shared with the other party, then the opponent may be able to impersonate the other party in subsequent messages encrypted with the derived key (even though the opponent does not know the derived key). In typical protocols, this is a theoretical concern, since exchanged messages (e.g., funds transfers) should generally identify the parties.

In general, if an opponent can obtain a certificate for another party's public key, the opponent can trivially mount an unknown key share attack by substituting its certificate for the other party's certificate. The CA can prevent this kind of attack by checking for duplicate public keys, or by requiring a proof of possession of the corresponding private key.

For variants of DH2 where one party's ephemeral key is combined with another party's static key, an opponent can also mount an unknown key share attack by substituting powers or multiples of the combined static and ephemeral public keys, obtaining a certificate for the new static key, and substituting the new certificate. (See [MQV95].) A CA can prevent this attack by requiring a proof of possession of the private key, but cannot prevent it by checking for duplicates.

For MQV, an opponent can mount an unknown key share attack with certain substitutions, as further described in [Kal98]. A CA cannot prevent the attack either by checking for duplicates or by requiring a proof of possession of the private key. As a result, if the unknown key share attack is a concern, protocol steps such as key confirmation are essential.

In general, to avoid unknown key share attacks and other possible attacks, the parties should perform a key confirmation protocol which securely associates the names of the parties with the knowledge of the shared secret key. Such protocols usually involve computations of cryptographic functions based on the identities of the parties and the derived keys (see, e.g., [MQV95], [BJM97], [ANS98b], [ANS98e]).

D.5.1.4 Key Derivation Parameters

The purpose of key derivation parameters is to distinguish one derived key from another. As such, the parameters should have an unambiguous interpretation. If no other key confirmation is performed, they should identify the parties exchanging the key and specify the purpose of the derived key (e.g., its type and position within a sequence of derived keys of the given type).

For KDF1, the set of all possible key derivation parameters for a given shared secret value should be prefix-free. (A string P is called a *prefix* of a string S if S begins with P, i.e., $S = P \| R$ for some string R; a set of strings is *prefix-free* if it does not contain a pair of strings P, S such that P is a prefix of S.) Otherwise, due to the nature of hash functions used in KDF1, an opponent may be able to derive new shared secret keys based on the same shared secret value and new key derivation parameters, without knowing the shared secret value. If all key derivation parameters are the same length, or if they are BER or DER encodings of ASN.1 structures, they will satisfy the prefix-free requirement.

D.5.1.5 Authentication of Ownership

If it is desired to verify that a particular party has the ability to compute a given derived key, the party’s ownership of a public key from which the derived key is computed should be authenticated. The following table summarizes the typical means of associating the derived key with a particular party, in terms of which public key should be authenticated:

Scheme	Authenticated Public Key
DH1	The party's public key
DH2	Either of the party's public keys
MQV	The party's first public key

Table D.3: Summary of typical means of associating public keys with parties in a key agreement scheme

In general, as noted in D.3.1 and D.5.1.2, the party's possession of the corresponding private key should also be authenticated, by means of a key confirmation protocol or otherwise. The risk of not doing so is the same as the lack of key confirmation, as further discussed in D.5.1.3.

A situation in which it may be acceptable not to authenticate a party's possession of a private key (directly or through key confirmation) is one in which the key derivation parameters include information identifying the party. The identifying information avoids any ambiguity in the sharing of the derived key.

D.5.1.6 Validation of Domain Parameters and Keys

As discussed in D.3.3, using invalid keys or domain parameters as inputs to a primitive carries with it certain risks. Specifically, for the key agreement schemes, the risks are outlined below. Note that the risks will vary with the particular implementation:

— failure of the implementation, resulting in an error state
— reduction in size of the key space for derived keys
— compromise of information about a private key that is combined with the invalid public key in a secret value derivation primitive [LL97]

An attack in which an opponent deliberately provides an invalid public key is sometimes referred to as a *small-subgroup attack*. If an opponent's public key is not valid, the opponent may be able to use a small subgroup of the underlying group to confine the shared secret value or to obtain information about the legitimate party's private key. (If both public keys are valid, then the shared secret value ranges over the subgroup of size r generated by the generator g; otherwise, it may be outside the subgroup of size r, possibly in a subgroup of small size.)

These risks can be migrated by ensuring the validity of domain parameters and public keys. However, validation does not address certain other risks. A key may be valid and still be insecure (for example, if the random number generation for the key generation was performed improperly, or if the private key is stored insecurely or deliberately disclosed). Therefore, an implementer should consider other ways in which the key may be insecure, and then decide how to appropriately mitigate the risk. FIPS 140-1 implementation validation and random number generation are typical ways to address some of these concerns (see D.6.1 and D.7).

An alternative to validating the public key is to select a secret value derivation primitive with cofactor multiplication (i.e., -DHC or -MQVC). Provided that the associated set of domain parameters is valid and the public key is validated as an element of the underlying group (i.e., element of the finite field or point on the elliptic curve), a -DHC or -MQVC primitive will operate appropriately on invalid public keys; no further validation is necessary.

A situation in which it may be acceptable not to validate a public key is one in which the public key is authenticated (typically by the other party in a key agreement operation) and the private key with which the public key is combined in a secret value derivation primitive has a short cryptoperiod. The authentication prevents an outside opponent from substituting an invalid public key in an attempt to reduce the key space. The short cryptoperiod of the private key mitigates the potential for an adversary to benefit from compromising information about the private key.

The amount of information about the private key that the opponent can possibly compromise by using an invalid public key may be limited if the group has few elements of order less than r (e.g., if $q = 2r + 1$ in the DL case or if the cofactor k is small in the EC case), as described in [LL97]. However, an opponent may still be able to reduce the variability of the shared secret value by confining it to a small subgroup.

D.5.1.7 Cryptoperiod and Protection Lifetime

In a key agreement scheme, if an opponent learns a party's private key or keys, then the opponent may be able to recover derived keys produced from the private key or keys. The longer the cryptoperiod, the more derived keys are potentially vulnerable to recovery. The private keys used for key agreement schemes should be erased after their cryptoperiods expire, in order to limit a private key's potential vulnerability to physical compromise.

By giving certain private keys a short cryptoperiod and erasing them after they expire, it is possible to overcome the risk of recovery of derived keys due to compromise of parties' cryptographic state. This prevents a passive opponent who merely recorded past communications encrypted with the shared secret keys from decrypting them some time in the future by compromising the parties' cryptographic state. This property is called *forward secrecy*. *One-party forward secrecy* means that an opponent's knowledge of that party's cryptographic state after a key agreement operation does not enable the opponent to recover derived keys. *Two-party forward secrecy* ([Gun90]; see also [DOW92]), means that an opponent's knowledge of *both* parties' cryptographic state does not enable recovery of previously derived keys.

The following table summarizes the typical means for achieving forward secrecy with respect to one party or both parties with the key agreement schemes, in terms of which private keys should have short cryptoperiod (i.e., be short-term):

Scheme	**Short-Term Private Key (One-Party Forward Secrecy)**	**Short-Term Private Keys (Two-Party Forward Secrecy)**
DH1	Party's private key	Both parties' private keys
DH2	Either of party's private keys	Both parties' second private keys (see the note below)
MQV	Party's second private key	Both parties' second private keys

Table D.4: Summary of typical means for achieving forward secrecy in a key agreement scheme

Here, "short-term" means that a private key is generated immediately prior to a secret value derivation operation, and destroyed as soon as possible thereafter.

Key cryptoperiod and authentication are independent considerations; a public key may be authenticated even if the corresponding private key is short-term. The STS protocol ([MOV96, Section 12.6], [Dif88, p. 568], [DOW92]) is an example of this approach.

NOTE—In the DH2 scheme, it is also possible to achieve two-party forward secrecy if both parties' first private keys are short-term; however, conventionally, it is the second private keys that are short-term for two-party forward secrecy. Two-party forward secrecy is not achieved if one party's first key and the other party's second key are short-term while the other keys are long-term.

D.5.2 Signature Schemes

Signature schemes are generally used to provide authenticity of data—an assurance that only the party possessing the corresponding private key is capable of producing a signature that verifies as valid with a given public key. A commonly used theoretical definition for the security of signature schemes is given in [GMR88].

There are two types of signature schemes: signature schemes with appendix, and signature schemes with message recovery. Signature schemes with appendix require the message to be transmitted in addition to the signature; without knowing the message signed, one cannot verify the signature. Signature schemes with message recovery produce signatures that contain the message within them. The verifier does not need to know the message in order to verify the signature: if the signature is valid, the message signed is recovered during the signature verification process. While signature schemes with appendix may be used to sign messages of practically any length (subject only to the limitations of the encoding method), signature schemes with recovery are generally used for messages that are short enough. Note that no signature schemes with recovery are defined in this standard (see C.3.4 and C.3.7), and hence no security considerations are given here for such signature schemes.

D.5.2.1 Primitives

Signature and verification primitive choices include the following pairs (the particular choices vary among the three signature schemes):

— DLSP-NR and DLVP-NR, DLSP-DSA and DLVP-DSA, ECSP-NR and ECVP-NR, ECSP-DSA and ECVP-DSA, IFSP-RSA1 and IFVP-RSA1, IFSP-RSA2 and IFVP-RSA2, IFSP-RW and IFVP-RW

The choice between DL, EC, and IF affects security to the extent that the difficulties of the underlying problems differ (see Annex D.4). Although none of the primitives is known to be equivalent to the underlying problem, attacking the underlying problem is the best currently known method for attacking the primitives. Note that the DL and EC signature primitives use randomness, and thus require an appropriate random number generator for security (see Annex D.6 for more on random number generation).

D.5.2.2 Encoding Methods

The recommended encoding methods for signatures with appendix are EMSA1 (for DL/ECSSA) and EMSA2 (for IFSSA).

An encoding method for a signature scheme with appendix in this standard (i.e., DL/ECSSA or IFSSA) should have the following properties, stated informally:

— it should be difficult to find a message with a given message representative (the *one-way* property)
— it should be difficult to find two messages with the same representative (*collision resistance*)

- the encoding method should have minimal mathematical structure that could interact with the selected signature primitive (e.g., if the signature primitive is multiplicative, the encoding method should not be)
- the encoding method may identify the hash function (and other options) within the message representative in order to prevent an opponent from tricking a verifier into operating with a different hash function than the signer intended -- perhaps a broken hash function; however, this may also be accomplished by key usage restrictions (see Annex D.3.5)

EMSA1 is considered to have these properties (except for identifying the hash function) for DL/ECSSA. EMSA2 is considered to have these properties for IFSSA. (Although EMSA1 does not identify the hash function, in practice it is typically combined with a single hash function, SHA-1 – which amounts to a key usage restriction, and removes any risk of ambiguity.)

The security of EMSA1 and EMSA2 depends on the underlying hash function; SHA-1 and RIPEMD-160 are allowed. The length of the hash function output governs the security of both encoding methods. Typical attacks for finding a message with a given representative involve on the order of 2^{160} operations assuming 160-bit hash function; attacks for finding two messages with the same representative involve 2^{80} operations.

If the maximum length l of the message representative is less than the output length of the hash function, then EMSA1 will simply truncate the hash function output. Therefore, for DL and EC signature schemes, if the length of the generator order r (and, hence, the maximum length of the message representative) is less than 160 bits, the security of the encoding method will be limited by 2^l and $2^{l/2}$, rather than 2^{160} and 2^{80}, respectively. In addition, for DLSVDP-DSA and ECSVDP-DSA, if the length of r is not greater than 160, then an opponent may pick r in such a way as to cause two different message to have the same signature (see [Vau96]). This can be prevented by increasing the length of r beyond 160 bits, or by auditing domain parameter generation (see Note 3 in D.4.1.4 and Note 5 in D.4.2.4). It is customary for the length of r and the length of the hash function output to be approximately equal, so the message representative input to the primitive appears to be a random value between 0 and r-1. If the length of r is greater than the length of the hash value, then this appearance will no longer hold, but this is not known to result in any security risk.

A further discussion of desired theoretical properties for signature schemes may be found in [BR96].

D.5.2.3 Repudiation

Signature schemes have a special security concern, similar to that for handwritten signatures. Namely, a dishonest signer may be interested in using deliberately weak methods in order to produce a signature that is accepted, but later be able to claim that the signature was forged by someone else who “broke” the cryptosystem. This may allow the signer to later *repudiate* the signature. Since signers are often in some way liable for the statements they sign, this will allow a dishonest signer to avoid the liability. In fact, a dishonest signer may not be using weak methods, but merely be able to claim that the methods were weak, by claiming, for example, that the key was leaked or that the random number generator was weak.

Systems in which this is a concern should consider what conditions may be accepted as a basis for repudiation, and then ensure that none of the conditions is present before accepting a signature. For example, if repudiation is possible on the basis that the key size was too small, systems should set policies by which signatures with keys under a certain size are not accepted. If repudiation is possible on the basis that a key was not authentic or invalid, systems should ensure authenticity and validity of keys and parameters (see Annex D.3.1, D.3.3, D.5.2.4 and D.5.2.5). If repudiation is possible on the basis that the implementation was insecure, implementation validation may be appropriate (see Annex D.7). Implementations may also ensure that users are not able to copy or view their own private keys, and thus cannot deliberately leak them to other parties (see also D.3.2 for more on key-sharing).

System-wide policies may also help address this concern. For example, if signers generate their own keys, a system may choose to prohibit repudiation on the basis that a key was invalid, reasoning that honest signers should not generate invalid keys in the first place. They may also choose to prohibit any repudiation as long as public keys are securely associated with the signers, by asserting that it is the signers' responsibility to ensure that their keys are secure. In the latter case, a signer has a motivation to perform key validation after key generation, if there is any risk that the keys were not generated correctly (e.g., an error occurred).

When performing security analysis of a system that uses signatures, one needs to take into consideration not only adversaries whose goal it is to forge signatures, but also adversaries whose goal it is to repudiate them.

D.5.2.4 Authentication of Ownership

If it is desired to verify that a particular party has the ability to generate a given signature (for example, to prevent future repudiation of the signature by the party—see Annex D.5.2.3), then the party's ownership of the public key with which the signature is verified should be authenticated. In general, as noted in D.3.1, the party's possession of the corresponding private key should also be authenticated. The risk of not doing so is that an opponent may present a public key that is identical or related to some other party's public key, to make it appear as though a message was signed by the opponent, when in fact it was signed by the other party. Note, of course, that the opponent can always sign any message it knows with its own key. Thus, this risk is of concern in the following two cases:

— the signature is stored in such a way that it cannot be modified by the opponent, and the opponent is trying to show that it generated the signature
— a portion of the message is secret, and the verifier is relying on the signature as assurance that the purported signer knows the secret. (Note that the use of signature schemes for verification of knowledge of secrets is not discussed in this standard, because such verification is usually accomplished by a protocol, rather than by a scheme [protocols and schemes are discussed in Section 4]. Such use requires additional security analysis by the implementer.)

If the message on which the signature is computed includes the signer's name, an opponent cannot make it appear as though the opponent is the actual signer, because the opponent's name is different. (This is essentially how a signature-based proof of possession protocol works.)

D.5.2.5 Validation of Domain Parameters and Keys

As discussed in D.3.3, using invalid keys or domain parameters as inputs to a primitive carries with it certain risks. Specifically, for the signature schemes, the risks are outlined below. Note that the risks will vary with the particular implementation:

— failure of the implementation, resulting in an error state
— potential signature forgery
— repudiation of signatures based on the argument that the key is invalid and, hence, insecure

The risk of implementation failure can be addressed by appropriate implementation handling of error cases, or by ensuring the validity of the domain parameters and public keys as described in D.3.3.

The risk of signature forgery can be mitigated by ensuring the validity of the parameters and keys. However, validation does not address certain other risks. A key may be valid and still be insecure (for example, if the random number generation for the key generation was performed improperly, or if the private key is stored insecurely or deliberately disclosed). Therefore, an implementer should consider other ways in which the key may be insecure, and then decide how to appropriately mitigate the risk. FIPS

140-1 implementation validation and random number generation are typical ways to address some of these concerns (see D.6.1 and D.7). Public key and parameter validation will ensure that no invalid public keys and parameters are used to verify a signature.

The risk of repudiation and ways to address it are described in more detail in D.5.2.3.

D.5.2.6 Cryptoperiod and Protection Lifetime

In a signature scheme, if an opponent learns a signer's private key, the opponent can generate new signatures that can be verified with the corresponding public key. The longer the cryptoperiod of a private key in a signature scheme, the more data is available to a cryptanalyst. The longer the private key is not erased, the longer it is potentially vulnerable to physical compromise. However, if the private key is erased and its owner needs to repudiate a signature that was forged by an opponent, the private key is not available as evidence (the owner may need to use the private key to show, for example, that the private key was somehow weak and therefore forgery was possible).

If a verifier requires a secure timestamp [MOV96, Section 13.8.1] on signed messages and only accepts signatures that have timestamps prior to a certain date, then the opponent is limited to generating new signatures before that date. Hence, if secure time-stamps are used, the protection lifetime of the private key can be limited by a specific date, provided that the date is securely associated with the corresponding public key (see Annex D.3.6). Note, however, that secure timestamping is more than just a date field added by the signer. The existence of the signature on the message at the claimed date must be independently confirmed by the verifier or by a third party.

D.5.3 Encryption Schemes

Encryption schemes are generally used to provide confidentiality of data. Encryption schemes are generally used to provide confidentiality of data. A theoretical definition of "semantic security" (or, equivalently, "indistinguishability" or "polynomial security") is commonly used as the basis for the meaning of "confidentiality" (see [MOV96, Section 8.7], [GM84], [MRS88], [Yao82]). Semantic security provides that it should be computationally infeasible to recover any information about the plaintext (except its length) from the ciphertext without knowing the private key; this implies, in particular, that it should also be infeasible to find out whether two ciphertexts correspond to the same, or in some way related, plaintexts, or whether a given ciphertext is an encryption of a given plaintext.In addition, encryption schemes are sometimes used to provide integrity of data in the form of plaintext-awareness—an assurance that nobody could generate a valid ciphertext without knowing the corresponding plaintext (see [BR95]). (Data integrity in this context is different than for a signature scheme, where it assures that a message has not been modified since it was signed by an identified signer. Here, data integrity assures that a message has not been modified since it was encrypted by a possibly unidentified sender. Plaintext-awareness means that the sender "knew" the message.) In particular, plaintext-awareness implies security against an adaptively chosen ciphertext attack (an attack in which the opponent requests decryptions of specially constructed ciphertexts in order to be able to decrypt other ciphertexts). The encryption scheme in this standard is believed to provide plaintext-awareness, and, hence, security against adaptively chosen ciphertext attack.

For more on relations among notions of security and modes of attack for encryption schemes, see [BDPR98].

D.5.3.1 Primitives

There is only one pair of encryption and decryption primitives for encryption schemes in the standard:

— IFEP-RSA and IFDP-RSA

Although the primitives are not known to be equivalent to the underlying problem of factoring, attacking the underlying problem is the best currently known method for attacking the primitives.

D.5.3.2 Encoding Method

The only recommended encoding method for encryption is EME1.

An encoding method for an encryption scheme in this standard (i.e., IFES) should have the following properties, stated informally:

— representatives of different messages should be unrelated
— the encoding method, through incorporation of randomness or otherwise, should ensure that representatives of the same message produced at different times are unrelated, and that it is difficult to determine (without the private key), given a ciphertext and a plaintext, whether the ciphertext is an encryption of the plaintext
— message representatives should have some verifiable structure, so that it is difficult to produce a ciphertext that decrypts to a valid message representative (plaintext awareness)
— the encoding method should have minimal mathematical structure that could interact with the encryption primitive (e.g., the encoding method should not be multiplicative, because IFEP-RSA is)

EME1 is considered to have these properties for IFES (one motivation for using EME1 as opposed to other encoding methods is a result by Bleichenbacher [Ble98]). It also has the additional property of securely associating optional encoding parameters with a message, where the encoding parameters are not encrypted but are protected from modification; see Annex D.5.3.3.

The security of EME1 depends on the underlying hash function, mask generation function, and random number generator for generating the seed. SHA-1 and RIPEMD-160 are allowed for the hash function and MGF1 (with SHA-1 or RIPEMD-160) is allowed for the mask generation function. See Annex D.6 for recommendations on random number generation. The length of the hash function output governs the security of the encoding method.

A further discussion of desired theoretical properties may be found in [BR95].

D.5.3.3 Encoding Parameters

The purpose of encoding parameters in an encryption scheme is to associate control information with a message. The parameters should have an unambiguous interpretation. Their content depends on the implementation, and they may be omitted (i.e., the parameters may be an empty string). The parameters are not encrypted by the encryption scheme, but are securely associated with the ciphertext and protected from modification. Whether they have been modified or not is verified during the decryption process. For information on the encoding parameters, see [JM96].

For EME1, the length of the encoding parameters can vary from one encryption operation to another.

D.5.3.4 Authentication of Ownership

If it is desired to verify that a particular party has the ability to decrypt a given ciphertext, the party's ownership of the public key with which the ciphertext is produced should be authenticated.

D.5.3.5 Validation of Domain Parameters and Keys

As discussed in D.3.3, using invalid keys as inputs to a primitive carries with it certain risks. Specifically, for the encryption schemes, the risks are outlined below. Note that the risks will vary with the particular implementation:

— failure of the implementation, resulting in an error state
— loss of confidentiality of the data

The risk of implementation failure can be addressed by appropriate implementation handling of error cases, or by ensuring the validity of the domain parameters and public keys as described in D.3.3.

The risk of loss of confidentiality can be mitigated by ensuring the validity of the parameters and keys. However, validation does not address certain other risks. A key may be valid and still be insecure (for example, if the random number generation for the key generation was performed improperly, or if the private key is stored insecurely or deliberately disclosed). Even if the key is secure, the recipient may (because of insecure implementation or deliberately) leak the content of the message. Therefore, an implementer should consider other ways in which the key or the message content may be insecure, and then decide how to appropriately mitigate the risk. FIPS 140-1 implementation validation and random number generation are typical ways to address some of these concerns (see D.6.1 and D.7). Public key validation will ensure that no messages are encrypted with invalid keys. Indeed, public key validation is one of the precautions a message sender may apply directly, whereas the other countermeasures mentioned here require the sender to rely on representations by another party, e.g., that the implementation is secure.

D.5.3.6 Cryptoperiod and Protection Lifetime

In an encryption scheme, if an opponent learns a recipient's private key, then the opponent can recover all messages encrypted with the corresponding public key. The longer the cryptoperiod of a public key in an encryption scheme, the more messages are potentially vulnerable to recovery. The longer the private key is not erased, the longer it is potentially vulnerable to physical compromise. However, once the private key is erased, no data encrypted with the corresponding public key can be decrypted.

The protection lifetime in an encryption scheme should be determined by how long the data encrypted with the public key remains sensitive.

D.6 Random Number Generation

As indicated throughout this standard, and this Annex in particular, generation of random numbers (or, more generally, random bit strings) is a tool commonly used in cryptography. Proper generation of random bit strings for cryptographic purposes is essential to security, particularly because secrets, such as private keys, are commonly derived from such strings. A failure of a random number generator (resulting in, for example, generation of predictable or repeating keys) is likely to make a system extremely vulnerable to an opponent. As opposed to random number generation in some other settings, where security is not a concern, cryptographic random number generation cannot generally be accomplished by simply invoking the software subroutine "random" in one's favorite software library.

This section describes some of the security concerns related to generating random bit strings. A more detailed exposition is given in [MOV96, Chapter 5]; the reader desiring to learn more is referred to that chapter. ISO and ANSI have recently started studying random number generation (the ANSI X9F1 working group has a work item numbered X9.82; in the ISO, ISO/IEC JTC1 SC 27 WG 2 is studying the issue); the reader may find the future results of this work useful.

Random strings in cryptography are commonly generated by the following method: collect enough data from a random source to get a seed for a pseudo-random bit generator, and then use the pseudo-random bit generator to get the string of the desired length. The two sections below address both steps of this two-step process.

D.6.1 Random Seed

An opponent can be assumed to have full knowledge of the pseudo-random bit generator (see Annex D.2), except for its seed input. The random seed is thus the only component of the random number generation process that the opponent may not know. Therefore, implementers should ensure that the seed is collected from a source that has sufficient variability and cannot be accessed, guessed, or influenced in a predictable way by the opponent. In short, the seed should be unpredictable. If the result of the random number generation process is to be kept secret, the seed should be kept secret as well.

Ideally, each bit produced by the random source should be evenly distributed and independent of all the other bits. In such a case, there are 2^n equally probable random seeds of length n, and the opponent has no advantage in guessing the seed. However, in reality, this is often not the case. As random sources usually have biased and correlated bits, opponents often have a better chance of guessing the seed of length n than 1 in 2^n. When evaluating a random source, it is essential to establish that an opponent with appropriately large computational resources and full knowledge of the nature of the random source has a negligible probability of guessing a seed output by the random source. Note that the opponent should be considered to have as many attempts at guessing as the opponent's potential computational resources would allow.

Informally, variability of a random source is measured in bits of randomness (also called "bits of variability") per bits of output. For example, if a random source is said to have variability of "one bit per byte," this generally means that there are about 2^k different strings of more-or-less equal probability of length $8k$ bits. This, in order to get about 160 bits of randomness from such a source, one needs 1280 bits of data.

If the random bits are biased or correlated, it is common to run them through a cryptographic hash function in order to "distill" the randomness. This method relies on the assumption that the cryptographic hash function will produce a distribution on the outputs that is reasonably close to uniform, because the hash function is not correlated to the biases of the random source. In the example above, the 1280 bits of data could be given as an input to, for example, SHA-1 to come up with a 160-bit output that has (presumably) about 160 bits of randomness. Note that the hash function output should not be longer that the number of bits of randomness believed to be provided by the data.

If the random seed needs to be kept secret, it is essential that the source of the random data be protected from eavesdropping. For example, if one is using the timing of a user's keystrokes as the source of the random data, it is important that the keystrokes cannot be observed by the opponent through, for example, a hidden camera installed in the room.

Whether or not the seed needs to be kept secret, it is also essential that the source of random data be protected from manipulation by the opponent. For example, if system loads or network statistics are used for random data, the opponent having access to the system or the network may be able to manipulate them in order to reduce their variability, even if not able to observe them directly.

A common precaution is to combine many available sources of randomness and to use a hash function, as described above, in order to "mix" them. If, for example, one collects enough data for 160 bits of variability each from three different sources, then even if two of the sources fail, the remaining source will provide about 160 bits of variability in the hash function output. It is generally safer to collect more data rather than less data from each of the sources, even if only a short seed is needed.

Some sources of randomness that may be used include:

- photon polarization detected at 45^o out of phase
- elapsed time between emissions of particles from a radioactive source
- quantum effects in a semiconductor, such as a noisy diode or a noisy resistor
- the frequency fluctuations of free-running oscillators
- the fluctuations in the amount a metal insulator semiconductor capacitor is charged during a fixed period of time
- fluctuations in read times caused by air turbulence within a sealed disk drive dedicated to this task [DIF94]
- the difference between the output of two microphones in two different points in a room (provided their amplification is normalized to minimize the difference signal)
- the noise within the signal of a microphone or video-camera in a room (tends to provide less variability and is easier to predict and observe than the previous method)
- the electronic noise of an A-D converter with an unplugged microphone (the variability depends heavily on the particular converter and environment)
- variation in mouse movements or keystrokes and their timing (for example, the user may be asked to sign his or her name with the mouse; note that accessibility of correct high-resolution timing data for mouse movements and key strokes varies among different systems)
- the system clock (tends to provide very little variability and can be assumed to be known to the opponent except for the last few digits, but may be used in combination with other sources; note also that the resolution of the system clock that is available to the application varies among different systems)
- operating system statistics that cannot be observed by a potential opponent (tend to provide little variability)

When using any of these sources, it is essential not to overestimate the amount of variability that is inaccessible to the opponent. For example, if the user is asked to sign his or her name, the signature may be known to the opponent, and only the deviations from the usual signature should be considered to provide variability. Or if the room sounds in a particular frequency range are used are used as a source of randomness, the opponent may be able to inject noises in that frequency range in order to bias the result. A more detailed analysis of some of these, as well as other, sources is provided in [ECS94].

The quality of a particular source of randomness for generating random seeds should be tested by one or more statistical tests. While statistical tests do not provide a guarantee that the generator is “good,” they help detect some weaknesses the generator may have. A number of tests are described in [MOV96, Section 5.4 “Statistical tests”] and FIPS 140-1 [FIP94a, Section 4.11.1] ([FIP94a] also describes a continuous random number generator test in Section 4.11.2); a particular test that detects many weaknesses is given in [Mau91] (see also [CN98]). Note that while statistical tests may detect certain weaknesses, they do not guarantee unpredictability of the random bit source.

It is important not to confuse random-looking but public sources of information for secret sources of randomness. For example, USENET feeds, TV broadcasts, or tables of pseudo-random numbers, or digits of π should be assumed to be accessible to the opponent and should not be relied upon as sources of random seeds.

D.6.2 Pseudo-Random Bit Generation

Pseudo-random bit generators are deterministic algorithms that, given a seed as input, produce a pseudo-random bit string of a desired length (deterministic here means that they do not use any randomness other than that given by the input seed). Because they are algorithms and should be assumed publicly known, the unpredictability of their output relies heavily on the unpredictability of the input seed.

A pseudo-random bit generator is generally used to output more bits than the seed provides. Therefore, its outputs are not random, even if the seeds are. In fact, the distribution of the outputs cannot possibly be close to uniform, because number of possible outputs is limited by the number of possible seeds. For example, if a pseudo-random bit generator uses a 160-bit seed as input and returns a 300-bit output, only 2^{160} out of the possible 2^{300} (or one in 2^{140}) 300-bit strings can be potentially output by the generator.

Thus, the appropriate security condition to impose on a pseudo-random bit generator is not that it behave like a true random source. Rather, the condition is that it be indistinguishable from a true random source by any statistical test that can be computed with resources available to an opponent. More precisely, no computation that can be performed with a feasible amount of computational resources should be able to distinguish a truly random string from the output of pseudo-random bit generator with probability significantly better than 1/2. (It is, of course, easy to distinguish with probability exactly 1/2 by simply randomly guessing which one is the random string and which one is the output of the generator.) This condition is sometimes called *indistinguishability* ([Yao82]).

A condition that is equivalent to indistinguishability is the following: given all but one bits of the output of the pseudo-random bit generator, it is infeasible to predict what the remaining bit is with probability significantly better than 1/2. In particular, it is infeasible to predict what the next bit of the output will be, even when given all the previous bits. This condition is sometimes called *unpredictability* ([BM84]).

It is essential that pseudo-random bit generators used in cryptography can be reasonably believed to satisfy the above conditions. Some such generators are defined in [FIP94b], [ANS85], [MS91], [BBS86]; see [MOV96, Sections 5.3 and 5.5] for their descriptions. The formal basis for the belief of security varies according to the generator. The security of some ([FIP94b], [ANS85]) is based on heuristic assumptions, such as the unpredictability of outputs of DES or SHA-1 when a key or other input is unknown. The security of others ([MS91], [BBS86]) is based on reduction from a hard problem, such as integer factorization. Some attacks on some of the above generators and suggestions for improving pseudo-random generator design are contained in [KSW98].

The above two conditions imply that it is infeasible to guess the seed given just the output of the generator. This is why pseudo-random generators may provide assurance that a set of parameters or keys was generated from a seed (rather than the seed reconstructed after the generation), as is further detailed in D.3.2 and D.4.

The above two conditions also imply that the output of the pseudo-random generator, if long enough, is very unlikely to repeat in any reasonable time. Thus, keys, if generated properly, are very unlikely to be repeated or accidentally shared among users.

Statistical tests described in the previous section for testing the quality of the seed may also be used to test the quality of the pseudo-random bit generator. However, some generators commonly used outside of cryptography (such as the linear congruential generator) may pass many of the statistical tests while being entirely predictable and therefore insecure for cryptographic purposes.

D.7 Implementation Considerations

This standard (and this Annex in particular) focuses on the cryptographic methods and algorithms to implement them. Many engineering-related considerations are just as important in a secure implementation. This section gives a sampling of the problems an implementer needs to consider. It is not meant to be comprehensive, as solutions to these problems are outside the scope of this standard. For more on secure implementation, see [FIP94a] and [MV97, Book 2, Appendix P].

As with other security considerations, sources of the potential threats and their capabilities need to be examined. For example, opponents may be able to induce errors in systems by operating them under unexpected conditions, whether physical (such as change in temperature, power source characteristics, electromagnetic radiation – e.g. TEMPEST issues [AK98]) or computational (such as ill-formed protocol messages or multiple requests overloading a server). Such errors may leak sensitive information, see, e.g. the fault-analysis attack of [BDL97], or the story of the "internet worm" in [HM95], [Keh95] or at `http://www1.minn.net/~darbyt/worm/worm.html`. Implementations, therefore, should make sure that they handle unexpected or erroneous inputs and environments appropriately at all levels where an adversary may be present.

Implementations should also consider what information they may be giving to an adversary by indirect means. For example, by measuring the time a computation takes, an adversary may be able recover some information about a private key (see [Koc96] and [DKL98]). By analyzing the error messages sent by an implementation, an adversary may also be able to recover information (see, e.g., [Ble98]). Therefore, an implementation of any operation that uses secrets (such as key agreement, signature production or decryption) should limit the information it provides in case of error. Even partial information about a secret may lead to recovery of the entire secret (see, e.g., [BDF98]).

Due consideration should also be given to protecting the secrets. They should be stored securely in such a way that unauthorized copying of even a portion of a secret is prevented. All the copies of the secrets should be accounted for. For example, care should be taken that, in a system with paging, a copy of a page containing the secret is not made on disk by the operating system without the knowledge of the implementation. Implementations should also be aware of the possibility of eavesdropping by, for example, the use of the electromagnetic radiation emanating from a video monitor (see [AK98]).

The authenticity and validity of the machinery (whether hardware or software) performing the cryptographic operations needs to be assured. Otherwise, an adversary may be able to plant a so-called "Trojan horse" within the implementation by substituting pieces of code, parameters, keys or root certificates that are imbedded in the implementation. Protecting the system itself from unauthorized modification is as important as protecting cryptographic parameters and keys. Implementation validation is addressed in more detail in [FIP94a].