Federal Office
for Information Security

BSI – Technical Guideline

Designation:	**Cryptographic Mechanisms: Recommendations and Key Lengths**
Abbreviation:	BSI TR-02102-1
Version:	2024-01
As of:	February 2, 2024

Version	Date	Changes
2019-01	22.02.2019	Addition of CCM mode among the recommended modes of operation. Addition of PKCS1.5 paddings among legacy mechanisms.
2020-01	24.03.2020	Recommendation of FrodoKEM and Classic McEliece with appropriate security parameters for PQC applications in conjunction with a previously recommended asymmetric mechanism. Recommendation of Argon2id for password-based key derivation. Transitional extension of the conformance of RSA keys with a key length of 2000 bits or more to the end of 2023.
2021-01	08.03.2021	Revision of the chapter on random generators, especially with regard to the use of DRG.3 and NTG.1 random generators. PTG.2 random generators are no longer recommended for general use. Addition of standardised versions of hash-based signature procedures.
2022-01	28.01.2022	Fundamental editorial revision of the entire text, minor adjustments to the layout. Updates in the areas of side-channel analysis, QKD and seed generation for random number generators.
2023-01	09.01.2023	Increase of the security level to 120 bits, updates in the area of PQC.
2024-01	02.02.2024	Fundamental restructuring in the context of quantum-safe cryptography, discontinuation of DSA recommendation from 2029, inclusion of the MLS protocol.

Federal Office for Information Security
P.O.B. 20 03 63, 53133 Bonn, Germany
Email: tr02102@bsi.bund.de
Internet: https://www.bsi.bund.de

Contents

List of Tables

Notations and Glossary

$\gcd$ The greatest common divisor $\gcd(a, b)$ is that natural number with the property that it divides both a and b and that any other natural number also dividing the numbers a and b is already a divisor of $\gcd(a, b)$.

$\mathbb{F}_n$ Field with n elements, also called Galois field GF(n).

$\mathbb{Z}_n$ Ring of residue classes modulo n in $\mathbb{Z}$, also called $\mathbb{Z}/n\mathbb{Z}$.

lcd The lowest or least common multiple $\mathrm{lcm}(a, b)$ of two integers $a, b \in \mathbb{Z}$ is the smallest positive integer that is both a multiple of a and a multiple of b.

φ Euler's phi function $\varphi\colon \mathbb{Z} \to \mathbb{Z}$, also known as Euler's totient function, is defined as $\varphi(n) \coloneqq \mathrm{Card}(\{a \in \mathbb{N}\colon 1 \leq a \leq n,\ \gcd(a, n) = 1\}) = \mathsf{Card}(\mathbb{Z}_n^*)$.

R^* Unit group of the commutative ring R.

Card Number of elements Card(M) of a finite set M.

Ceiling function Ceiling function $\lceil\cdot\rceil\colon \mathbb{R} \to \mathbb{Z}$, defined as $\lceil x\rceil \coloneqq \min\{z \in \mathbb{Z}\colon z \geq x\}$.

Floor function Floor function $\lfloor\cdot\rfloor\colon \mathbb{R} \to \mathbb{Z}$, defined as $\lfloor x\rfloor \coloneqq \max\{z \in \mathbb{Z}\colon z \leq x\}$.

A

AES Advanced encryption standard, block cipher standardised by NIST in FIPS 197 [84] with a block size of 128 bits. According to the length of the keys a distinction is made between AES-128, AES-192 and AES-256. Apart from related-key attacks against AES-192 and AES-256, there are no known attacks against AES that provide a significant advantage over generic attacks on block ciphers.

Asymmetric cryptography Generic term for cryptographic mechanisms in which the execution of some cryptographic operations (such as the encryption of a message or the verification of a signature) can be performed by parties that do not know the secret data.

Authenticated encryption scheme Encryption schemes that ensure not only the confidentiality but also the integrity of the data being encrypted.

Authentication Objective of secure identification of a person or information-processing system. In the context of the present Technical Guideline, this involves persons or systems that are the source or destination of a communication connection and authentication is realised by making use of a cryptographic secret.

Authentication tag Cryptographic checksum on data that is designed to reveal both accidental errors and the intentional modification of the data.

Authenticity Property of veracity, verifiability and reliability of a person, a system or data.

B

Backward secrecy (of cryptographic protocols) Also known as future secrecy or post-compromise security property of a cryptographic protocol that ensures that encrypted messages remain secret even if a key has been compromised in the past.

Birthday problem The birthday problem (also birthday paradox) is the phenomenon that intuitive estimation of certain probabilities is often incorrect. For example, the probability that among 23 people at least two of them have their birthdays on the same day in the year is over 50%, which most people misjudge by a power of ten. In the context of cryptography, this effect plays a role in cryptographic hash functions, among other things, which are supposed to calculate a unique hash value from an input. It is much easier to find two random inputs that have the same hash value than to find another input that has the same hash value as a given one (see also collision attack).

Block cipher Key-dependent, efficiently computable, invertible mapping that maps plaintexts of a fixed given bit length n to ciphertexts of the same length. Without knowledge of the key, it should be practically infeasible to distinguish the output of the block cipher from the output of a random bijective mapping.

Brute-force attack Also called exhaustion method (exhaustion for short); attack method based on an automated, often systematic trial and error of all possibilities, for example to determine secret keys or passwords. If sufficiently long keys are used, brute-force attacks on modern encryption algorithms are practically impossible, as the required computational effort (and thus the time and/or costs) would be too high. Since the performance of modern hardware is continuously increasing and the time required to try out all keys of a certain length is reduced as a result, the minimum key length must be chosen sufficiently large and increased regularly in order to prevent attacks by exhaustion.

C

Challenge-response-authentication Protocol for authenticating to a counterpart on the basis of knowledge. In this process, a verifier poses a challenge which the proving party must solve (response) in order to prove that he knows a certain piece of information without revealing this information itself.

Chinese Remainder Theorem, CRT Theorem concerning the existence and uniqueness of solutions of congruence systems. Let $n_1, \ldots, n_k \in \mathbb{N}$ be pairwise coprime natural numbers, let $n \coloneqq n_1 \cdot \ldots \cdot n_k$ be their product and let $a_1, \ldots, a_k \in \mathbb{Z}$ be arbitrary. Then the system of congruences

$$\begin{aligned} x &= a_1 \bmod n_1, \\ &\vdots \\ x &= a_k \bmod n_k \end{aligned}$$

has a unique solution $x \in \{0, \ldots, n-1\}$.

Chosen-ciphertext attack Cryptographic attack in which the attacker can can gather information by obtaining the plaintexts of chosen ciphertexts. The attacker's aim is usually to decipher a given ciphertext that does not belong to any of these plaintext-ciphertext compromises. Depending on whether the attacker knows this ciphertext before or after the end of the attack, a distinction is made between adaptive and non-adaptive chosen-ciphertext attacks.

Chosen-plaintext attack Cryptographic attack in which the attacker can gather information by obtaining the ciphertexts of chosen plaintexts.

Cipher block chaining mode (CBC-mode) Mode of operation of a block cipher in which a plaintext block is XORed with the ciphertext block generated in the previous step before encryption. For secure use, only unpredictable initialisation vectors such as a random number must be used.

Collision attack Attack on a cryptolographic hash function with the aim of finding two different input values mapped to an identical hash value. In contrast to preimage attacks, both input values (and thus also the hash value) can be chosen arbitrarily.

Collision resistance A function $h\colon M \to N$ is called collision resistant if it is practically impossible to find $x, y \in M, x \neq y$ with $h(x) = h(y)$.

Confidentiality Objective of binding read access to an information to the right of access. In the cryptographic context, this means that access to the content of a message should only be possible for the holders of a secret cryptographic key.

Counter mode (CTR-mode) Mode of operation in which block ciphers can be operated to generate a stream cipher from them. Here, a nonce is encrypted and XORed with the plaintext. The special feature of the counter mode compared to other modes of operation is the fact that the initialisation vector consists of anonce to be newly chosen for each ciphertext block, combined with a counter that is incremented with each further block. The combination can be made, for example, by concatenation, addition or XOR.

Counter with cipher block chaining mode (counter with CBC-MAC, CCM-mode) Mode of operation of a block cipher that combines the counter mode for encryption with the CBC-MAC mode for integrity, thus turning a block cipher into an authenticated encryption algorithm that is capable of guaranteeing both confidentiality and integrity. With CCM, it must be ensured that an initialisation vector is not used twice with the same key, since CCM is derived from the counter mode and the latter represents a stream cipher.

Cryptographic agility, crypto-agility A cryptosystem is considered crypto-agile if its components, for example cryptographic algorithms, key lengths, key generation schemes or technical implementation, can be replaced by other components without having to make significant changes to the rest of the overall system.

D

Data authentication Protection of the integrity of a message by means of cryptographic mechanisms.

Dictionary attack Attack method to determine an unknown password (or user name) by systematically trying out a password list (also called wordlist or dictionary). The success of such attacks is based on the fact that user-chosen passwords are often easy to guess in practice, for example if they consist of regular or only slightly modified dictionary entries or are used in a similar form in various places, so that password lists from previous security incidents lead to a successful attack.

Diffie-Hellman-problem (DH) Problem of calculating g^{ab} given g, g^a, g^b in a cyclic group G generated by $g \in G$. The difficulty of this problem depends on the representation of the group. The DH problem is easily solvable by adversaries who are able to calculate discrete logarithms in G.

Discrete-logarithm-problem (DL) Problem of calculating d given g^d in a cyclic group G generated by $g \in G$. The difficulty of this problem depends on the representation of the group.

Disk encryption Complete encryption of a data carrier with the objective that no confidential information can be read from the encrypted system, at least when it is switched off.

DLIES Discrete logarithm integrated encryption scheme, hybrid authenticated encryption scheme based on DH in $\mathbb{F}_p^*$.

E

ECIES Elliptic curve integrated encryption scheme, hybrid authenticated encryption scheme based on DH in elliptic curves.

EME-OAEP Encoding Method for Encryption-Optimal Asymmetric Encryption Padding, padding scheme for RSA, see also OAEP.

Ephemeral key A cryptographic key is called ephemeral if it is generated for each execution of a cryptographic protocol (for example key agreement, signature generation). Depending on the application, further requirements may be imposed on the respective key type, among them uniqueness per message or session.

F

Factorization problem Number theoretic problem in which a composite number is to be decomposed into the product of its prime factors or, more generally, a non-trivial divisor is to be determined.

Fault attack Attack on a cryptographic system in which the attacker uses or actively causes an incorrect execution of a cryptographic operation.

Forward secrecy (of cryptographic protocols) Security property of a cryptographic protocol that states that the disclosure of long-term cryptographic secrets does not enable an adversary to compromise previous sessions of the protocol [29]. It must be noted that for any protocol, forward secrecy can only be reached if a random number generator that guarantees at least Enhanced Backward Secrecy according to [42] was used within the protocol for the generation of the ephemeral keys. If *future* sessions that have not been manipulated by an adversary are also to remain protected in the case that all long-term secrets are compromised, a random number generator that additionally provides Enhanced Forward Secrecy [42] must be used when generating the ephemeral keys.

Forward secrecy (of deterministic random number generators) Security property of a deterministic random number generator that states that future output values of the random number generator cannot be predicted with more than negligible advantage by adversaries who only know previous output values of the random number generator, but not its internal state, and whose computational power is below a limit given by the security level of the deterministic random number generator [42].

G

GCM Galois counter mode, mode of operation for block ciphers, which constructs an authenticated encryption scheme on the basis of a block cipher and supports authentication of non-encrypted data.

GMAC Message authentication code resulting from a use of GCM without data to be encrypted.

H

Hash function Function $h\colon M \to N$ that is efficiently computable and for which M is significantly larger than N. The output of a hash function is called hash value, message digest or simply hash. If h is both collision resistant and resistant to the calculation of first and second preimages, then h is called a *cryptographic* hash function. In the present Technical Guideline, the term *hash function* refers to a cryptographic hash function.

Hybrid encryption Combination of symmetric and asymmetric encryption that combines the advantages (efficiency respective convenient key exchange) of both techniques. Thereby, the sender selects a random symmetric key, the so-called session key, and uses it to symmetrically encrypt the data to be protected. He then encrypts the session asymmetrically with the recipient's public key and sends it together with the encrypted message. The recipient first decrypts the session key with his private key and therewith subsequently the actual message.

I

Information-theoretic security A cryptographic mechanism is called *information-theoretically secure* if any adversary fails in an attempt to break the mechanism due to *lack of information*. In this case, the security objective confidentiality will be achieved irrespective of the computing power available to the adversary, as long as the assumptions about the system information accessible to the adversary are correct. Information-theoretically secure mechanisms exist in many areas of cryptography, for example for the encryption of data (One Time Pad), for the authentication of data (Wegman-Carter MAC), or in the area of secret sharing (Shamir's secret sharing algorithm, see also Chapter 7). Usually there are no security guarantees in mechanisms of this kind if the prerequisites for the operation of the mechanism are not exactly met.

Initialisation vector (IV) Input to a cryptographic primitive used to establish an initial state. Usually, initialisation vectors must be (pseudo-)random, but in some applications it may be sufficient if they are unpredictable or do not repeat.

Instance authentication Proof of the possession of a secret by a user or an information processing system to another party.

Integrity Objective of binding the write access to an information to the right to modify the information. In the cryptographic context, this means that a message can only be changed unnoticed using a certain secret cryptographic key.

K

Key derivation function Cryptographic function that generates one or more other keys from one or several secret input value(s), such as a master key, password or passphrase. Key-dependent cryptographic hash functions are commonly used as key derivation functions.

Key encapsulation mechanism, KEM Cryptographic technique with which a session key, which is usually intended for use with a symmetric encryption mechanism, is transmitted using an asymmetric encryption mechanism, see also hybrid encryption.

Key length For symmetric cryptographic mechanisms, the key length, also known as key size, is the bit length of the secret key. For RSA (signature and encryption algorithms), the bit length of the RSA modulus n is referred to as key length. For schemes based on the Diffie-Hellman problem or discrete logarithms in $\mathbb{F}_p^*$ (DLIES, DH key exchange, DSA), the key length is defined as the bit length of p. For schemes based on the Diffie-Hellman problem or discrete logarithms in an elliptic curve C over the finite field $\mathbb{F}_n$ (ECIES, ECDH, ECDSA and variants), the key length is the bit length of n.

Key stretching Cryptographic key derivation technique designed to make a weak key, usually a password, more secure by ensuring that more resources (time, memory) are required for brute-force attacks. It must be impossible to calculate the strengthened key from the initial key with less effort.

L

Learning with errors, LWE Mathematical problem consisting of solving a linear system of equation containing errors.

M

MAC Message authentication code, a key-dependent cryptographic tag. Without knowledge of the key, it should be practically infeasible for an attacker to distinguish the MACs of non-repeating messages from random data. In this case, no adversary can successfully forge tags with a probability considerably above 2^{-t}, where t denotes the length of the authentication tags. Specifications for the length of t depend highly on the given application.

Man-in-the-middle-attack Type of attack in which an attacker inserts himself unnoticed either physically or – nowadays mostly – logically between two or more communication partners in order to, for example, read or manipulate information. The attacker thus enters „in the middle“ of the communication by pretending to be the receiver to the sender and the sender to the receiver.

Min-entropy The min-entropy of a discrete random variable X is defined as $-\log_2(p)$, where p denotes the probability of the most likely value for X.

Mode of operation of a block cipher A mode of operation is a construction to describe how messages longer than the block size of the block cipher are encrypted by the cipher. Only the combination of block cipher and mode of operation allows messages longer than the block length to be encrypted. Usually, the message is divided into several blocks and brought to a suitable length by so-called padding. An initialisation vector can additionally randomise the scheme of the key.

N

Nonce A (cryptographic) nonce is an arbitrary number to be used only once in a cryptographic communication. It is often a random or pseudo-random number issued in an authentication protocol to ensure that old communications cannot be reused in replay attacks. Nonces are also often encountered as initialization vectors and in cryptographic hash functions.

O

OAEP Optimal Asymmetric Encryption Padding, cryptographic padding scheme often used together with RSA encryption. The OAEP is a special form of a Feistel network with which, in the random oracle model, an encryption method that is semantically secure under chosen plaintext attacks can be constructed from any trapdoor function. If used with RSA as trapdoor function, the resulting method is also proved secure against chosen ciphertext attacks. In general, an OAEP achieves the following two goals: It adds an element of randomness which can be used to convert a deterministic encryption scheme into a probabilistic scheme, and it prevents partial decryption of ciphertexts (or other information leakage) by ensuring that an adversary cannot recover any portion of the plaintext without being able to invert the trapdoor one-way permutation.

One-way function Mathematical function that is easy to calculate but difficult to invert. Here, „easy“ and „hard“ are to be understood in the sense of computational complexity theory, specifically the theory of polynomial time problems. In a broader sense, functions are also referred to in this way for which no inversion is yet known that can be computed practically in a reasonable amount of time. A one-way permutation is a one-way function that is also a permutation, that is, a bijective one-way function. Trapdoor one-way functions, also called trapdoor functions, as well as trapdoor permutations represent a special type of one-way functions. They can only be inverted efficiently if some additional information is known. Trapdoor functions are used in asymmetric encryption methods such as RSA.

P

Padding Term for filling messages with *padding data* before encrypting them. Padding is mainly used to bring given data into the format specified by an algorithm or protocol, to randomise the result (for example, the ciphertext or digital signature) of a cryptographic mechanism, or to hide the beginning and end of the relevant data of a transmitted ciphertext.

Partition encryption Partition encryption refers to the complete encryption of a partition of a data medium. The mechanisms used are similar to those used for hard disk encryption.

Pepper Secret string chosen by a server to be appended to a password before calculating a hash value to further complicate dictionary and brute force attacks, also referred to as *secret salt* by NIST. The pepper is not stored in the same database as the hash value, but is stored in a different and as secure as possible place.

Personal identification number (PIN) In the context of this Technical Guideline, a PIN is understood to be a password consisting only of the digits 0-9.

Post-Quantum Cryptography (PQC) Cryptographic algorithms implemented on classical hardware that are thought to be secure even against attacks by a quantum computer.

Preimage attack Attack on a cryptographic hash function with the aim of finding a preimage for a given hash value of an unknown input value (first-preimage attack) or to find another preimage for a given input value that provides the same hash value (second-preimage attack).

Preimage resistance A function $h\colon M \to N$ is called preimage resistant if it is practically impossible to find an $x \in M$ with $h(x) = y$ for a given $y \in N$. It is called resistant to calculation of *second* preimages, if for given x, y with $h(x) = y$ it is practically impossible to compute an $x' \neq x$ with $h(x') = y$.

Pseudo Random Function, PRF Family of efficiently computable functions that are practically indistinguishable from a random oracle.

Public-key cryptography See Asymmetric cryptography.

Public key infrastructure System, that can create, distribute, store, verify and revoke digital certificates and is generally used for the management of public keys in the context of asymmetric cryptographic mechanisms.

Q

Quantum Key Distribution (QKD) Protocols that make use of quantum mechanical effects for secure key exchange.

R

Rainbow table Data structure that allows a fast, memory-efficient search for the original input (usually a password) of a given hash value. The search via a table is considerably faster than with the brute-force method, but the memory requirement is significantly higher (time-memory tradeoff).

Random oracle Theoretical black box that responds to every unique query with a (truly) random response chosen uniformly from its output domain. If a query is repeated, it responds the same way every time that query is submitted. Random oracles are typically used when cryptgraphic proofs cannot be carried out using weaker assumptions on the cryptographic hash function, as due to their construction they fulfill the classical properties of a cryptographic hash function (strong collision resistance and resistance to the calculation of first and second preimages) in a perfect way. A system that is proven secure when every hash function is replaced by a random oracle or a security security proof that uses a random oracle is said to be secure in the random oracle model, as opposed to secure in the standard model of cryptography (the standard model is the model of computation in which the adversary is only limited by the amount of time and computational power available).

Random oracle model See Random oracle.

Related-key attack Attack on a cryptographic mechanism in which an attacker can query encryptions and possibly decryptions not only under the actually used key K, but also under a number of other keys not known to the attacker, which are related to K in a way known to the attacker. This model is very advantageous for the attacker, yet there are situations in which related-key attacks can be practically relevant, for example in the context of constructing a cryptographic hash function based on a block cipher.

RSA Asymmetric cryptographic algorithm (named after its inventors Ronald Rivest, Adi Shamir and Leonard Adleman) that can be used for encryption and digital signatures and is based on the difficulty of the factorisation problem.

S

Salt Randomly chosen string appended to a given plaintext before it is further processed (for example, before input to a hash function) to increase the entropy of the input. Salts are often used for storing and transmitting passwords to make rainbow tables more difficult to use.

Secret sharing A mechanism for distributing secret data (for example a cryptographic key) to several parties or storage media. The original secret can only be reconstructed by evaluating several shared secrets. For example, a secret-sharing scheme may require that at least k of a total of n shared secrets must be known in order to reconstruct the cryptographic key to be protected.

Security Level (of Cryptographic Mechanisms) A cryptographic mechanism achieves a security level of n bits if there are costs associated with each attack against the mechanism that breaks the mechanism's security objective with a high probability of success, equivalent to 2^n calculations of the encryption function of an efficient block cipher (for example, AES).

Seed Start value with which a random number generator is initialised in order to generate a sequence of random numbers or pseudo-random numbers. If the same seed is used in deterministic random number generators, the same sequence of pseudo-random numbers is output.

Shannon entropy The Shannon entropy of a discrete random variable X is defined as $-\sum_{x \in W} p_x \log_2(p_x)$, where W is the range of values of X and p_x is the probability of X taking the value $x \in W$, that is $p_x = \mathbb{P}(X = x)$.

Side-channel attack Attack on a cryptographic system that exploits the results of physical measurements on the system (for example, energy consumption, electromagnetic emanation, runtime of an operation) to gain insight into sensitive data. Side-channel attacks are of very high relevance for the practical security of information-processing systems.

Symmetric cryptography Generic term for cryptographic mechanisms in which all parties involved must have pre-distributed shared secrets in order to be able to perform the entire mechanism.

T

TDEA Triple DES.

Trapdoor one-way function, Trapdoor function, Trapdoor permutation See One-way function.

U

Uniform distribution In the context of this Technical Guideline, *uniformly distributed* generation of a random number from a base set M means that the generating process is practically indistinguishable from an ideally random (that means, from a truly random, equally distributed, independent) drawing of elements from M.

V

Volume encryption See partition encryption.

1. Introduction

In this technical guideline, the Federal Office for Information Security (BSI) provides an assessment of the security of selected cryptographic mechanisms, combined with a long-term orientation for their use. The recommendations made are reviewed annually and adapted if necessary. However, no claim is made to completeness, that means mechanisms not listed are not necessarily considered to be insecure by the BSI. Conversely, it is also wrong to conclude that cryptographic systems which only use the mechanisms recommended in this Technical Guideline as basic components are automatically secure: The requirements of the concrete application and the linking of different cryptographic and non-cryptographic mechanisms can lead to the fact that the recommendations made here cannot be implemented directly or that vulnerabilities arise. Due to these considerations, it must be emphasised in particular that the recommendations made in this Technical Guideline do not anticipate any decisions, for example as in the course of governmental evaluation and approval processes.

This Technical Guideline addresses primarily, in a recommendatory manner, developers who are planning to introduce new cryptographic systems from 2024 onwards. For this reason, this document deliberately refrains from mentioning cryptographic mechanisms which, although still considered secure at the present time, can no longer be recommended in the medium-term, since they show, if not yet exploitable, at least theoretical weaknesses. In the development of new cryptographic systems, various other documents issued by the BSI may also play a role, including [43, 44, 48, 45, 46, 49, 37, 40, 38, 47]. For certain applications, the specifications contained in these documents – in contrast to the recommendations of this Technical Guideline – are binding. A discussion of the regulations contained can be found in [58]. The following two sections first describe the security objectives as well as the selection criteria of the recommended cryptographic mechanisms. Further, very general information on the practical implementation of the recommended mechanisms is given.

The recommended cryptographic mechanisms for the following applications are listed in Chapters 3 to 8:

2 Asymmetric Encryption and Key Agreement,

3 Symmetric Encryption and Key Agreement,

4 Cryptographic Hash Functions,

5 Data Authentication,

6 Instance Authentication,

7 Secret Sharing and

8 Random Number Generators.

In the respective sections, the required (minimum) key lengths and other constraints to be observed are stated as well.

Often, various cryptographic algorithms must be combined with each other in order to ensure that a mechanism meets the security requirements placed on it. For example, it is often necessary not only to encrypt confidential data, but a recipient must also be sure who sent the data and/or

whether it was manipulated during transmission. Therefore, the data to be transmitted must additionally be authenticated by means of an adequate method. Another example for the need of the combination of cryptographic primitives are key agreement procedures. Here, it is important to know with whom the key agreement is carried out in order to be able to eliminate so-called man-in-the-middle attacks and unknown key share attacks [16]. This is achieved by schemes which combine key agreement and instance authentication. For these two application scenarios, Annex A specifies corresponding schemes that are constructed by combining the schemes listed in Chapters 3 to 8 and that meet the security level required in this Technical Guideline. In addition, Annex B recommends frequently used functions and algorithms that are required, for example, for key derivation for symmetric mechanisms or for the generation of prime numbers and other system parameters for asymmetric mechanisms. Finally, in Appendix C, recommendations are made for the use of selected cryptographic protocols. In the current version of this Technical Guideline, this only applies to the protocols SRTP and MLS; recommendations for TLS, IPsec and SSH have been transferred to the Technical Guidelines TR-02102-2 [33], TR-02102-3 [34] and TR-02102-4 [35] respectively.

1.1. Security Objectives and Selection Criteria

The security of cryptographic mechanisms essentially depends on the strength of the underlying cryptographic primitives. For this reason, this Technical Guideline only recommends mechanisms that can be assessed and evaluated on the basis of the results of many years of analysis and discussion. Other factors of central importance for security are the concrete implementations of the algorithms and the reliability of background systems, such as the public key infrastructures required for the secure exchange of certificates. The realisation of concrete implementations is not considered here, nor are possible problems related to patent law. Even though care was taken in the selection of the mechanism to ensure that the algorithms are free of patents, this cannot be guaranteed by the BSI. In addition, this Technical Guideline contains individual notes to possible difficulties and problems arising during in the implementation of cryptographic mechanisms, but these remarks are not to be understood as an exhaustive list of such problems.

Overall, all cryptographic mechanisms specified in this Technical Guideline achieve a security level of at least 120 bits when used with the parameters specified in the individual sections. The bit lengths recommended in this Technical Guideline for use in new cryptographic systems are based on this minimum level only to the extent that no recommended mechanism falls below it. The effective strength of the recommended mechanisms is in many cases higher than 120 bits. Thus, a security margin against possible future progress in cryptanalysis is provided. As already mentioned before, the converse is not true: mechanisms not specified in this Technical Guideline can nevertheless achieve the required level of security.

Table 1.1 shows the key lengths of selected algorithms and types of algorithms for which a security level of 120 bits is just achieved according to current knowledge.

Symmetric Schemes		Asymmetric Schemes		
Ideal Block Cipher	Ideal MAC	RSA	DSA[1]/DLIES	ECDSA/ECIES
120	120	2800	2800	240

Table 1.1: Examples of key lengths for a security level of at least 120 bits.

[1] The use of the DSA signature algorithm (see Section 5.3.2) is in this Technical Guideline only recommended until 2029 due to its low prevalence and the discontinuation in [98].

Table 1.2 summarises the *recommended* key lengths of different types of cryptographic primitives.

Block Cipher	MAC	RSA	DH $\mathbb{F}_p$	ECDH	ECDSA
128	128	3000	3000	250	250

Table 1.2: Recommended key lengths for various cryptographic mechanisms.

Key exchange schemes based on Diffie-Hellman are to be handled in Tables 1.1 and 1.2 in accordance with DSA/ECDSA.

In many applications, other security parameters besides the key length play a role in the overall security of a cryptographic system. In the case of message authentication codes (MACs), for example, the length of the digest output is an important security parameter in addition to the key length. Ideally, a MAC should in practice be indistinguishable for an attacker from a random function with a corresponding digest length. As long as this criterion is met, the attacker is left with the option of generating fake messages by guessing, with a per-attempt probability of success of 2^{-n} when n is the tag length. In many applications, $n = 96$ can be considered acceptable in such a situation, that is a tag length of $n = 96$ bits.

In case of block ciphers, the block width is a security parameter independent of the key length. In the absence of structural attacks on a block cipher, the most significant impact of a small block width is that keys have to be exchanged more frequently. The exact impact depends on the used mode of operation. This Technical Guideline does not recommend block ciphers that have a block width of less than 128 bits.

An important type of cryptographic primitives that do not process secret data at all are cryptographic hash functions. Here, the length of the digest value returned is the most important security parameter and should be at least 200 bits for general applications to achieve the minimum level of security required by this Technical Guideline. The hash functions recommended in Chapter 4 have a minimum hash length of 256 bits; deviations from this rule for special applications are discussed at appropriate places in this Technical Guideline.

1.2. General Remarks

Reliability of Predictions on the Security of Cryptographic Mechanisms When determining the size of system parameters (such as key length, size of the image domain for hash functions, etc.) not only the best algorithms known today for breaking the corresponding mechanisms and the performance of today's computers have to be taken into account, but above all a forecast of the future development of both aspects, see in particular also [73, 72, 47].

The development of the performance of classical computers can be estimated relatively well today. Fundamental scientific progress (either in terms of attack algorithms or, for example, the development of a cryptographically relevant quantum computer), on the other hand, cannot be predicted. Therefore, any prediction beyond a period of six to seven years is difficult, especially for asymmetric mechanisms, and even for this period of six to seven years, the predictions can turn out to be wrong due to unforeseen developments. The information provided in this Technical Guideline is therefore only limited to a period until the end of 2030.

General Guidelines for Handling Confidential Data with Long-Term Protection Requirements Since an attacker can store data and decrypt it later, there remains a fundamental risk to the long-term protection of confidentiality. This results in the following direct consequences:

- The transmission and storage of confidential data should be reduced to the necessary extent. This applies not only to plaintexts, but also, for example, in particular to the avoidance of stor-

ing session keys on any kind of non-volatile media, as well as their undelayed secure deletion as soon as they are no longer needed.

- The cryptosystem must be designed in such a way that a transition to larger key lengths and stronger cryptographic mechanisms is possible (cryptoagility).

- For data whose confidentiality has to remain secure in the long-term, it is recommended to choose for the encryption of the transmission via generally accessible channels such as the Internet, the strongest possible mechanisms of the recommended ones in this Technical Guideline. In most contexts, for example, AES-256 is considered stronger than AES-128 due to its longer key length. However, since such general assessments are difficult – in the concrete example, for example, in some (constructed) scenarios AES-192 and AES-256 are *weaker* than AES-128 against the best known attacks (see [15]) – the advice of an expert should already be sought at an early stage if possible.

- With regard to the selection of cryptographic components for a new application, it should generally be taken into account that the overall system is in general not stronger than the weakest component. Therefore, if a security level of, for example, 256 bits is aimed at for the overall system, all components must at least meet this security level. Selecting individual components that achieve a higher level of security against the best known attacks than the overall system may still make sense under certain circumstances, because this increases the robustness of the system against advances in cryptanalysis.

- In order to minimise the possibility of side-channel attacks and implementation errors, in the case of software implementations of the cryptographic mechanisms presented here, preference should be given to the use of open-source libraries over proprietary developments if it can be assumed that the functions used in the library have been subjected to broad public analysis. When evaluating a cryptosystem, the trustworthiness of all system functions must be assessed; in particular, this also includes dependencies of the solution on properties of the hardware used.

Focus of this Document The security assessment of the cryptographic mechanisms recommended in this Technical Guideline is carried out without taking the concrete use case into consideration. Other security requirements may arise for specific scenarios which may not be met by the mechanisms recommended in this Technical Guideline. Examples include the encryption of storage devices, the encrypted storage and processing of data on systems operated by external providers („Cloud Computing“ or „Cloud Storage“) or cryptographic applications on devices with extremely low computational resources („Lightweight Cryptography“). References to some of the mentioned application scenarios can be found in Section 1.6. This document can therefore support the development of cryptographic infrastructures, but cannot replace the assessment of the overall system by a cryptologist or anticipate the results of such an evaluation.

General Recommendations for the Development of Cryptographic Systems The following list summarises some principles that are generally recommended to be observed in the development of cryptographic systems:

- When planning systems for which cryptographic components are intended, collaboration with experts in the cryptographic field should be sought at an early stage.

- The cryptographic mechanisms must be implemented in trusted technical components in order to achieve the required level of security aimed for in this Technical Guideline.

- The implementations of the cryptographic mechanisms and protocols themselves must be included in the security analysis to prevent, for example, side-channel attacks or implementation weaknesses.
- If the conformity of a product with the requirements of this Technical Guideline is to be shown, the security of technical components and implementations has to be demonstrated according to the applicable protection profile provided by Common Criteria certificates or similar mechanisms of the BSI, for example in the course of an approval.
- After development and before productive use of a cryptographic system, an evaluation of the system should be carried out by independent experts who were not involved in the development. An assessment of the procedural security by the developers alone should not be considered reliable, even if the developers of the system have good cryptographic knowledge.
- The consequences of a failure of the security mechanisms used must be thoroughly documented. Wherever possible, the system should be designed in such a way that the failure or manipulation of individual system components is detected immediately and the security objectives are preserved by means of a transition to an adequate secure state.

1.3. Cryptographic Remarks

A cryptographic mechanism can often be used for different applications, for example signature methods can be used for both data authentication as well as for instance authentication. In general, different keys should be used for different applications. Another example is symmetric keys for encryption and symmetric data authentication. In concrete implementations, it must be ensured that different keys are used for each of the two mechanisms, which in particular cannot be derived from each other, see also Section A.1.

In some places, this Technical Guideline only provides an informative description of the cryptographic primitives. However, since cryptographic security can only be assessed within the framework of the exact specification and the protocol used in each case, the corresponding standards referred to here must be observed. Further specific information is given, if necessary, in the corresponding sections.

1.4. Implementation Aspects

Besides the cryptanalytic security of the algorithms, the security of their implementation, for example against side-channel and fault attacks, is crucial for the security of a cryptosystem. This is especially true for symmetric encryption methods. A detailed treatment of this topic is beyond the scope of this Technical Guideline, especially since the countermeasures to be taken in individual cases are also highly dependent on the concrete implementation. At this point, only the following general measures are recommended:

- Whenever it is possible with reasonable effort, cryptographic operations should be carried out in security-certified hardware components (for example, on a suitable smart card) and the keys used should not leave these components.
- Attacks that can be carried out by remote, passive attackers are inherently difficult to detect and can therefore lead to significant unnoticed data leakage over a long period of time. These include, for example, attacks exploiting variable bit rates, file lengths or variable response times of cryptographic systems. It is recommended to thoroughly analyse the effects of such side-channels on system security when developing a new cryptographic system and to take the results of the analysis into account in the development process.

- Both attacks on symmetric and asymmetric mechanisms have recently increasingly used attack methods based on mechanisms from the field of machine learning (ML) or artificial intelligence (AI). Neural networks in particular often achieve state-of-the-art results. It is becoming apparent that AI-based methods could be superior to the classic attack methods that are currently most commonly used (for example based on correlations or templates) in some use cases. Therefore, an AI-side-channel guide containing more detailed recommendations on this topic is in preparation.
- At the protocol level, the occurrence of error oracles should be prevented. This can be done most effectively by protecting all ciphertexts by a MAC. The authenticity of the ciphertext should be checked before any other cryptographic operations are performed and no further processing of non-authentic ciphertexts should take place.

As in other contexts, the general recommendation already mentioned several times applies here in particular to always use, wherever possible, components that have already been subjected to intensive analysis by a broad public and to involve experts in the development of new cryptographic systems from an early stage onwards.

1.5. Dealing with Legacy Algorithms

There are algorithms against which no practical attacks are known so far and which still have a high prevalence and thus a certain importance in some applications, but which are basically considered no longer state-of-the-art for new systems. We briefly discuss the most important examples below.

HMAC-MD5 The lack of collision resistance of MD5 is not yet an immediate problem in the HMAC construction with MD5 as the hash function [10], since the HMAC construction only requires a very weak form of collision resistance from the hash function. However, it seems fundamentally inadvisable to use primitives in new cryptosystems that have been completely broken in their original function. Systems using MD5 for cryptographic purposes are therefore not compliant with this Technical Guideline.

HMAC-SHA1 SHA1 is not a collision-resistant hash function. The generation of SHA1 collisions, while requiring moderate effort, is practically possible [75, 74, 107], even though, according to current knowledge, there are no known weaknesses when using SHA1 in constructions that do not require collision resistance (for example, as the basis for an HMAC, as part of the mask generation function in RSA-OAEP, or as a component of a pseudorandom number generator). However, as a basic security measure, it is recommended to use a hash function of the SHA2 or SHA3 family in these applications as well.

RSA with PKCS1v1.5 padding In principle, it is not recommended to use this format in new systems, neither for encryption nor for signature generation, since there are padding procedures with RSA-OAEP or RSA-PSS with more solid theoretical security properties. In addition, RSA implementations with PKCS1v1.5 padding have proven to be more vulnerable to attacks that exploit side-channel information or implementation errors.

1.6. Other Relevant Aspects

Finally, we would like to explicitly mention some important topics that are either not covered or not covered in detail in this Technical Guideline. The list explicitly does not claim to be complete.

Key Lengths for Information Requiring Long-Term Protection and in Systems with a Long Intended Period of Use For the purposes of this section, *information requiring long-term protection* means information whose confidentiality is intended to be maintained significantly longer than the period of time for which this Technical Guideline makes predictions about the suitability of cryptographic mechanisms, that means well beyond 2030. A reliable prognosis about the suitability of cryptographic mechanisms over the entire life cycle of a system is no longer possible in this case. It is recommended to provide protection mechanisms that go significantly beyond the minimum requirements of this Technical Guideline with the help of an expert. The following are examples of different ways to minimise risk:

- When developing new cryptographic systems with a projected long period of use, it is advisable to provide for the possibility of future operation with higher key lengths already during development. A possible future need to change the mechanisms used or the implementation of such mechanism changes should also be taken into account during the development of the original system („cryptoagility“).

- Already when introducing the system, higher asymmetric key lengths than required in this Technical Guideline should be used. An obvious option is to aim for a uniform security level of ≥ 128 bits for all system components. Guidance on the minimum asymmetric key lengths required for different security levels can be found in Table 2.2.

- Overall, the amount of information requiring long-term protection that is transmitted over public networks should be reduced to what is absolutely necessary. This applies in particular to information that is transmitted after encryption with a hybrid or asymmetric cryptographic mechanism.

For a more detailed discussion regarding long-term secure key lengths for asymmetric cryptographic mechanisms that are currently in wide use, we refer to [30, 72].

Lightweight Cryptography In this context, particularly restrictive requirements arise with regard to the computing time and memory requirements of the cryptographic mechanisms used. Depending on the application, the security requirements may also differ from the classical ones.

Response Times of a System When using cryptographic mechanisms in areas where tight specifications on the response times of the system must be adhered to, special situations may occur which are not dealt with in this guideline. The recommendations on the use of SRTP in Appendix C cover parts of this topic.

Hard Disk Encryption In the context of hard disk encryption, the problem arises that in most application scenarios neither encryption with data expansion nor a significant expansion of the amount of data that needs to be read from or written to the storage medium is acceptable. None of the recommended encryption modes is directly suitable as the basis of a hard disk encryption solution. Provided that an attacker cannot combine images of the state of the hard disk at several different points in time, XTS-AES offers relatively good security properties and good efficiency [87]. However, if the attacker can create copies of the encrypted storage medium at a larger number of different points in time, a certain, not necessarily insignificant, leakage of information must be assumed. For example, by comparing two images of a hard disk encrypted with XTS-AES made at different points in time, the attacker can immediately see which plaintext blocks on the hard disk have been changed within this period and which have not.

Disk Encryption of SSD Disks In connection with the encryption of a solid state drive (SSD), it is important to note that the SSD controller does not implement the overwriting of logical storage addresses physically in-place, but distributes them to different physical storage areas. Thus, the current state of an SSD always contains information about certain previous states of the storage medium. An attacker with good knowledge of how the SSD controller works can potentially exploit this to track successive states of a logical storage address. A single image of the encrypted SSD may thus be more valuable to an attacker than a single image of a classical hard disk.

Cloud Storage Similar problems as with the encryption of data media arise with the encrypted storage of entire logical drives on remote systems that are not under the control of the data owner (so-called cloud storage). If the provider of the remote server or its security measures cannot be trusted to a high degree, it must be assumed that an attacker can create disk images at any time without being noticed. If files with sensitive data are stored on a storage system that is regularly under foreign control, cryptographically strong file encryption should be applied before transmission. This also applies if the data is encrypted using volume encryption before it is transferred to the storage medium. The use of a volume encryption solution alone is only recommended if it includes effective cryptographic protection against manipulation of the data and if the other requirements for the use of the corresponding mechanism in general cryptographic contexts are met (for example, the requirement of unpredictable initialisation vectors). In particular, mechanisms should be selected in such a way that, unlike in XTS mode, no significant leakage of information through frequency analysis of successive states is to be expected when a block of data is written repeatedly.

Physical Aspects The present Technical Guideline essentially only addresses those aspects of the security of cryptographic systems that can be reduced to the underlying primitives. Physical aspects such as the emission security of information-processing systems or cryptographic systems whose security is based on physical effects (for example, quantum cryptographic systems) are not or only marginally covered in this Technical Guideline, nor are side-channel attacks, fault attacks and other physical security issues. Any comments in this regard are to be understood explicitly as exemplary references to potential risks without any claim to completeness.

Traffic Flow Analysis None of the mechanisms and protocols for data encryption described in this Technical Guideline achieve by themselves the objective of security against *traffic flow analysis* (so-called *traffic flow confidentiality*). Traffic flow analysis – that is, an analysis of an encrypted data stream taking into account the source, destination, time of existence of a connection, size of the transmitted data packets, data rate and time of transmission of the data packets – can allow significant conclusions to be drawn about the content of encrypted transmissions, see for example [6, 25, 106]. Traffic flow confidentiality is an objective that can usually only be fully achieved with a great deal of effort and is therefore not feasible in many applications that process sensitive information. However, it should be checked in each individual case by experts to what extent and what kinf of confidential information information is disclosed in a given cryptosystem through traffic flow analysis (and of course other side-channel attacks). Depending on the particular situation, the outcome of such an evaluation may necessitate significant changes to the overall system. It is therefore recommended that the resistance of a cryptographic system to disclosure of sensitive information through traffic flow analysis be considered as an objective from the outset in the development of new systems.

Endpoint Security The security of the endpoints of a cryptographically secured connection is essentially for the security of the transmitted data. When designing a cryptographic system, it must be clearly documented which system components must be trusted to achieve the intended security

objectives, and these components must be hardened against compromise in a manner appropriate to the context of use. Appropriate considerations must encompass the entire life cycle of the data to be protected as well as the entire life cycle of the cryptographic secrets generated by the system. Cryptographic mechanisms can reduce the number of components of an overall system whose trustworthiness must be ensured in order to prevent data leakage, but they cannot solve the basic problem of endpoint security.

Quantum-Safe Cryptography Encryption methods may need to protect data once transmitted for a long time. Attacks by future quantum computers should therefore be considered as part of risk management. On the other hand, the standardisation of quantum computer-resistant cryptographic mechanisms has not yet been completed, and at the present time there is also not as much knowledge about their secure implementation as is the case with classical public key methods. Chapter 2 provides some preliminary recommendations for dealing with issues in this area. An overview of the current state of development of the technology underlying quantum computing can be found in the study [47], among others.

This Technical Guideline does not provide any recommendations, or at least no comprehensive recommendations, with regard to the implementation of mechanisms in the previously mentioned areas. It is therefore advised that in the development of cryptographic systems as a whole – but especially in these areas – experts from the relevant fields should be involved in the development work from the very beginning.

2. Asymmetric Encryption Schemes and Key Agreement

Key agreement schemes are used to exchange a shared secret encryption key over an insecure channel. In practice, asymmetric encryption methods (also known as public key encryption methods) are usually only used for the transmission of symmetric keys, as they are considerably less efficient than symmetric encryption methods. This chapter deals with both asymmetric encryption and key agreement mechanisms.

The security of current "classical" asymmetric methods is based on the assumed difficulty of certain mathematical problems, where "classical" is to be understood in the sense of "provides protection against attacks that can be implemented on classical hardware". The underlying problems of the classical asymmetric methods that are currently used in practice are in most cases the factorization problem (Factorization problem) and the discrete logarithm problem (Discrete-logarithm-problem (DL)).

However, even if no efficient *classical* algorithms for solving these problems are currently known, *Shor's algorithm* [105] from the 1990s provides a corresponding quantum algorithm. If a sufficiently large quantum computer is available, Shor's algorithm is able to efficiently solve the factorization and the discrete logarithm problem and thus break the classical asymmetric encryption. It is therefore recommended to use quantum-safe key derivation methods (see Section 2.4) for systems designed to protect information that requires long-term protection. Since the quantum-safe methods are comparatively new or have been less investigated, especially with regard to implementation security, this Technical Guideline currently only recommends the hybrid use (see Section 2.2) of quantum-safe methods in combination with classical methods.

It is absolutely essential that key agreement schemes are combined with instance authentication schemes, since otherwise there is no way to decide with which party the key agreement is performed. This chapter only deals with the key derivation and encryption methods without the additionally required authentication mechanism. For a secure key agreement, the methods described here must be embedded in corresponding protocols that among others further ensure the authenticity of the communication partners. For this reason, we give only general ideas for key agreement schemes in this chapter and refer to Section A.2 for specific key agreement schemes that also include instance authentication.

After successful key agreement, both communication partners are in possession of a shared secret (or several secrets in the case of hybrid key agreement). For recommended methods for generating symmetric keys from this secret or these secrets, see Section 2.2.

Essentially, the use of a key derivation function is recommended for this task. In some situations, it may make sense to allow a pre-distributed secret to enter the key derivation function. This can be used, for example, to separate different user groups. Also, additional protection against attacks on the key agreement scheme can be achieved in this way. With regard to a separation of different user groups, it can also be reasonable to take further public data into account that is specific to both communication partners in the key derivation.

Remark 2.1 When cryptographic keys are negotiated with a key agreement scheme or transmitted securely with a key transport scheme, these keys or the cryptographic mechanisms using these keys have at most the same security level as the key agreement or key transport scheme. Since there is a possibility that an attacker records the communication during key agreement or key transport, changed recommendations for key agreement or key transport schemes also have an impact on pre-

viously negotiated or transmitted keys. For example, if the key agreement or key transport scheme loses conformance to this Technical Guideline, the keys negotiated or transferred with it should also no longer be used.

The asymmetric cryptographic mechanisms recommended in this document require as components further subcomponents, such as hash functions, message authentication codes, random number generators, key derivation functions and/or block ciphers, which in turn must also meet the requirements of the present Technical Guideline in order to achieve the desired level of security. Relevant standards [59] sometimes recommend the use of mechanisms that are not recommended in the present Technical Guideline. In principle, it is recommended to observe the following principles when implementing a standard:

- For cryptographic subcomponents, only the respective mechanisms recommended in this Technical Guideline should be used.
- If in this Technical Guideline mechanisms are recommended that are standardised exclusively with a non-recommended subcomponent, this subcomponent is to be regarded as recommended in the context of the mechanism.
- If this is not compatible with standards compliance, an expert must be involved and the final decisions made regarding the chosen cryptographic subcomponents have to be documented in detail and justified from a security point of view.

In the selection of the recommended asymmetric encryption schemes, care has been taken to ensure that only probabilistic algorithms[1] are recommended here. In particular each time a ciphertext is computed, a new random value is needed. Some of the requirements for these random values cannot be met directly by generating equally distributed values of fixed bit length. More details on these random values are given in the sections on the corresponding schemes.

Remark 2.2 (Side-Channel Attacks and Fault Attacks) Depending on the situation at hand, various types of side-channel attacks and/or fault attacks may be relevant to asymmetric encryption schemes and/or asymmetric digital signature schemes. This topic cannot be dealt with comprehensively in the present Technical Guideline. The security of an implementation against side-channel and fault attacks must therefore always be examined on a case-by-case basis. Detailed recommendations on this topic can be found for cryptographic mechanisms based on elliptic curves in [46] and for RSA, $\mathbb{F}_p$-DH and corresponding signature methods in [45].

Remark 2.3 (Public Key Infrastructures) The asymmetric encryption methods described in this Technical Guideline do not in themselves offer any protection against man-in-the-middle attacks. The security guarantees of the described mechanisms are therefore only valid if man-in-the-middle attacks can be reliably prevented by additional mechanisms. For this, an authentic distribution of the public keys of all participants must be ensured.

This can be done in various ways, usually a public key infrastructure (PKI) is used. In a PKI, the problem of authentic distribution of public keys is reduced to the distribution of the root certificates of the PKI. When planning a PKI for an asymmetric encryption or signature procedure, it is recommended to consider the aspects listed below. This is not an exhaustive list of development requirements for public key infrastructures, but merely a list of comparatively generic aspects that are recommended to be considered when developing a PKI; for more information see also [39]. During the development and evaluation of a concrete system, further requirements usually arise, which are not considered here. The development of a suitable PKI for a new cryptographic application is not a trivial task and should therefore only be dealt with in close consultation with appropriate experts.

[1] The RSA algorithm itself is not probabilistic, but the padding method for RSA recommended here is.

- When issuing certificates, the PKI should verify that the applicant is in possession of a private key to his public key. This can be done, for example, by a challenge-response mechanism for instance authentication, which requires knowledge of the private key. It is also conceivable to generate the key pairs in an environment that is secure from the PKI's point of view, if it is combined with a secure transport of the generated key pairs to the end user.
- There should be possibilities for the deactivation of certificates in a timely manner and it should not be possible for an attacker to prevent a verifying party from having the information about the current status of a certificate available at the time of verification without being noticed.
- Certificates should only be issued with a limited validity period.
- All certificate issuers must be trustworthy.
- A certificate should indicate whether it authorises the signing of further certificates. In general, any system that comes into contact with a certificate should be able to clearly determine what this certificate may be used for.
- The length of certificate chains should be limited upwards (by a value as low as possible).

2.1. Use of Quantum-Safe Mechanisms

The classical, asymmetric key agreement and encryption mechanism that are currently used are under increasing threat by the ongoing development of sufficiently large quantum computers. Even if the quantum computers available today are not yet capable of breaking cryptographic mechanisms, the threat is already relevant today. In particular, for data with longer-term protection requirements, since encrypted data can already be stored for later decryption ("Store Now, Decrypt Later"). In addition, potentially long migration times to new cryptographic algorithms must be taken into account. It is therefore advisable to use quantum-safe mechanisms in the very near future, especially for systems that process data with longer-term protection requirements. These methods should only be used in combination with a classical key derivation method; further details can be found in Section 2.2.

Remark 2.4 In the context of post-quantum cryptography, a cryptographic method whose secret key is based on the combination of a secret key from a quantum-safe method with a secret key from a classical method is also referred to as "hybrid method" or "hybrid key agreement". This term should not be confused with the term "Hybrid encryption", which refers to the combination of symmetric and asymmetric encryption. However, it is usually clear from the context which notion is meant, so that in the sequel no further clarification is given unless necessary.

A fundamentally different approach to key agreement than post-quantum cryptography (PQC) is offered by quantum key distribution (QKD). In contrast to post-quantum cryptography, QKD addresses the problem of establishing a secure communication channel by making use of quantum physical effects. However, the practical restrictions of QKD, such as limited transmission distances and the need to use specialised hardware, are severely limiting compared to the use of PQC mechanisms. Therefore, QKD is only suitable for specific use cases. Furthermore, since no standardised protocols with associated security proofs are available yet, the BSI is not making any recommendations of suitable protocols at this time. As soon as the necessary preconditions are met, the BSI plans in the medium-term to make recommendations on protocols, authentication mechanisms and the use of QKD. Irrespective of this, the BSI does not and will not recommend the use of the one-time pad alone with keys obtained via QKD or via other key agreement mechanisms in the future.

2.2. Key Derivation and Hybridisation

After a key agreement, both parties are in possession of a shared secret from which symmetric keys, for example for encryption and data authentication, can be derived using a key derivation function. For recommended mechanisms for key derivation, please refer to Section B.1.1.

In general, the quantum-safe methods recommended in this Technical Guideline are not yet given the same confidence as the established classical methods, as they among others have not been equally well investigated with regard to side-channel resistance or implementation security. This Technical Guideline therefore recommends the use of a quantum-safe mechanism only in combination with a classical key derivation mechanism. Hybrid key agreement should be secure as long as one of the methods used is secure. It is hereby important to consider in detail how the shared secrets are combined with each other and how context-dependent information is included so that the desired property mentionend before is actually achieved.

For recommended mechanisms for hybrid key derivation, please refer to Section B.1.2.

2.3. Classical Asymmetric Mechanisms

In simplified terms, the most practically relevant asymmetric encryption and signature methods are based either on the difficulty of the problem of calculating discrete logarithms in suitable representations of finite cyclic groups (Discrete-logarithm-problem (DL)) or on the difficulty of decomposing large integers into their prime factors (Factorization problem). Occasionally, the question arises which of these two approaches is to be considered cryptographically more secure. The present Technical Guideline regards the factorisation of large numbers, the RSA problem, the problem of computing discrete logarithms in suitable fields $\mathbb{F}_p$ (p prime), the problem of computing discrete logarithms in suitable elliptic curves, and the corresponding Diffie-Hellman problems as well-studied hard problems, and there is no reason in this respect to prefer mechanisms based on factorisation to mechanisms based on discrete logarithms, or vice versa. For particularly high security levels, the use of EC mechanisms is advantageous for efficiency reasons, see also Table 2.2.

Remark 2.5 For asymmetric mechanisms, there are usually different equivalent, practically relevant representations of the private and public keys. The bit length of the keys on a storage medium can vary depending on the chosen representation of the keys. For the exact definition of the key length for the recommended asymmetric cryptographic mechanisms, we therefore refer to the entry Key length in the glossary.

The following Table 2.1 provides an overview of the recommended classical asymmetric encryption and key agreement schemes as well as key lengths l in bits.

Scheme	RSA	DLIES	ECIES	DH	ECDH
Key Length l in bits	3000	3000	250	3000	250
Reference	[82]	[1]	[1, 59]	[1, 102]	[1, 102]
More detailed information in	Section 2.3.2	Section 2.3.3	Section 2.3.4	Section 2.3.5	Section 2.3.6

Table 2.1: Recommended classical asymmetric encryption and key derivation schemes as well as key lengths and normative references.

Remark 2.6 The current recommendations result in only a small buffer between the minimum security level of about 125 bits achieved by the recommended EC bit lengths and the security level of 120 bits targeted in this Technical Guideline. In certain applications that have particularly high demands on security or whose security must be guaranteed substantially exceeding the prediction

period of this Technical Guideline, it may therefore make sense to provide significantly longer key lengths for EC mechanisms in order to increase the security buffer. The key length requirements of the Country Signing CA from [38], for example, can be explained in this way. Since the security of EC mechanisms depends on the assumption that an attacker cannot use any of the mathematical structure of a given elliptic curve to calculate discrete logarithms faster than the Pollard-Rho algorithm allows, it is conceivable that in the coming years the requirements of the present Technical Guideline will be increased in this area as a basic precaution. It is also recommended as a basic security measure to use curve parameters in EC algorithms that have been generated verifiably at random, whose construction has been comprehensibly documented and whose security has been subjected to a thorough analysis. An example of such curve parameters are the Brainpool curves [76].

2.3.1. Equivalent Key Lengths for Symmetric and Classical Asymmetric Cryptographic Mechanisms

The recommendations of this Technical Guideline on the key lengths of classical asymmetric cryptographic mechanisms are based on calculations of equivalences of symmetric and asymmetric key lengths, which are based on the following basic assumptions:

- For mechanisms based on elliptic curves: It is assumed that no that over the prediction period of this Technical Guideline, no attacks become known that solve the Diffie-Hellman problem on the used curve significantly faster than the calculation of discrete logarithms on the same curve. It is further assumed that the computation of discrete logarithms on the elliptic curve used is not possible with significantly less complexity (measured by the number of group operations performed) than for generic representations of the same cyclic group.[2] For a generic group G, a complexity of computing discrete logarithms of $\approx \sqrt{|G|}$ group operations is assumed.

- For RSA and mechanisms based on discrete logarithms in $\mathbb{F}_p^*$: It is assumed that over the prediction period of this Technical Guideline, no attacks become known that are more efficient than the general number field sieve when the parameters are chosen as recommended in this Technical Guideline. For RSA and mechanisms based on discrete logarithms in $\mathbb{F}_p^*$, the same key lengths are recommended. In the case of mechanisms based on discrete logarithms, it is assumed that no mechanism exists to solve the Diffie-Hellman problem in a subgroup $U \subset \mathbb{F}_p^*$ with $\mathrm{ord}(U)$ prime more efficiently than by computing discrete logarithms in U.

- It is assumed that there is no application of attacks using quantum computers.

These assumptions are pessimistic from an attacker's point of view in that they contain no scope for structural progress in cryptanalysis of asymmetric mechanisms. Progress incompatible with the above assumptions may be of a very specific nature and relate, for example, to new insights into *a single* elliptic curve. Although in principle a calculation with 2^{120} elementary operations is not considered practical for the period of time relevant to this Technical Guideline, all recommended key lengths are above the minimum 120-bit security level targeted in this document.

With regard to mechanisms whose security is based on the difficulty of calculating discrete logarithms, especially discrete logarithms in elliptic curves, attacks that require oracle access to operations with a user's private key may also be relevant. Such attacks can significantly speed up the calculation of discrete logarithms in a group, see for instance attacks using a static-Diffie-Hellman oracle [23, 26].

For the assessment of runtimes, we follow [31]. In particular, as in [31], we assume that factorising a 512-bit number of arbitrary form is roughly equivalent to the computational cost of 2^{50} DES

[2] Algorithms operating on a generic representation of a group have only black-box access to elements and group operations. Intuitively, one can think of something like an oracle that accepts encrypted group elements and outputs the result of group operations in encrypted form.

$\log_2(R)$	ECDLP	Factorisation/DLP in $\mathbb{F}_p^*$
60	120	700
70	140	1000
100	200	1900
128	256	3200
192	384	7900
256	512	15500

Table 2.2: Approximate computational effort R (in multiples of the computational effort for a simple cryptographic operation, for example the one-time evaluation of a block cipher on a block) for the computation of discrete logarithms in elliptic curves (ECDLP) or the factorisation of general composite numbers with the specified bit lengths.

operations. Using the methods given there – without any security margins for progress in factorisation techniques or techniques for efficient computation of discrete logarithms in the respective groups, respectively – yields approximately the equivalences reproduced in Table 2.2 (compare [31, Table 7.2] and [30, Table 4.1]).

For recommended key lengths, please refer to Table 2.1.

2.3.2. RSA

The RSA algorithm, named after its inventors R. Rivest, A. Shamir and L. Adleman, is an asymmetric cryptographic mechanism that can be used for both encryption and digital signing. It uses a key pair consisting of a private key, which is used to decrypt or sign data, and a public key, which is used to encrypt or verify signatures. The security of the mechanism is based on the assumed difficulty of decomposing integers into the product of their prime factors.

Key Generation

1.) Choose two prime numbers p and q randomly and independently of each other. The numbers p and q should be of comparable bit length and not too close to each other. Otherwise if, for example, p and q are chosen independently from a too narrow interval, attacks based on knowledge of the leading bits of p and q are possible.

More details on the procedure for prime number generation can be found in Section B.5. If p and q are chosen according to Section B.5, the previously mentioned vulnerability does not occur.

For more details on the procedure for generating prime numbers, see Section B.5.

2.) Choose the public exponent $e \in \mathbb{N}$ under the constraints

$$\gcd(e, (p-1)\cdot(q-1)) = 1 \quad \text{and} \quad 2^{16}+1 \leq e \leq 2^{256}-1.$$

3.) Calculate the private exponent $d \in \mathbb{N}$ as a function of e under the constraint

$$e \cdot d = 1 \bmod \operatorname{lcm}(p-1, q-1).$$

Then (n, e) represents the public key, where $n = p \cdot q$ is the so-called modulus, and d is the private key. In addition to the private key d, the two prime numbers p and q must also be kept secret, otherwise everyone would be able to calculate the private exponent from the public key (n, e) as described under Item 3. It is recommended not to store any data from the key generation persistently

except the generated keys and to overwrite all generated data in the computer memory after the key generation. It is further recommended to store the private key on a protected storage medium and/or encrypted in such a way that only authorised users can perform decryption operations.

Remark 2.7 (i) The order of the choice of exponents during key generation, that means first the choice of e and then that of d, is intended to prevent the random choice of small private exponents, see [20].

(ii) When using probabilistic prime number tests to generate the two primes p and q, the probability that one of the numbers is composite after all should be at most 2^{-120}, see Section B.5 for suitable methods.

Encryption and Decryption For the encryption and decryption, please refer to the standard [82]. It is to be noted that in addition the message must be formatted to the bit length of the modulus n before the private key d is applied. The formatting procedure must be chosen carefully, the following procedure is recommended:

EME-OAEP, see [82].

Table 2.3: Recommended formatting method for the RSA encryption algorithm.

Using the older PKCS#1v1.5 paddings is not recommended, as variants of Bleichenbacher's attack [17] have repeatedly turned out to be problematic, see for example [18] for a recent example.

Key Length The length of the modulus n should be at least 3000 bit.

A necessary condition for the security of the RSA mechanism is that it is practically impossible to decompose the modulus n into its prime factors without knowledge of p and q. With the recommended minimum bit length of 3000 bits, this is the case according to current knowledge.

Remark 2.8 Since the modulus n is very large, the bit representations of the numbers used in the computer are also very long. The Chineses Remainder Theorem allows the computations when en- or decrypting or signing messages to be performed in the two smaller groups of sizes p and q instead of in one group of size n, and the result to be combined afterwards. Since $p, q \ll n$, this computation is more efficient overall. This variant is also called CRT-RSA after the acronym of the Chinese remainder theorem (CRT). The private key in this case consists of the components $(n, d, p, q, d_p, d_q, q_{\text{inv}})$, where

$$d_p = d \bmod p - 1, \qquad d_q = d \bmod q - 1, \qquad q_{\text{inv}} = q^{-1} \bmod p.$$

2.3.3. DLIES Encryption Scheme

A *Discrete Logarithm Integrated Encryption Scheme* is a hybrid encryption scheme where the security of the asymmetric component is based on the difficulty of the Diffie-Hellman problem in a suitable subset of $\mathbb{F}_p^*$. In the following, we describe a version of DLIES that is compatible with the rest of the recommendations in this Technical Guideline, closely following [1] in the description of the scheme.

A DLIES requires the following components:

- Symmetric encryption scheme E_K: All combinations of block cipher and operating mode recommended in this policy are suitable for this purpose.

- Message Authentication Code MAC_{KM}: The mechanisms recommended in section 5.2 may be used.
- Key derivation function H: For example, H can be a hash function if its output is at least the length of the entire symmetric key material to be derived. Alternatively, the key derivation function recommended in Section B.1 or one of the key derivation functions proposed in [59] may be used to generate derived key material of the desired length from the given data.

In addition, a DLIES requires key material as described in the following section on key generation.

Key Generation

1.) Randomly choose a prime q of appropriate bit length (see subsection on key lengths).

2.) Randomly choose k of a bit length that ensures that kq is of the length of the key to be generated. Repeat this step until $p := kq + 1$ is prime.

3.) Choose an $x \in \mathbb{Z}_p^*$ such that $x^k \neq 1 \bmod p$ and set $g := x^k$. Then g is an element of order q in $\mathbb{Z}_p^*$.

4.) Randomly choose a number $a \in \{2, \ldots, q-1\}$ and set $A := g^a$.

Then (p, g, A, q) is the public key and a is theprivatesecret key.

Encryption Given are a message $M \in \{0,1\}^*$ and a public key (p, g, A, q) that can be reliably assigned to the authorised recipient E of the message. For encryption, the sender S chooses a random number $b \in \{1, \ldots, q-1\}$ and calculates $B := g^b$, $X := A^b$ and from these $h := H(X)$. Suffiently many bits are taken from h to form a key K for the symmetric encryption method and a key KM for the MAC. From the message M, S computes the ciphertext $C := E_K(M)$ and a MAC $T := \mathrm{MAC}_{KM}(C)$ and sends the triple (B, C, T) to the receiver E.

Decryption Receiver E receives (B, C, T) and computes $X := B^a$ and therewith further $h := H(X)$, K and KM. It calculates $T' := \mathrm{MAC}_{KM}(C)$ and checks whether $T = T'$. If this is not the case, the decryption process stops. If, on the other hand, $T = T'$, then E recovers the message through $M = E_K^{-1}(C)$.

Key Length The length of the prime number p should be at least 3000 bits. The length of the prime q should be at least 250 bits in both cases. F

Remark 2.9 The DLIES mechanism is a probabilistic algorithm, since several random numbers are required during key generation, including a random number $b \in \{1, \ldots, q-1\}$ that must be chosen randomly with respect to the uniform distribution on $\{1, \ldots, q-1\}$. For recommended algorithms for generating the random number b, please refer to section B.4.

Remark 2.10 The efficiency of the key generation mechanism described at the beginning of the section can be increased by having multiple users use the values (p, q, g) so that they can be precomputed once. Alternatively, it is also possible to use published parameters. In this case, the present Technical Guideline recommends using the MODP groups from [70] or the ffdhe groups from [53], in each case combined with the choice of suitable key lengths (this means that, for instance, MODP-1536 is *not* regarded as suitable, independently of the projected period of use). In each of the previously mentioned groups, $q = (p-1)/2$ and $g = 2$. The use of a common p by multiple users is recommended only when $\log_2(p) \geq 3000$, since the computation of discrete logarithms can be simplified by precomputation attacks that depend only on the parameter p.

2.3.4. ECIES Encryption Scheme

An *Elliptic Curve Integrated Encryption Scheme* (ECIES) is a hybrid encryption scheme in which the security of the asymmetric component is based on the Diffie-Hellman problem in the particular elliptic curve used. In the following, we describe a version of ECIES that is compatible with the other recommendations of the present Technical Guideline, closely following [1] in the description of the scheme.

The description of ECIES reproduced here is almost completely identical to the description of the closely related mechanism DLIES in Section 2.3.3. The main reason for treating the two schemes separately are potential difficulties that could arise from different notations and the different recommendations regarding secure key lengths for the two mechanisms. ECIES-HC in [59] is recommended as a normative reference. For an overview of the standardisation of ECIES and DLIES, we refer to [78].

An ECIES requires the following components:

- Symmetric encryption scheme E_K: All combinations of block cipher and operating mode recommended in this policy are suitable for this purpose.
- Message Authentication Code MAC_{KM}: The mechanisms recommended in Section 5.2 may be used.
- Key derivation function H: For example, H can be a hash function if its output is at least the length of the entire symmetric key material to be derived. Alternatively, the key derivation function recommended in Section B.1 or one of the key derivation functions proposed in [59] may be used to generate derived key material of the desired length from the given data.

In addition, an ECIES requires key material as described in the following section on key generation.

Key Generation

1.) Generate cryptographically strong EC system parameters (p, a, b, P, q, i), see Section B.3.

2.) Choose d randomly and uniformly distributed in $\{1, \ldots, q-1\}$.

3.) Set $G := d \cdot P$.

The EC system parameters (p, a, b, P, q, i) together with G form the public key and d is the private key. It is recommended to use the curve parameters given in Table B.4.

Encryption Given are a message $M \in \{0,1\}^*$ and a public key (p, a, b, P, q, i, G) that can be reliably assigned to the authorised recipient E of the message. For encryption, the sender S chooses a random number $k \in \{1, \ldots, q-1\}$ and calculates $B := k \cdot P$, $X := k \cdot G$ and from these $h := H(X)$. Sufficiently many bits are taken from h to form a key K for the symmetric encryption method and a key KM for the MAC. From the message M, S computes the ciphertext $C := E_K(M)$ and a MAC $T := \mathrm{MAC}_{KM}(C)$ and sends the triple (B, C, T) to the receiver E.

Decryption Receiver E receives (B, C, T) and calculates $X := d \cdot B$ and therewith $h := H(X)$, K and KM. He determines $T' := \mathrm{MAC}_{KM}(C)$ and checks whether $T = T'$ holds. If it does not, it aborts the decryption process. If $T = T'$, then E recovers the message by $M = E_K^{-1}(C)$.

Key Length For the order q of the base point P should be at least $q \geq 250$.

A necessary condition for the security of the ECIES mechanism is that it is practically impossible to solve the Diffie-Hellman problem in the subgroup generated by P. This is the case for the curve parameters recommended in Table B.4 according to the current state of knowledge.

Remark 2.11 The presented ECIES mechanism is a probabilistic algorithm, since in the second step of the key generation a random number $k \in \{1, \ldots, q-1\}$ must be chosen randomly with respect to the uniform distribution on $\{1, \ldots, q-1\}$. For recommended algorithms for generating the random number k, please refer to Section B.4.

2.3.5. Diffie-Hellman Key Agreement

The security of this mechanism is based on the assumed difficulty of the Diffie-Hellman problem in groups (respective subgroups of) $\mathbb{F}_p^*$, where p is a prime number.

System Parameters

1.) Randomly choose a prime number p.

2.) Choose an element $g \in \mathbb{F}_p^*$ with $\mathrm{ord}(g)$ prime and $q := \mathrm{ord}(g) \geq 2^{250}$.

The triple (p, g, q) must be authentically exchanged in advance between the parties A and B involved in the communication, where the same system parameters may in principle be used by many users. For the generation of suitable system parameters see Remark 2.10.

Key Agreement

1.) A chooses a random value $x \in \{1, \ldots, q-1\}$ according to the uniform distribution and sends $Q_A := g^x$ to B.

2.) B chooses a random value $y \in \{1, \ldots, q-1\}$ according to the uniform distribution and sends $Q_B := g^y$ to A.

3.) A calculates $(g^y)^x = g^{xy}$.

4.) B calculates $(g^x)^y = g^{xy}$.

The key agreement, too, must be secured by means of strong authentication to prevent man-in-the-middle attacks. The negotiated shared secret is g^{xy}. A mechanism for the subsequent key derivation from this secret is recommended in Section B.1.

Key Length The length of p should be at least 3000 bits.

Remarks on the Implementation A number of implementation errors are common in the implementation of the Diffie-Hellman protocol. Some of these implementation problems are discussed in [102]. It is recommended that particular attention be paid to [102, Section 7].

2.3.6. EC Diffie-Hellman Key Agreement

The security of this mechanism is based on the assumed difficulty of the Diffie-Hellman problem in elliptic curves.

System Parameters Choose cryptographically strong EC system parameters (p, a, b, P, q, i) according to B.3. Let the elliptic curve thus defined be denoted by C and let $\mathcal{G}$ be the cyclic subgroup generated by P. The system parameters (p, a, b, P, q, i) must be authentically exchanged in advance between the parties A and B involved in the communication.

Key Agreement

1.) A chooses a random value $x \in \{1, \ldots, q-1\}$ according to the uniform distribution and sends $Q_A \coloneqq x \cdot P$ to B.

2.) B chooses a random value $y \in \{1, \ldots, q-1\}$ according to the uniform distribution and sends $Q_B \coloneqq y \cdot P$ to A.

3.) A calculates $x \cdot Q_B = xy \cdot P$.

4.) B calculates $y \cdot Q_A = xy \cdot P$.

The key agreement, too, must be secured by means of strong authentication. The negotiated secret is $xy \cdot P$. A mechanism for subsequent key derivation from this secret is recommended in Section B.1.

Wherever possible, it is recommended to test on both sides during the execution of the key agreement whether the points Q_A and Q_B have been chosen according to the requirements of the protocol and to abort the protocol if not. If the above protocol is executed correctly, $Q_A \in \mathcal{G}$, $Q_B \in \mathcal{G}$, $Q_A \neq \mathcal{O}$ and $Q_B \neq \mathcal{O}$ should hold. As part of the test of $Q_A, Q_B \in \mathcal{G}$, it should also be explicitly checked whether $Q_A, Q_B \in C$. Further remarks can be found in [37, Section 4.3.2.1].

Key Length The length of q should be at least 250 bits.

Remarks on the Implementation There are several common implementation errors when implementing a Diffie-Hellman key exchange. Some of these implementation problems are addressed in [102]. It is recommended to observe [102, Section 7], furthermore the remarks in [37, Section 4.3] and the AIS 46 [46] should be taken into account.

2.4. Quantum-Safe Asymmetric Mechanisms

The quantum-safe asymmetric mechanisms for key agreement that are currently being standardised are either based on lattices or binary codes. The methods were designed specifically as "key encapsulation mechanism" (KEM, see Key encapsulation mechanism, KEM) due to their intended use, i.e. the distribution or agreement of key material.

2.4.1. FrodoKEM Key Agreement

FrodoKEM is a lattice-based method based on the learning with errors problem (LWE, see Learning with errors, LWE). NIST has decided not to standardise FrodoKEM because ML-KEM (see Section 2.4.3), which is also based on LWE, is more efficient. However, since FrodoKEM, unlike ML-KEM, is based on unstructured grids, it is considered the more conservative choice. FrodoKEM is currently being standardised at ISO; future versions of this Technical Guideline will reference the corresponding standard.

FrodoKEM with the following parameters is considered to be cryptographically suitable for the long-term protection of confidential information at the security level targeted in this Technical Guideline:

FrodoKEM-976 and FrodoKEM-1344, see [3, Section 2.5].

Table 2.4: Recommended parameters for FrodoKEM.

2.4.2. Classic McEliece Key Agreement

Classic McEliece is a code-based KEM. The underlying mechanism, which is instantiated with binary Goppa codes, is about as old as the RSA mechanism and is therefore considered conservative and very thoroughly analysed. One disadvantage are the comparatively large public keys; on the other hand, Classic McEliece has very short ciphertexts.

Classic McEliece is currently being standardised by ISO and possibly also by NIST after the 4th round of the standardisation process. Future versions of this Technical Guideline will reference corresponding standards.

Classic McEliece with the following parameters is considered to be cryptographically suitable for the long-term protection of confidential information at the security level targeted in this Technical Guideline:

• mceliece460896, mceliece6688128 and mceliece8192128, see [2, Section 7], • mceliece460896f, mceliece6688128f and mceliece8192128f (faster variants), see [2, Section 7].

Table 2.5: Recommended parameters for ClassicMcEliece-KEM.

2.4.3. ML-KEM (CRYSTALS-Kyber) Key Agreement

As part of its PQC standardisation process, NIST is expected to publish a standard for the ML-KEM method (also known as CRYSTALS-Kyber) in the course of 2024. This is a lattice-based KEM whose security is related to the “module learning with errors” problem. BSI intends to include ML-KEM in the recommendations of this Technical Guideline after publication of the standard with the parameter sets corresponding to NIST Security Strength Categories 3 and 5.

3. Symmetric Encryption Schemes

This chapter deals with symmetric encryption schemes, that is, schemes in which the encryption and decryption keys are identical – in contrast to asymmetric schemes, where the private key practically cannot be calculated from the public key without additional information. For asymmetric encryption methods, which in practice are usually only used as key transport schemes, please refer to Chapter 2.

Symmetric encryption schemes are used to guarantee the confidentiality of data that is transmitted, for example, via a public channel such as the telephone or Internet. Authenticity and/or integrity of the data is usually not automatically guaranteed. For integrity protection, see Chapter 5 and Section A.1. Even in cases where at first glance the protection of the confidentiality of transmitted data seems to be the dominant or even the only security objective, neglecting integrity-securing mechanisms can easily lead to weaknesses in the overall cryptographic system, which then also makes the system vulnerable to attacks on confidentiality. In particular, such vulnerabilities can arise from certain types of active side-channel attacks, for an example see for instance [108].

The development of fault-tolerant quantum computers has a much less severe impact on the security of symmetric mechanisms than on the security of asymmetric mechanisms. The use of Grover's algorithm [54] could theoretically accelerate the search of the key space of symmetric mechanisms quadratically. Whether an acceleration compared to a classic exhaustive search of the key space can also be achieved in practice is the subject of current research and has not been definitively answered yet, see e.g. [68]. Nevertheless, especially for applications with high or long-term protection requirements or long-living systems it is advisable to use a key length of 256 bits for the symmetric encryption methods recommended below.

Furthermore, this chapter also presents symmetric key agreement and transport schemes as well as key update mechanisms.

3.1. Block Ciphers

General Recommendations A block cipher is an algorithm that encrypts a plaintext of fixed bit length (for example 128 bits) by means of a key to a ciphertext of the same bit length. This bit length is also called *block size* of the cipher. For the encryption of plaintexts of other lengths, block ciphers are applied in different modes, see Section 3.1.1. For new cryptographic applications, only block ciphers whose block size is at least 128 bits should be used.

The following block ciphers are recommended for use in new cryptographic systems:

AES-128, AES-192, AES-256, see [84].

Table 3.1: Recommended block ciphers.

In Version 1.0 of the present Technical Guideline, other block ciphers were also recommended. However, their security has been examined much less intensively since the end of the AES-competition than that of the Rijndael algorithm, which emerged from the competition as the winner and thus future AES. This applies both to classical cryptanalytic attacks and to other

security aspects, for example the side-channel resistance of concrete implementations. For this reason, the present version of this Technical Guideline does not recommend any other block ciphers besides AES.

Related-Key Attacks and AES Related-key attacks assume that the attacker has access to encryptions or decryptions of known or chosen plaintexts or ciphertexts under different keys that have a relationship to each other that is known to the attacker (for example, differ in exactly one bit position of the key). Certain attacks of this kind against round-reduced versions of AES-256 [14] and against unmodified versions of AES-192 as well as AES-256 [15] represent the only known cryptanalytic techniques so far against which AES shows a significantly worse behaviour than an ideal cipher with corresponding key length and block size.

At this point in time, these findings on the security of AES under specific types of related-key attacks have no impact on the recommendations made in this Technical Guideline. In particular, a related-key boomerang attack on AES-256 from [15] with computation time and data complexity of $2^{99.5}$ is not considered to violate the medium-term security level of 120 bits targeted in this Technical Guideline due to the technical prerequisites of related-key boomerang attacks. The best known attacks against AES that do not require related-keys achieve only a slight advantage over generic attacks [19].

3.1.1. Modes of Operation

As mentioned in Section 3.1, a block cipher only provides a mechanism for encrypting plaintexts of a single fixed length. To encrypt plaintexts of arbitrary length, an encryption scheme for plaintexts of (approximately) arbitrary length must be constructed from the block cipher using an appropriate *mode of operation*. Another effect of a cryptographically strong mode of operation is that the resulting encryption scheme will in some respects be stronger than the underlying block cipher, for example if the mode of operation randomises the encryption process, making it difficult to recognise multiple encryptions of the same plaintexts.

Various modes of operation for block ciphers can initially only handle plaintexts whose length is a multiple of the block size. In this case, the last block of a given plaintext may be too short and must be padded accordingly. Formatting by filling this last block to the required block size is also called *padding*. In Section 3.1.3 suitable padding mechanisms are presented. However, among the recommended modes of operation for block ciphers, only the CBC mode requires a padding step.

The simplest way to encrypt a plaintext whose length is already a multiple of the block size is to encrypt each plaintext block with the same key; this mode of operation is also called the Electronic Code Book (ECB). However, the use of the ECB mode of operation leads to the fact that the same plaintext blocks are encrypted into the same ciphertext blocks. The ciphertext thus at least provides information about the structure of the plaintext and, if the entropy per block of the plaintext is low, it may be possible to reconstruct parts of the plaintext by frequency analysis. For this reason, the n-th cipher block should not only depend on the n-th plaintext block and the key used but also on an additional value, such as the $(n-1)$-th ciphertext block or a counter.

This is the case for the following recommended modes of operation, which are adequate for the block ciphers listed in Table 3.1:

• Counter with Cipher Block Chaining Message Authentication (CCM), see [85],
• Galois/Counter Mode (GCM), see [86],
• Cipher Block Chaining (CBC), see [83], and
• Counter Mode (CTR), see [83].

Table 3.2: Recommended modes of operation for block ciphers.

Remark 3.1 Both GCM mode and CCM mode provide cryptographically secure data authentication in addition to encryption if the tag length is sufficient. For the other two modes of operation, it is generally recommended to provide separate mechanisms for data authentication in the overall system. Ideally, no decryption or other further processing should take place for unauthenticated encrypted data. If unauthenticated encrypted data is decrypted and further processed, then there are increased residual risks with regard to the exploitation of error oracles, see for example [108].

3.1.2. Conditions of Use

The modes of operation listed in Section 3.1.1 require initialisation vectors and furthermore certain other conditions must be met for secure operation, which are summarised below:

For CCM:

- The length of the authentication tag must be chosen appropriately. For general cryptographic applications, a tag length of ≥ 96 bits is recommended. In general, attackers can modify cipher rate or authenticated data undetected with a success probability of $\approx 2^{-t}$ per attempt when using tag length t in CCM mode. When using tag lengths lower than those recommended here, the associated residual risks must be carefully examined by an expert.

For GCM:

- For GCM initialisation vectors, a bit length of 96 bits is recommended in [86]. This recommendation is followed by the present Technical Guideline, in particular with reference to the results from [67].[1] In [86], it is required that the probability of repetition of initialisation vectors under a given key should be $\leq 2^{-32}$. This implies a key change after at most 2^{32} calls of the authenticated encryption function. If the initialisation vectors are generated deterministically, it must be demonstrated that a repetition of initialisation vectors over the entire lifetime of a key is not possible.
- For general cryptographic applications, GCM with a length of the GCM tags of at least 96 bits should be used. For special applications, shorter tags can be used after consultation with experts. In this case, the guidelines on the number of allowed calls to the authentication function with a common key from [86] must be strictly adhered to.

For CCM, GCM and CTR mode:

- Initialisation vectors must not repeat within the lifetime of a key. More precisely, no two AES encryptions (that means applications of the underlying AES block cipher) with the same input values (key, message) must ever be performed. Failure to comply with this condition

[1] In [67], errors in previously accepted security proofs on Galois/Counter mode are pointed out and a corrected analysis of the security of GCM is presented. In this corrected analysis, an IV length of exactly 96 bits turned out to be advantageous.

will result in a potentially complete loss of confidentiality for the affected plaintext blocks, in the case of the GCM, additionally also of integrity.[2]

For CBC:

- Only unpredictable initialisation vectors are to be used.

To generate unpredictable initialisation vectors, various methods are recommended in Section B.2. For applications where the initialisation vector requirements given here cannot be met, it is strongly advised to consult an expert.

3.1.3. Padding Schemes

As already explained in Section 3.1.1, the CBC mode requires an additional padding step: it may happen during the partitioning of a plaintext to be encrypted that the last plaintext block is smaller than the block size of the cipher used.

The following padding schemes are recommended in this Technical Guideline:

- ISO-Padding, see [61, Padding Method 2] and [83, Appendix A],
- Padding according to[57, Section 6.3],
- ESP-Padding, see [69, Section 2.4].

Table 3.3: Recommended padding schemes for block ciphers.

Remark 3.2 In CBC mode of operation, care must be taken to ensure that an attacker cannot learn from error messages or other side-channels whether the padding of an introduced data packet was correct [108]. More generally, in encryption schemes where an attacker can make changes to the ciphertext in such a way as to result in controlled changes to the plaintext, no side-channel information must be available to tell whether a given ciphertext corresponds to a valid plaintext or whether it is of invalid format.

3.2. Stream Ciphers

In case of stream ciphers, a key stream is first generated from a key and an initialisation vector, that is, a pseudo-random sequence of bits that is then XOR-added bitwise to the message to be encrypted. At the moment, no dedicated stream ciphers are recommended, but AES in counter mode (AES-CTR) can be considered a stream cipher. If a stream cipher is used, it is strongly recommended to protect the integrity of the transmitted information by separate cryptographic mechanisms. An attacker can make bit-precise changes to the plaintext in the absence of such mechanisms.

3.3. Symmetric Key Agreement Schemes, Key Transport and Key Update

In addition to key agreement schemes, key transport schemes are also of practical importance. In a key transport scheme, secret key data is generated by one party and transported securely to one or more recipients. The generating entity can be a trusted third party or one of the parties involved

[2] If the repetition of a nonce cannot be precluded, it may be advisable to use the AES-GCM-SIV mode [55], in which confidentiality and integrity of a message are ensured even in the case of a nonce repetition.

in the communication. In the latter case, it is recommended that all parties involved only use self-generated keys for the transmission of their own sensitive data. At this point, the recipients have no control over the distributed session keys.

Finally, this chapter also deals with key update schemes. Here, two parties already share a common secret and derive a new key from it at the end of a key change period. This can be achieved either by deriving new session keys from a permanent master key or by an update procedure that generates a new key from the current key and possibly other data. Various factors must be taken into account when determining the lifetime of key material, among them the type of key, the environment of use or the sensitivity of the data to be protected. Further information on this topic can be found, for example, in [96].

Remark 3.3 (Asymmetric versus Symmetric Key Agreement Schemes) Asymmetric key agreement schemes can be used to achieve security properties that cannot be realised using symmetric cryptography alone. For example, both recommended classical asymmetric key agreement schemes have the property that an adversary who knows all the long-term secrets[3] (if any) of the two parties involved in the communication still cannot determine the key negotiated during an uncompromised protocol execution if he cannot efficiently solve the mathematical problem underlying the asymmetric mechanism used (here: the Diffie-Hellman problem). In comparison, in symmetric key agreement schemes, at most the security objective *post-compromise security* can be achieved, that means an attacker who knows all the long-term secrets of the two parties involved cannot determine the results of *previous* properly performed key agreements.[4]

Key Transport In general, all of the previously recommended symmetric encryption schemes can be used for the transport of session keys. It is recommended to combine an encryption scheme from Section 3.1 3 with a MAC from Section 5.2 (in Encrypt-then-MAC mode) to ensure a manipulation-resistant transmission of the key material.

Key Agreement Key agreement schemes, too, can be realised solely on the basis of symmetric schemes provided that the existence of a common long-term secret can be assumed. Key Establishment Mechanism 5 from [62] represents a suitable scheme. If an implicit key confirmation by possession of the same session keys is not sufficient for the given cryptographic application, it is recommended to extend this protocol by a further key confirmation step. As key derivation function, the mechanism recommended in Section B.1 should be used.

Key Update In some situations it may be necessary to synchronously exchange the keys used in a cryptographic system for all parties involved without a new key exchange or further communication. In this case, key update mechanisms can be used. Assuming that the master key K_t of a cryptosystem is to be replaced at time t via such a mechanism, we recommend to define

$$K_{t+1} \coloneqq \mathrm{KDF}(s, \text{Label}, \text{Context}, L, K_t).$$

Here, KDF denotes a two-step cryptographic key derivation function according to [95, Section 5], and s is the salt value used in the expansion step. The parameters Label and Context enter the expansion step provided in [95] according to [97]. Here, Label is a string that identifies the function of the key to be derived and Context contains information about the further protocol context. The parameter L denotes the length of the key K_{t+1} to be derived and also enters into the expansion step.

[3] Here, by long-term secret we refer to the long-term secrets that have to be used to secure the connection against man-in-the-middle attacks.

[4] *Merkle puzzles* are an exception in this respect in that they constitute a public key key agreement scheme using only symmetric primitives [81]. However, this mechanism is only of academic relevance.

In this mechanism, it is absolutely essential to ensure that different derivation parameters are used for any derivation of further key material from K_t than those used for the derivation of K_{t+1}. It is recommended to enforce this by using appropriate label values and furthermore to encode in label or context at least also the cryptoperiod t. As an additional measure, it may further make sense to use a new salt value for each key derivation. It is recommended to securely delete K_t immediately after K_{t+1} has been calculated, as well as all intermediate results of the calculation. For further recommendations on the implementation of these schemes, please refer to [95, 97].

4. Hash Functions

Hash functions $H\colon \{0,1\}^* \to \{0,1\}^n$ play an important role in many cryptographic mechanisms, for example when deriving cryptographic keys or authenticating data. They map a bit string $m \in \{0,1\}^*$ of arbitrary length[1] to a bit string $h \in \{0,1\}^n$ of fixed length $n \in \mathbb{N}$.

Hash functions used in cryptographic mechanisms, must – depending on the application – meet the following three conditions:

One-Way Property: For given $h \in \{0,1\}^n$, it is practically impossible to find a value $m \in \{0,1\}^*$ with $H(m) = h$.

2nd-Preimage-Property: For given $m \in \{0,1\}^*$, it is practically impossible to find a value $m' \in \{0,1\}^* \backslash \{m\}$ with $H(m) = H(m')$.

Collision Resistance: It is practically impossible to find two values $m, m' \in \{0,1\}^*$ with $m \neq m'$ and $H(m) = H(m')$.

A hash function H satisfying all of the above conditions is called *cryptographically strong*.

these three terms can each be described mathematically more precisely by comparing the best known attacks against these properties with optimal generic attacks. The length of the hash output is a security parameter of crucial importance, as it determines the effort of generic attacks. For the minimum security level required in this Technical Guideline of 120 bits, at least the requirement $n \geq 240$ must be imposed on hash function $H\colon \{0,1\}^* \to \{0,1\}^n$ because of the birthday problem. At this point, it is not necessary to distinguish different cases depending on the period of use of a system, since the hash mechanisms recommended in this Technical Guideline all already have a digest length of ≥ 256 bits.

Remark 4.1 There are cryptographic applications of hash functions in which not all three specified properties of a cryptographically strong hash function are required. Conversely, there are other relevant cryptographic requirements for hash functions that do not follow from the three stated properties. One example is the property of *Zero Finder Resistance* (resistance to the search for preimages of the hash value zero, [22]), which is relevant in the context of ECDSA signatures. All of the hash functions recommended in this Technical Guideline have no known cryptographic weaknesses that are of relevance to the recommended cryptographic mechanisms in which they are used.

The development of fault-tolerant quantum computers has a much less severe impact on the security of hash functions than on the security of asymmetric mechanisms. The use of Grover's algorithm [54] could theoretically accelerate the search of preimages quadratically. Furthermore, quantum algorithms are theoretically able to find collisions of a hash function with an output length of n bits within $2^{\frac{n}{3}}$ calls of the hash function (see [21, 110]). Whether an acceleration compared to a classic search of preimages or collisions can also be achieved in practice is the subject of current research and has not been definitively answered (see e.g. [68]). Nevertheless, especially for applications with high or long-term protection requirements or long-living systems it is advisable to use the hash functions recommended below with an output length of at least 384 bits.

According to current knowledge, the following hash functions are considered to be cryptographically strong and are therefore applicable for all mechanisms mentioned in this Technical Guideline:

[1] Specifications of real hash functions usually include a length restriction, but this is so high that it is not exceeded by real input strings.

- SHA-256, SHA-512/256, SHA-384 and SHA-512, see [89].
- SHA3-256, SHA3-384, SHA3-512, see [90].

Table 4.1: Recommended hash functions.

Remark 4.2 (i) For the hash function SHA1, examples of hash collisions were first published in [107]. Owing to significant cryptanalytic progress [74, 75], the cost of computing such a collision has since been reduced to order-of-magnitude of several thousand euros, and even chosen-prefix collisions are within the reach of academic adversaries. SHA1 should therefore never be used as a secure cryptographic hash function. This does not rule out its use in other cryptographic applications, for example as part of an HMAC construction, but this should also be avoided.

(ii) Even a single collision of a hash function can lead to insecurity in signature algorithms, see for example [77] and [52].

(iii) Both the hash functions of the SHA2 family and those of the SHA3 family are considered cryptographically strong. With regard to classical attacks on collision resistance and one-way properties, as far as is currently known, there is no practically relevant difference between the two families of functions. In other application scenarios, however, there are differences; the functions of the SHA3 family, for example, are resistant to length extension attacks, see also [9].

5. Data Authentication

In the context of this Technical Guideline, data authentication refers to cryptographic mechanisms that ensure that data transmitted or stored have not been modified by unauthorised persons and/or applications. For this purpose a prover (usually the sender of the data) uses a cryptographic key to calculate a tag of the data to be authenticated. A verifier (usually the recipient of the data) of the data) then verifies whether the received tag of the data to be authenticated corresponds to the one he would have expected if the data were authentic and the correct key was used.

A distinction is made between symmetric and asymmetric schemes for data authentication. In symmetric schemes, the prover and the verifier use the same cryptographic key. In this case, a third party cannot check who calculated the tag or whether it was calculated correctly. For asymmetric schemes, the private key is used for the calculation of the tag and verified with the associated public key. This is usually implemented by digital signatures (see Section 5.3).

When symmetric methods for data authentication are used, the verifier of a message can in principle also generate forged messages. Thus, such mechanisms are only suitable if the additional risk of compromise resulting from the distribution of the symmetric key and its availability to (at least) two parties is acceptable. In addition, it must be uncritical if the verifying party forges a message. If one of these conditions is not met, symmetric data authentication methods are unsuitable and digital signatures must be used. In scenarios where these properties are unproblematic, the use of symmetric methods is more efficient. A standard scenario in which the use of symmetric methods for data authentication offers itself is the integrity-secured transport of encrypted data over a network after negotiation of ephemeral keys.

5.1. Security Objectives

When using cryptographic mechanisms for data authentication, a clarification of the security objectives to be achieved in the respective scenario is crucial for the selection of the mechanisms. Roughly speaking, the following scenarios can be distinguished, which are important in many applications:

- Ensuring the integrity of data transmitted over a network on the way from the sender to the receiver: In this use case, the sender and receiver usually have a common secret, and the receiver has no interest in producing forged transmissions. The use of a symmetric methods for data authentication is therefore natural.

- Ensuring the non-repudiation of a message: Here it should be ensured that the owner of a particular key can be reliably identified as the originator of a message and that the author himself cannot create a signed message in such a way that doubts can subsequently arise about the validity of the signature. In such situations, the verifier of a message must not have the corresponding signature key, so in this case only the use of digital signatures is possible. In addition, depending on the specific scenario and the sought level of protection, the private signature key may also have to be protected from inspection by the signature provider himself. This is the case, for example, if there is a risk that the signer might subsequently invalidate past signatures by deliberately distributing his own private key. In addition, it must be ensured that the message is displayed to the recipient in the same way as to the creator, and that any unsigned parts (for example, the unsigned subject line in the case of a signed e-mail) are unambiguously identifiable as such for the recipient as well as for the creator.

- Protection of an asymmetric key exchange against man-in-the-middle attacks: In this use case, no common secret is available, so that integrity-protected transmission of the key exchange messages must be ensured by means of digital signatures.

Remark 5.1 In some application scenarios, there may be special requirements for the security functions involved. For example, a code signature pursues the security objective of the integrity of the transmitted application as well as the non-repudiation of possibly contained malicious functionality in the delivered software, although the signed data can usually neither be meaningfully displayed to the recipient or to the originator nor can their content be checked with reasonable effort. The security functionality of a secure displaying component on the originator's side thus transfers completely to the originator's quality assurance processes and to the security of the technical components used by him.

Remark 5.2 When processing authenticated data, only those data components that have actually been signed must be considered to have data integrity. Enforcing this principle is not always trivial, partly because cases critical to an application may never appear in legitimately signed data. Especially when using more complex signature formats (for example XML signatures) or in contexts where security objectives are to be enforced by digital signatures that were not foreseen during the development of the components used, it should therefore always be thoroughly checked by an expert whether additional safeguards may be required.

Remark 5.3 The authenticity of signed data may still not be sufficiently confirmed by a signature, for example if replay attacks are possible. Such attacks must be prevented by additional measures. In general, this can be achieved by a suitable combination of data authentication schemes with methods for performing challenge-response-based instance authentication. In some situations (for example, software or key updates), checking version counters or timestamps covered by the signature may also be sufficient.

5.2. Message Authentication Code (MAC)

Message authentication codes are symmetric methods for data authentication, usually based on block ciphers or hash functions. They are used when large amounts of data are to be authenticated or when the verification or creation of tags must be particularly efficient for other reasons. The prerequisite in this case is that the proving and the verifying party have agreed on a common symmetric key in advance. Frequently, both the confidentiality as well as the authenticity of the data have to be ensured. Such mechanisms are discussed in Section A.1. In Chapter 2, methods which allow the exchange of keys over insecure channels are presented.

The development of fault-tolerant quantum computers has a much less severe impact on the security of symmetric mechanisms than on the security of asymmetric mechanisms. The use of Grover's algorithm [54] could theoretically accelerate the search of the key space of symmetric mechanisms quadratically. Whether an acceleration compared to a classic exhaustive search of the key space can also be achieved in practice is the subject of current research and has not been definitively answered (see e.g. [68]). Nevertheless, especially for applications with high or long-term protection requirements or long-living systems it is advisable to use a key length of 256 bits for the message authentication codes methods recommended below.

In principle, the following schemes are considered secure if, for the CMAC and GMAC scheme, one of the block ciphers listed in Table 3.1 is used, and for the HMAC scheme one of the hash functions listed in Table 4.1 is used. Furthermore, the length of the key for all schemes should at least be 16 bytes:

- CMAC, see [92],
- HMAC, see [10],
- KMAC128, KMAC256, see [91],
- GMAC, see [86].

Table 5.1: Recommended MAC schemes.

When using these schemes, the following aspects must be observed:

- CMAC, HMAC and KMAC are pseudorandom functions, whereas GMAC is not. Furthermore, compared to the other two methods, GMAC requires a 96-bit nonce as initialisation vector.
- A tag length of ≥ 96 bits is recommended for general cryptographic applications in all of the above schemes. Shorter tag lengths may only be used after experts have weighed up all the circumstances affecting the application in question. For GMAC tags, there are forgery attacks with a success probability of $2^{-t+\log_2(n)}$ per attempt known, where t denotes the tag length and n is the number of blocks of the message. and This probability increases further upon detection of successful forgeries [50]. This means that GMAC (and thus also the authenticated encryption mode GCM) provides weaker integrity protection for the same tag length than is expected for CMAC or HMAC with the respective block ciphers or hash functions recommended in this Technical Guideline. The practical relevance of these attacks increases considerably when short authentication tags (< 96 bits) are used. The use of short tags with GMAC/GCM is therefore strongly discouraged.
- When using the HMAC scheme, the tag length should be truncated to no more than half the output length of the hash function.
- With regard to the GMAC scheme, the other remarks on the operating conditions for GCM from Section 3.1.2 apply accordingly as far as the authentication function is concerned.
- The authentication keys used must be protected in an equally safe manner as other cryptographic secrets in the same context.
- In general, all requirements from [10, 92, 91, 86] must be met in the respective scheme used and their compliance must be documented.

The following table summarises the recommendations on key and checksum length when using MAC mechanisms:

Scheme	CMAC	HMAC	KMAC128	KMAC256	GMAC
Key length	≥ 128	≥ 128	≥ 128	≥ 256	≥ 128
Recommended tag length	≥ 96	≥ 128	≥ 96	≥ 96	≥ 96

Table 5.2: Parameters for recommended MAC schemes.

5.3. Signature Algorithms

In signature algorithms, the data to be signed is first hashed before the tag and/or the signature is calculated with the private key of the proving party from this hash value. The verifier then checks the signature with the corresponding public key. As with asymmetric encryption schemes, it must not be possible to calculate the signature without knowledge of the private key. This implies in particular that it must not be practically possible to derive the private key from the public key.

To distribute the public keys to the verifiers, usually a public key infrastructure is used. In any case, a reliable way (protected against manipulation) to distribute the public keys is essential, as with all public key schemes. However, an in-depth discussion of the technical and organisational options for solving this problem exceeds the scope of this Technical Guideline, so the topic is only considered in the margins.

For the specification of signature algorithms, the following algorithms are to be specified:

- An algorithm for the generation of key pairs.
- A hash function that maps the data to be signed to a data block of fixed bit length.
- An algorithm for the signing the hashed data and an algorithm for the verification of the signature.

Basically all of the hash functions listed in Table 4.1 are suitable for the calculation of the hash value, so it remains to specify the algorithms and key lengths listed under the first and third point, respectively. In addition, we give recommendations for minimum key lengths.

Table 5.3 provides an overview of the signature methods recommended in the following. All recommended mechanisms can be used for signing data as well as for issuing certificates.

• RSA, see [60]
• DSA[1], see [63],
• DSA variants on elliptic curves:
– ECDSA, see [37],
– ECKDSA/ECKCDSA, ECGDSA, see [37, 63], and
• Merkle signatures[2], more precisely XMSS or LMS and their multi-tree variants according to [94].

Table 5.3: Recommended signature algorithms.

[1] The use of the DSA signature algorithm (see Section 5.3.2) is in this Technical Guideline only recommended until 2029 due to its low prevalence and the discontinuation in [98].

[2] Merkle signatures differ in essential points from the other signature algorithms recommended here. For a more detailed description of the most important aspects, see Section 5.3.4.3

According to the current state of knowledge, with a suitable choice of security parameters, all signature mechanisms recommended here achieve a comparable level of security if the private keys are reliably kept confidential and, in particular, cannot be determined by exploiting implementation weaknesses, such as for instance side channels, fault attacks or mathematical attacks directed against a specific kind of key generation.

Remark 5.4 With the exception of DS 3 (compare Table 5.4) the recommended asymmetric signature algorithms are probabilistic algorithms.[1] Thus, each time a signature is calculated, a new random value is required; further requirements for these random values are specified in the corresponding sections.

Remark 5.5 Due to the threat posed to classical signature algorithms by the ongoing development of sufficiently large quantum computers, a migration to quantum-safe algorithms should take place. This Technical Guideline already recommends stateful hash-based signature algorithms (see Section 5.3.4.3), which are already standardised, but only suitable for special use cases.

As part of its PQC standardisation process, NIST is expected to publish a standard for the SLH-DSA (also known as SPHINCS+) and ML-DSA (also known as CRYSTALS-Dilithium) schemes before the end of 2024. These are stateless hash-based respective lattice-based signature schemes. The BSI intends to include SLH-DSA and ML-DSA with the parameter sets corresponding to NIST Security Strength Categories 3 and 5 in the recommendations of this Technical Guideline after publication of the corresponding standards.

This Technical Guideline recommends the use of a quantum-safe signature scheme only in combination with a classic signature scheme. Hybridisation should be implemented in such a way that the hybrid signature scheme is secure as long as at least one of the schemes is secure. A natural and robust hybridisation is the concatenation of a quantum-safe signature with a classic signature so that the concatenated signature is accepted as valid if all individual signatures are valid. Care should be taken here to generate key material for hybrid signatures specifically for this purpose and not to use it for non-hybrid signatures as well.

Provided that the implementation security of stateful and stateless hash-based procedures is properly considered, hash-based signatures can in principle also be used on their own (i.e. not hybrid).

Remark 5.6 Merkle signatures, unlike all other signature algorithms listed in this Technical Guideline, are considered secure against attacks using quantum computers [24]. Moreover, they are the only scheme mentioned here that is *forward secure* in the sense of [11].

5.3.1. RSA

The security of the RSA scheme is based on the assumed difficulty of calculating e-th roots in $\mathbb{Z}_n$, where n is an integer of unknown factorisation into two prime factors p, q and e is an exponent coprime to $\varphi(n) = (p-1)(q-1)$.

Key Generation The key generation is analogous to that of the RSA encryption scheme, for details see Section 2.3.2. The signature verification key is of the form (n, e), where $n = p \cdot q$ is composite, e is invertible modulo $\varphi(n)$ and $2^{16} < e < 2^{256}$, and the signature key is $d \coloneqq e^{-1} \bmod \varphi(n)$.

Generation and Verification of Signatures For signature generation and/or verification we refer to [60]. Here, the hash value of the message must be padded to the bit length of the module n before the private key d is applied. The padding scheme must be chosen carefully (see for example [27]); the following schemes are recommended:

[1] The RSA algorithm itself is deterministic, but not the here recommended padding procedures to RSA (except DS 3).

- EMSA-PSS, see [82].
- Digital Signature Scheme (DS) 2 and 3, see [65].

Table 5.4: Recommended padding schemes for the RSA signature algorithm.

Key Length The length of the modulus n should be at least 3000 bits.

5.3.2. Digital Signature Algorithm (DSA)

The security of the DSA method is based on the assumed difficulty of calculating discrete logarithms in $\mathbb{F}_p^*$.

Key Generation

1.) Choose two prime numbers p and q such that $q \mid (p-1)$.

2.) Choose $x \in \mathbb{F}_p^*$ and calculate $g := x^{(p-1)/q} \bmod p$.

3.) If $g = 1$, go to 2.).

4.) Randomly choose a number $a \in \{2, \ldots, q-1\}$ and set $A := g^a$.

Then (p, q, g, A) is the public key and a is the private key.

Generation and Verification of Signatures For the signature generation and verification we refer to [63] and [88]. Both signature generation and signature verification require a cryptographic hash function. One of the hash functions recommended in this guideline should be used and the length of the hash values should correspond to the bit length of q. If none of the hash functions recommended in Table 4.1 has a suitable hash length, the k leading bits of the hash output should be used, where k denotes the bit length of q. If the length L_H of the hash value is *shorter* than the bit length of q, the resulting signature algorithm will have a security level of (at most) $L_H/2$ bits.

Key Length The length of the prime number p should be at least 3000 bits.

Remark 5.7 The use of the DSA signature method is in this Technical Guideline only recommended until 2029 due to its low prevalence and the discontinuation in [98].

Remark 5.8 The DSA method is a so-called probabilistic algorithm, since a random number $k \in \{1, \ldots, q-1\}$ is needed to calculate the signature. Here, k should be chosen with respect to the uniform distribution on $\{1, \ldots, q-1\}$, since otherwise attacks exist, compare [100]. Two algorithms for calculating k are presented in Section B.4.

Remark 5.9 Regarding the generation of the system parameters, see Remark 2.10.

5.3.3. DSA Versions based on Elliptic Curves

The security of these mechanisms is based on the assumed difficulty of calculating discrete logarithms in elliptic curves.

Key Generation

1.) Generate cryptographically strong EC system parameters (p, a, b, P, q, i), see Section B.3.

2.) Choose d randomly and uniformly distributed in $\{1, \ldots, q-1\}$.

3.) Set $G := d \cdot P$.

Then the EC system parameters (p, a, b, P, q, i) together with G form the public key and d the private key.

Generation and Verification of Signatures The following algorithms are recommended in this Technical Guideline:

- ECDSA, see [37].
- ECKDSA/ECKCDSA, ECGDSA, see [37, 63].

Table 5.5: Recommended signature algorithms based on elliptic curves.

For the generation and verification of signatures, a cryptographic hash function is required. In general, all hash functions recommended in this Technical Guideline are suitable. The length of the hash values should correspond to the bit length of q. The other remarks on the choice of hash function from Section 5.3.2 apply accordingly.

Key Length All signature algorithms listed in Table 5.5 ensure a security level of n bits if $q \geq 2^{2n}$ holds for the order q of the base point P and it is assumed that the calculation of discrete logarithms on the curves used is not possible more efficiently than with generic mechanisms. It is recommended to choose $q \geq 2^{250}$.

Remark 5.10 Like the DSA scheme, all of the signature algorithms recommended in this section are probabilistic algorithms. Here too, a random value $k \in \{1, \ldots, q-1\}$ must be chosen according to the uniform distribution on $\{1, \ldots, q-1\}$, since otherwise attacks exist, compare [100]. Two methods for calculating k are presented in Section B.4.

5.3.4. Quantum-Safe Signature Schemes

Due to the threat posed to classical signature algorithms by the ongoing development of sufficiently large quantum computers, a migration to quantum-safe algorithms should take place. To this aim, this Technical Guideline recommends quantum-safe signature schemes.

This Technical Guideline recommends the use of a quantum-safe signature scheme only in combination with a classic signature scheme. Hybridisation should be implemented in such a way that the hybrid signature scheme is secure as long as at least one of the schemes is secure. A natural and robust hybridisation is the concatenation of a quantum-safe signature with a classic signature so that the concatenated signature is accepted as valid if all individual signatures are valid. Care should be taken here to generate key material for hybrid signatures specifically for this purpose and not to use it for non-hybrid signatures as well.

Provided that the implementation security of stateful and stateless hash-based procedures is properly considered, hash-based signatures can in principle also be used on their own (i.e. not hybrid).

5.3.4.1. SLH-DSA (SPHINCS+)

As part of its PQC standardisation process, NIST is expected to publish a standard for the SLH-DSA (also known as SPHINCS+) scheme before the end of 2024. This a stateless hash-based lattice-based signature scheme. The BSI intends to include SLH-DSA with the parameter sets corresponding to NIST Security Strength Categories 3 and 5 in the recommendations of this Technical Guideline after publication of the corresponding standard.

5.3.4.2. ML-DSA (CRYSTALS-Dilithium)

As part of its PQC standardisation process, NIST is expected to publish a standard for the ML-DSA (also known as CRYSTALS-Dilithium) scheme before the end of 2024. This is a lattice-based signature scheme. The BSI intends to include ML-DSA with the parameter sets corresponding to NIST Security Strength Categories 3 and 5 in the recommendations of this Technical Guideline after publication of the corresponding standard.

5.3.4.3. Merkle Signatures

In contrast to the signature algorithms described so far, the security of the signature methods described in [94] (XMSS, XMSS^{MT}, LMS und HSS) is based only on the cryptographic strength of a hash function and a family of pseudorandom functions, but not on the assumed difficulty of a mathematical problem (like the determination of a prime factorisation or the calculation of discrete logarithms in selected groups). In particular, no assumptions on the absence of efficient algorithms for these problems from algorithmic number theory are required. According to current knowledge it is therefore generally assumed that, unlike the other signature methods recommended in this Technical Guideline, Merkle signatures are also secure against attacks using quantum computers.[2]

The generally low complexity-theoretic assumptions underlying the security of Merkle signatures make Merkle signatures appear to be a good scheme for the generation of long-term secure signatures. This is also true under the assumption that attacks by quantum computers are not used during the period of time in which the signature is to remain valid.

However, when using the Merkle signatures recommended in this Technical Guideline – unlike in case of the rest of the signature schemes described in this Technical Guideline – only a limited number of messages can be authenticated with a given public key. With the single-tree variants XMSS and LMS, the computing time for generating the public key is proportional to this maximum number of messages that can be authenticated with a key pair and is thus comparatively long. If a large number of messages are to be signed without intermediate generation and authenticated distribution of a new public key, the use of the multi-tree variants XMSS^{MT} and HSS is recommended.

Remark 5.11 In [101], an attack against SLH-DSA (formerly SPHINCS+) with SHA-256 as the underlying hash function is described. This attack reduces the security of the attacked SLH-DSA instance by approximately 40 bits, i.e. instead of the claimed security level of 256 bits, this SLH-DSA instance only offers a security level of approximately 216 bits. Theoretically, the idea of this attack can be transferred to LMS/HSS in combination with SHA-256, so that this attack should be taken into account when using such LMS/HSS instances. However, the security level of 120 bits targeted in this Technical Guideline is achieved by parameter sets that are standardised in [94] even when the attack is accounted for.

5.3.5. Long-Term Preservation of Evidentiary Value for Digital Signatures

If the intended period of time over which the authenticity and integrity of the data to be protected by means of a data authentication system is to remain secure significantly exceeds the prediction

[2] A discussion of quantum security of the collision resistance of hash functions can be found in [12].

period of this Technical Guideline, it is irrespective of the present recommendations on mechanisms and key lengths for digital signatures recommended to take into account the possibility of a future migration of the system to new signature algorithms or longer signature keys already during the development. This should include mechanisms for the signature renewal of old signed documents using the updated schemes. More information on this topic can be found in Technical Guideline TR-03125 (TR-ESOR) [41].

6. Instance Authentication

In this Technical Guideline, instance authentication refers to cryptographic protocols in which a prover confirms to a verifier the possession of a secret. For symmetric mechanisms, this secret is a symmetric key, which has to be exchanged in advance. In case of instance authentication with asymmetric mechanisms, the proving party shows that he is in possession of a private key. A public key infrastructure (PKI) is usually required for this, so that the verifier can assign the corresponding public key to the proving party. Password-based schemes are primarily used to unlock smart cards or other cryptographic components. Here, the owner of the component proves that he is in possession of a password or PIN.

Authentication should – where reasonable and possible – be mutual and can be accompanied by key agreement to ensure the confidentiality and integrity of any subsequent communication, see Chapter 2 for recommended key exchange and key agreement schemes and Section A.2 for recommended protocols that combine both schemes.

In this chapter, for the first two schemes (Sections 6.1 and 6.2), only general ideas about instance authentication are provided and the corresponding cryptographic primitives are recommended. For the required cryptographic protocols it is referred to Section A.2. In particular, among others recommendations for key lengths can be found there.

6.1. Symmetric Schemes

The possession of the secret key is demonstrated by the prover (P) to the verifier (V) by sending a random value r to V. For the scheme to reach the minimum security level aimed at in this Technical Guideline, r should have at least 120 bits of min-entropy. If a large number of authentications are performed with the same secret key, then the probability of a collision of two of these challenge values should be limited to $\leq 2^{-32}$. P uses the shared key K to calculate an authentication code for the message r and sends it back to V, who verifies it. Such schemes are also called *Challenge-Response Methods*, see Table 6.1 for a schematic representation.

Prover (P)		Verifier (V)
		Choose random value r
	$\xleftarrow{r}$ (Challenge)	
Calculate authentication code c		
	$\xrightarrow{c}$ (Response)	
		Verify authentication code

Table 6.1: Schematic representation of a Challenge-Response method for instance authentication.

The calculation and verification of the authentication code depends on the selected scheme. In principle, all encryption schemes recommended in Chapter 3 and all MAC schemes recommended in Section 5.2 can be used. For recommended bit lengths and constraints on the random values used, see Section A.2.

6.2. Asymmetric Schemes

Also for asymmetric mechanisms, challenge-response protocols are used for instance authentication. Here, the prover uses his secret key to calculate a tag for a random value r sent by the verifier. The verifier then verifies the tag with the help of the corresponding public key. In general, all of the schemes recommended in Chapter 2 can be used for this purpose. For recommended bit lengths and constraints on the random values used, see also Section A.2.

Remark 6.1 Even though the signature algorithms recommended in Section 5.3 for data authentication can also be used for instance authentication, it has to be ensured that the used keys are different, meaning that a key used to generate signatures is not used for instance authentication. This must also be indicated in the corresponding certificates for the public keys.

6.3. Password-Based Methods

Passwords for unlocking the cryptographic keys made available on cryptographic components (for example signature cards) are usually short, so that the owner of the component can remember the password. In many situations, the permitted character set is also restricted to the digits 0-9; in this case, one also speaks of PIN instead of password. In order to nevertheless reach an adequate security level, the number of access attempts is usually limited.

6.3.1. Recommended Password Lengths for Access to Cryptographic Hardware Components

The following constraints for password lengths and number of attempts for access to cryptographic hardware components are recommended:

- It is generally recommended to use passwords with an entropy of at least $\log_2(10^6)$ bits. This can be achieved, for example, by assigning ideally random six-digit PINs.
- The number of consecutive unsuccessful access attempts must be tightly limited. With a password entropy of $\log_2(10^6)$ bits, a limit of three attempts is recommended.

Table 6.2: Recommended password lengths and number of access attempts for access protection of cryptographic components.

Remark 6.2 If access passwords for cryptographic components are not (at least approximately) ideally randomly generated by a technical process, but chosen by the user, it is strongly recommended to raise the user's awareness with respect to the choice of secure passwords. Furthermore, it is recommended in this case to refrain from using purely numeric passwords (PINs). For passwords formed over an alphabet containing at least the letters A-Z, a-z, 0-9 and, if applicable, special characters, a length of eight characters is recommended. In addition, it is recommended to take safeguards to exclude passwords that are easy to guess (for example, individual words in the respective national language or an important foreign language and dates in easy guessable formats).

Remark 6.3 In some applications, after consideration of all the circumstances by an expert, the use of passwords with lower entropy than recommended above may also be compliant with this Technical Guideline. However, a single unauthorised access attempt should at least never succeed with

a probability of success greater than $\approx 10^{-4}$. The number of consecutive unsuccessful access attempts must be tightly limited, where the exact limitations depend on the application. The residual risks should be thoroughly documented and it is recommended to inform the authorised user of any unauthorised access attempts, if possible, even if the component was not blocked subsequently.

Remark 6.4 (i) To prevent denial-of-service attacks or accidental blocking of the component, there must be a mechanism to unblock the blocking. The entropy of the *personal unblocking key* (PUK) should be at least 120 bits if offline attacks are possible.

(ii) If no offline attacks on the PUK are possible, it is recommended to use a PUK with a min-entropy of 32 (for example 10 digits) and to irrevocably delete the cryptographic secrets contained in the component after a relatively low number of access attempts (for example 20).

(iii) The general recommendation of at least about 20 bits of entropy for the password used in a password-based authentication scheme applies only to authentication to a security component that does not allow of offline attacks and that can reliably enforce the stated restrictions on the number of access attempts allowed. In other situations where these conditions are not met (for example, when a cryptographic secret is directly derived from the password that provides access to sensitive information), it is recommended to choose passwords via a method that offers at least 120 bits of entropy. For access to data or for authentication of transactions with high protection requirements, single-factor authentication is generally not recommended. Instead, two-factor authentication by means of knowledge (knowing a password) and ownership (of a secure hardware component) is recommended in this situation.

6.3.2. Recommended Method for Password-Based Authentication to Cryptographic Hardware Components

For contact-based chip cards, cryptographic protection of the transmission of the PIN to the chip card can currently be omitted if the card terminal itself can be considered as trustworthy and a physical tapping or manipulation of the information transmitted between reader and card is prevented by suitable measures in the operational environment. However, cryptographic protection (with regard to integrity and confidentiality of the transmitted identification data) is also recommended here. In principle, the same mechanisms as for contactless cards are suitable for contact-based cards as well.

In the case of contactless chip cards, the communication between the card reader and the chip card can be read from a distance. Here, the password for activating the chip cannot simply be sent from the card reader to the chip card.

The following password-based method is recommended for the access protection to contactless chip cards:

PACE: Password Authenticated Connection Establishment, see [36].

Table 6.3: Recommended password-based method for the protection of access to contactless chip cards.

The method recommended in Table 6.3 not only demonstrates to the contactless chip card that the user is in possession of the correct password, but at the same time performs a key agreement method, so that subsequently a confidential and authenticated communication can take place.

Remark 6.5 Also with the method recommended in Table 6.3, the number of attempts must be limited. It is recommended to block the chip card after three unsuccessful attempts. The further remarks from Section 6.3.1 apply accordingly.

7. Secret Sharing

In many cases, cryptographic keys have to be stored over a long period of time. This requires in particular that copies of these keys must be made to prevent the loss of the keys. However, as the number of copies grows, so does the likelihood that the secret to be protected is compromised. Therefore, in this chapter, we give a method that allows dividing a secret, such as a cryptographic key K, into n shared secrets $K_1, \ldots, K_n$ in such a way that any $t \leq n$ of these shared secrets are sufficient to reconstruct the secret, but $t-1$ shared secrets provide no information about K. Another application of this scheme is to use a four-eyes principle or, more generally, a t-out-of-n-eyes principle, for example to distribute the password for a cryptographic component among n different users so that at least t users are required to reconstruct the password.

The secret-sharing algorithm presented here was developed by A. Shamir and is therefore also called the Shamir secret-sharing mechanism, see [104]. We assume in the following that the secret to be shared is a key K of bit length r, that is $K = (k_0, \ldots, k_{r-1}) \in \{0,1\}^r$. To compute the shared secrets for n users such that t users can reconstruct the secret K, we proceed as follows:

1.) Choose a prime $p \geq \max(2^r, n+1)$ and set $a_0 := \sum_{i=0}^{r-1} k_i \cdot 2^i \bmod p$.

2.) Independently choose $t-1$ random values $a_1, \ldots, a_{t-1} \in \{0, 1, \ldots, p-1\}$ according to the uniform distribution on $\{0, 1, \ldots, p-1\}$. The values $a_0, a_1, \ldots, a_{t-1}$ define a random polynomial

$$f(x) = \sum_{j=0}^{t-1} a_j x^j \bmod p$$

over $\mathbb{F}_p$, for which $f(0) = a_0 = \sum_{i=0}^{r-1} k_i \cdot 2^i \bmod p$ holds.

3.) Calculate the values $K_i := f(i)$ for all $i \in \{1, \ldots, n\}$.

Table 7.1: Calculation of the secret shares in Shamir's Secret-Sharing algorithm.

.

Each shared secret K_i is then handed over, along with i, to the i-th user.

Remark 7.1 The basis for the algorithm mentioned in Table 7.1 is the so-called *Lagrange-interpolation-formula*, which allows to determine the coefficients $a_0, \ldots, a_{t-1}$ of an unknown polynomial f of degree $t-1$ from t points $(x_i, f(x_i))$ as follows:

$$f(x) = \sum_{i=1}^{t} \left[f(x_i) \prod_{\substack{1 \leq j \leq t \\ j \neq i}} \frac{x - x_j}{x_i - x_j} \right] \bmod p.$$

In particular, $a_0 = f(0)$ (and thus K) can be calculated in this way from t given points.

In order to reconstruct the secret K from t shared secrets $K_{j_1}, \ldots, K_{j_t}$ (with pairwise different j_l), one calculates $K = a_0 = \sum_{i=0}^{r-1} k_i \cdot 2^i \bmod p$ as follows:

1.) For all $j \in \{j_1, \ldots, j_t\}$ calculate the value $c_j = \prod_{\substack{1 \le l \le t \\ j_l \neq j}} \frac{j_l}{j_l - j} \bmod p$.

2.) Calculate $K = a_0 = \sum_{l=1}^{t} c_{j_l} K_{j_l} \bmod p$.

Table 7.2: Reassembly of the shared secret in Shamir's secret-sharing algorithm.

Both when dividing into shared secrets in Table 7.1 and when recombining them in Table 7.2, note, that in each case the calculation is carried out in $\mathbb{F}_p$, that means modulo p.

Remark 7.2 The condition $p \geq \max(2^r, n+1)$ ensures that on the one hand the secret can be represented as an element of $\mathbb{F}_p$ and on the other hand that at least n independent partial secrets can be generated. The algorithm achieves information-theoretic security, which means that it is not possible even for an attacker with unlimited resources to reconstruct the distributed secret without learning t shared secrets or a value derived from the knowledge of t shared secrets in a suitable way.

The security of the scheme therefore does not depend on any further security parameters apart from the stated condition. However, organisational and technical measures must be taken to ensure that an attacker cannot gain knowledge of t shared secrets. Any communication about the shared secrets must therefore be encrypted and authenticated, as far as it is physically possible for an attacker to eavesdrop, record or manipulate this communication.

Moreover, information-theoretic security is only given if for $i > 0$, the a_i are chosen truly at random and according to the uniform distribution on $\mathbb{F}_p$. In order to achieve at least complexity theoretical security, a physical random generator of the functionality class PTG.3 or a deterministic random generator of the functionality class DRG.3 or DRG.4 should be used to generate the a_i. The values returned from this random generator must be post-processed to follow the uniform distribution on $\mathbb{F}_p$; suitable methods for this can be found in Section B.4.

8. Random Number Generators

The majority of cryptographic applications require random numbers, for example, to generate cryptographic long-term or ephemeral keys, system parameters or for instance authentication. This applies to symmetric and asymmetric encryption schemes as well as to signature, authentication and padding methods. In general, unsuitable random number generators can substantially weaken strong cryptographic mechanisms, so special care must be taken in cryptographic applications to ensure that suitable random number generators are used. Thus – in contrast to, for example, numerical simulations or experiments, in which reproducibility can play an important role – in the cryptographic context, unpredictability and secrecy of the random numbers and/or the values derived from them are indispensable properties for most applications. Even if an adversary knows long subsequences of random numbers, this should not allow him to determine their predecessors or successors.

Usually, the goal in generating random numbers is to produce output values uniformly distributed on $\{0,1\}^n$. However, in some cases, random numbers with certain other distributions are needed. For this reason, Appendix B contains algorithms that can be used to calculate random values with desired properties (for example, uniformly distributed on $\{0,\ldots,q-1\}$) from the output values of a random number generator.

In the German certification scheme, the AIS 20 [43] (for deterministic random number generators) and AIS 31 [44] (for physical random number generators) are binding. Of central importance is the mathematical-technical annex [42] common to them, which defines functionality classes for physical random number generators (PTG.1 - PTG.3), for deterministic random number generators (DRG.1 - DRG.4) and for non-physical non-deterministic random number generators (NTG.1). Furthermore, [42] explains the mathematical background and illustrates the concepts with numerous examples.

In the following sections, the different types of random number generators are discussed in more detail. The main recommendations for the use of random number generators in general cryptographic applications can be summarised as follows:

- When using a physical random number generator, it is generally recommended to use a PTG.3 generator. This is especially true for the generation of ephemeral keys when computing digital signatures and Diffie-Hellman based key negotiation. In cases where the use of a certified cryptographic component is required for random number generation, this recommendation only applies if correspondingly certified components are available. Otherwise, a PTG.3 generator can usually be constructed by cryptographic post-processing of the output of a PTG.2 generator, implemented in software and compatible with the requirements of the PTG.3 functionality class.

- For some specific cryptographic applications, PTG.2 generators are also sufficient, for example when generating symmetric session keys or when generating a seed for a strong deterministic random number generator. Random numbers from PTG.2-compliant random number generators have high entropy, but may have some biases and/or dependencies. They may be used in some circumstances if the resulting advantage to an adversary is demonstrably small. In general, however, the direct use of PTG.2 random number generators is discouraged. This applies in particular to applications in which the existence of even relatively minor biases in the distribution of the generated random numbers can lead to exploitable weaknesses, such as in the generation of nonces in DSA-like signature procedures.

- When using a deterministic random number generator, it is recommended to use a DRG.3 or a DRG.4 generator whose seed is generated from a physical random source of class PTG.2 or PTG.3. If no such random source is available, the use of a non-physical, non-deterministic random number generator may also be considered in some circumstances. For example, a DRG.3 generator can also be seeded with an NTG.1 generator, for further details please refer to Sections 8.3 and 8.5. In comparison, when using PTG.3 or DRG.4 random number generators, side-channel and fault attacks are also relevant, but only lead to the compromise of relatively few generated random.
- In general, PTG.3 and DRG.4 generators have the advantage of an improved resistance to side-channel and fault attacks compared to PTG.2 and DRG.3 generators. In the case of a PTG.3 generator, the permanent inflow of large amounts of entropy into the internal state means that possible side-channel attacks against cryptographic post-processing are made considerably more difficult, as an adversary can combine information about the internal state at two consecutive points in time t and $t + 1$ only with great difficulty.
- In addition to the risk of long-term compromise through side-channel and fault attacks, DRG.3 random number generators have an increased residual risk of long-term conceivable cryptanalytic compromise compared to DRG.4 and PTG.3 generators if the random number generator produces large amounts of key material worthy of long-term protection from a single seed value.
- In the general case, a system security of n bits requires a min-entropy of the DRNG-seed of n bits.
- For both physical and deterministic random number generators, resistance to high attack potential should be demonstrated in the respective application context.

8.1. Physical Random Number Generators

Physical random number generators use dedicated hardware (usually an electronic circuit), to generate „true“ randomness, that means unpredictable random numbers. This is usually done by exploiting the unpredictable behaviour of simple electrical circuits, as can be caused by various forms of noise in the circuits. In the end, the entropy of the signal is usually physically based on quantum effects or on the amplification of environmental influences in a chaotic system, where the influences cannot be controlled or measured separately. An adversary should only have a negligible (ideally no) advantage over blindly guessing the random numbers even with knowledge of partial sequences of random numbers or, more explicitly, the exact knowledge of the random number generator including the physical environmental conditions at the time of the generation of previous or subsequent random numbers. Deterministic post-processing of the „raw noise data“ (usually digitised noise signals) is often necessary to eliminate any biases or dependencies that may be present.

When using a physical random number generator, it is generally recommended to use a PTG.3 generator in the sense of AIS 31 (compare [42, Chapter 4]). This applies in particular to applications in which an adversary can, at least in principle, combine information about different random numbers. If an implementation of the random number generator in a certified cryptographic component is required, the recommendation to use a PTG.3 generator only applies if suitable certified components exist.

It is possible to construct a PTG.3 generator from a PTG.2 generator by cryptographically post-processing the output of the PTG.2 generator in a suitable manner. This post-processing can usually be implemented in software. The exact requirements for post-processing can be found in [42]. Roughly speaking, the post-processing must implement a DRG.3-compatible deterministic random number generator and at least as much new entropy must always be added to the internal state of

this random number generator by a random number generator of class PTG.2 as is requested by the cryptographic application.

In short, PTG.2 or PTG.3 compliant random number generators must fulfil the following properties:

- The statistical properties of the random numbers can be described sufficiently well by a stochastic model. Based on this stochastic model, the entropy of the random numbers can be reliably estimated.
- The average entropy increase per random bit is above a given minimum limit (close to 1).
- The digitised noise signals are subjected to statistical tests online, which are suitable to detect unacceptable statistical defects or degradation of statistical properties within a reasonable period of time.
- A total failure of the noise source is de facto detected immediately. No random numbers generated after a total failure of the noise source must be output.
- The detection of a total failure of the noise source or unacceptable statistical defects of the random numbers leads to a noise alarm. A noise alarm is followed by a defined, appropriate response (for example, shutting down the noise source).
- (Only PTG.3-compliant random number generators): A (possibly additional) strong cryptographic post-processing ensures that even in the case of an unnoticed total failure of the noise source, the security level of a DRG.3-compliant deterministic random number generator is still assured.

Hybrid random number generators combine security properties of deterministic and physical random number generators. In addition to a strong noise source, hybrid physical random number generators of functionality class PTG.3 have a strong cryptographic post-processing with memory. This is typically realised by a cryptographic post-processing of the random numbers of a PTG.2-compliant random number generator in an appropriate manner.

The development and security-critical assessment of physical random number generators require comprehensive experience in this field and it is recommended to seek advice of experts at an early stage.

8.2. Deterministic Random Number Generators

Deterministic random number generators (also known as pseudorandom number generators) can compute a pseudorandom bit sequence of practically any length from a fixed-length random value called a *seed*. Publicly known parameters can also be included in the computation. For this purpose, the inner state of the pseudorandom number generator is first initialised with the seed. In each step, the internal state is renewed, and a random number (usually a bit sequence of fixed length) is derived from the internal state and output. Hybrid deterministic random number generators refresh the inner state from time to time with „true“ random values (reseed/seed update). This can be initiated in different ways, for example regularly or on request of the application. The inner state of a deterministic random number generator must be reliably protected against readout and manipulation. If a deterministic random number generator is used, then it is recommended to use a DRG.3 or DRG.4-compliant random number generator against the attack potential „high“ in the sense of AIS 20 (see [42]).

When using random number generators of the functionality class DRG.3, a steady inflow of fresh entropy into the internal state is desirable, even if it is not sufficiently regular or of high enough quality to achieve DRG.4-compliance for the overall design. Roughly speaking, DRG.3 conformity means:

- It is practically impossible for an adversary to calculate predecessors or successors of random numbers to a known subsequence of random numbers or to guess them with significantly higher probability than would be possible without knowledge of this subsequence.

- It is practically impossible for an adversary to calculate previously output random numbers based on knowledge of an internal state or to guess them with significantly higher probability than would be possible without knowledge of the internal state.

For DRG.4-compliance, even if an adversary knows the current internal state, it is not practically possible for them to compute random numbers that are generated after the next reseed/seed update or to guess them with significantly higher probability than would be possible without knowledge of the inner state.[1] DRG.4-compliant generators also have certain advantages over DRG.3-compliant random number generators in terms of implementation attacks.

8.3. Non-Physical Non-Deterministic Random Number Generators

For many cryptographic applications, for example in the field of e-business or e-government, neither a physical nor a deterministic random number generator is available, since these applications are generally run on computers without certified cryptographic hardware. Instead, non-physical non-deterministic random number generators (NPTRNG) are generally used in these situations, either directly or to seed a strong deterministic random number generator. A well-known example of a non-physical non-deterministic random number generator is the Linux RNG (device file `/dev/random`), which is analysed in detail in the study [5].

Like physical random number generators, NPTRNGs generate „true“ random numbers and rely on security in the information-theoretic sense through sufficient entropy. However, they do not use dedicated hardware for this, but system resources (system time, RAM contents, etc.) and/or user interaction (for example, keystrokes or mouse movements). NPTRNGs are usually used on systems that are not specifically designed for cryptographic applications, for example, commercially available PCs, laptops or smartphones.

A typical approach for generating random numbers using NPTRNGs is as follows: First, a long bit string of „random data“ (more precisely: of non-deterministic data) is generated, where the entropy per bit is usually rather low. This bit string is mixed with an internal state and random numbers are then calculated from the internal state and output afterwards.

In the mathematical-technical annex [42], a functionality class for such random number generators (NTG.1) is defined. For NTG.1 random number generators, it is roughly speaking required that the amount of entropy collected during operation can be reliably estimated and that the output data have a Shannon entropy of > 0.997 bits per output bit.

This means, among other things:

- The entropy of the internal state is estimated. If a random number is output, the entropy counter is reduced accordingly.

- Random numbers may only be output if the value of the entropy counter is high enough.

- It is practically impossible for an adversary to calculate previously output random numbers based on the knowledge of the internal state and the random bit strings previously used for seed updates or to guess them with significantly higher probability than would be possible without knowledge of the state and bit strings.

[1] Significantly higher probability here refers to a probability at least higher than the probability of guessing the true random values generated for the seed update. For each seed update, at least 120 bits of min-entropy must be generated.

It is crucial for NPTRNG that the entropy sources used by the random number generator cannot be manipulated by an adversary in the sense of entropy reduction or become predictable if the adversary has precise information about the execution environment. This requirement is not a matter of course even when using a principally good NPTRNG. An example of a critical situation in this respect is the use of virtualisation solutions [103]. In this situation, the output of an NPTRNG can, in extreme cases, be completely predicted if the system is started twice from the same system image and all entropy sources of the virtual system are controlled by the host computer.

If an NPTRNG is to be used as the sole or most important random source for a system intended to process sensitive data, it is strongly recommended to always consult an expert.

8.4. Various Aspects

Hybrid random number generators combine security properties of deterministic and physical random number generators. The security of a hybrid deterministic random number generator of functionality class DRG.4 is primarily based on the complexity of the deterministic part, which belongs to class DRG.3. During the use of the random number generator, new randomness is added again and again. This can be done, for example, at regular intervals or at the request of an application.

In addition to a strong noise source, hybrid physical random number generators of functionality class PTG.3 have a strong cryptographic post-processing with memory. Compared to PTG.2-compliant random number generators, the PTG.3 functionality class also offers the advantage that the random numbers have neither biases nor exploitable dependencies. Especially for applications in which a potential adversary can, at least in principle, combine information about many random numbers (for example, ephemeral keys), a physical random number generator of functionality class PTG.3 should be used.

The derivation of signature keys, ephemeral keys and prime numbers (for RSA) or the like from the generated random numbers has to be done with suitable algorithms (for elliptic curves compare [46]). Roughly speaking, a potential adversary should have as little information as possible about the derived values (to be kept secret). Ideally, all values within the respective permissible range of values occur with the same probability, and different random numbers should at least have no practically exploitable correlations. As explained in [42], the generation of secret signature keys, ephemeral keys and prime numbers can also be the target of side-channel attacks, just like signature algorithms (see for example [52, 45, 46]).

8.5. Seed Generation for Deterministic Random Number Generators

For the initialisation of a deterministic random number generator, a seed with sufficiently high entropy is required. This seed should be generated with a physical random number generator of the functionality classes PTG.2 or PTG.3. On PCs, a physical random number generator is usually not available, or at least such a random number generator has not been subjected to thorough manufacturer-independent certification. In such cases, the use of a non-physical non-deterministic random number generator is recommended; NTG.1-compliant random number generators (high attack potential) are suitable for this purpose. Currently, there are no NTG.1-certified random number generators. Therefore, we give below suitable methods for seed generation for the two most important PC operating systems.

However, the use of the methods for seed generation recommended in the following two subsections can only be viewed as secure if the computer is under the user's complete control and no third-party components have direct access to the entire internal state of the computer, as may be the case, for example, if the entire operating system runs in a virtual environment. This means for example in particular that the existence of viruses or Trojan horses on this computer can be ruled out. Users must be informed about these risks.

8.5.1. GNU/Linux

The following mechanism is recommended for seed generation under the GNU/Linux operating system:

Reading of data from the device file `/dev/random`.

Table 8.1: Recommended method for seed generation under GNU/Linux.

Remark 8.1 The randomness provided by the device file `/dev/random` has so far only been reviewed by the BSI for certain kernel versions and found to be suitable when used in PC-like systems. The Linux RNG is thereby assessed by the present Technical Guideline as generally suitable for general cryptographic applications if the requirements of the functionality class DRG.3 or NTG.1 according to [42] are fulfilled. However, since the underlying mechanisms differ considerably depending on the kernel version used and the available random sources depend on the exact system environment, an expert should always be consulted if `/dev/random` is to be used as the main random source in a new system to be developed. A cryptographic evaluation of `/dev/random` in different Linux kernel versions can be found in the BSI study [5]. A cross-check of the security properties of the Linux random number generator in different Linux kernel versions with the functionality classes of AIS 20 and AIS 31 respectively is given in the kernel overviews included there.

Remark 8.2 The use of `/dev/urandom` can be problematic [56], as it does not check whether a sufficient amount of system data has been collected for cryptographic purposes at initialisation of the random number generator. On the other hand, `/dev/random` blocks in some kernel versions if the internal entropy estimator falls below a specified bound. This can slow down random number generation a lot, leading to usability problems.

Remark 8.3 In principle, `/dev/random` can be used not only to seed a pseudorandom generator, but also to generate cryptographic keys directly.

8.5.2. Windows

In contrast to GNU/Linux operating systems, there is currently no function for the operating systems of the Windows family that guarantees adequate high entropy and has been examined by the BSI. For the generation of secure seeds, several sources of entropy should be combined in an appropriate manner. For example, in Windows 10, to generate a seed value of at least 120 bits of entropy, the following method may be considerable:

1.) Reading of 128 bits of random data into a 128-bit buffer S_1 from the Windows API function `BCryptGenRandom()`.

2.) Obtaining a bit string S_2 with at least 120 bits of entropy from a *different source*. Here, for example, the following can be considered:

- Evaluation of the time intervals between successive keystrokes of the user: If these can be demonstrably recorded with a precision of one millisecond, about three bits of entropy per keystroke can be conservatively assumed for this. In order to estimate the temporal resolution of the measured time intervals, the entire processing chain must be examined for entropy-limiting factors. For example, it is possible that the accuracy of the internal clock gives one resolution limit, the polling frequency of the keyboard another, and the time interval within which the used system timers are being updated yet another. It is

recommended to measure the distribution of the keyboard stroke times in practical tests beforehand and to examine them for anomalies. The sequence of the recorded timings of a sufficiently large number of keyboard events can then be encoded into a binary string B. Subsequently, one sets $S_2 \coloneqq \text{SHA256}(B)$ and the data collected in the process (like the recorded data on keyboard stroke times) are cleared from working memory by overwriting with zeros.

- User-initiated events: The times $T_1, T_2, T_3, T_4, T_5, T_6$ of six events initiated by the user are recorded using the Windows API function `QueryPerformanceCounter()`. This usually has an accuracy of at least the order of a microsecond. One can, under the conditions,

  (i) that each T_i cannot be estimated more precisely than to one second even if T_j is known to the adversary for all $j \neq i$,

  (ii) that even if T_j is known to the adversary for all $j \neq i$, the value of T_i cannot be constrained by other considerations (for example, to the polling frequency of the keyboard) to less than 2^{20} possibilities if any interval of one second in length containing T_i is given,

  assume that the bit string $T \coloneqq T_1||T_2||T_3||T_4||T_5||T_6$ contains about 120 bits of entropy from the adversary's point of view. As in the previous example, one sets $S_2 \coloneqq \text{SHA256}(T)$ and clears T from working memory.

  It is not always easy to meet the requirements for independence and unpredictability of user-initiated events. The problem here is that the time at which the software requests the user to initiate an event may be tightly predictable if the timing of an earlier event is known. The time that elapses between the request for an input and the user input itself might also be more accurately predictable than in the range of seconds. The fulfilment of the prerequisites and the plausibility of such entropy estimates must always be examined in the each particular case at hand.

- In a similar way, mouse movements of the user can also be used to gather entropy. The entropy contained in mouse movements cannot be estimated precisely without further analysis. Therefore, a case-by-case analysis is always required, taking into account the type and number of events recorded (pointer positions, and additionally time measurements if necessary), so that it can be ensured that the measurements collected cannot be compressed without loss to a data set of less than 120 bits in size. One then defines S_2 again by a SHA2 hash over the recorded mouse events.

3.) In all cases, a seed value S for a suitable pseudorandom generator can then be derived by setting $S \coloneqq \text{SHA256}(S_1||S_2)$. Ideally, as many independent entropy sources $S_1, \ldots, S_n$ as possible are used to achieve a desired level of security.

Remark 8.4 There is nothing known to the BSI indicating that in the above example a 128-bit value obtained from `BCryptGenRandom()` does not already contain approximately 128 bits of entropy. However, the exact operation of `BCryptGenRandom()` is not described in detail in publicly available vendor documents, nor has the function been intensively studied by parties independent of the vendor, as is the case, for example, for the random number generator integrated into the Linux kernel. Therefore, combining randomness from `BCryptGenRandom()` with output from other entropy sources is recommended as a basic precaution.

Appendix A.

Applications of Cryptographic Mechanisms

The mechanisms explained in the previous chapters often have to be combined in order to ensure the protection of sensitive data. In particular, sensitive data to be transmitted should not only be encrypted but also authenticated in order to enable a recipient to detect any changes.

Moreover, a key agreement must always be accompanied by an instance authentication and an authentication of all messages transmitted during the key agreement, so that both parties can be sure who they are communicating with. Otherwise, the communication can be compromised by a so-called man-in-the-middle attack. Depending on the application, in addition to man-in-the-middle attacks, other types of attacks on the authenticity of message transmission (for example, replay attacks) can also threaten the security of an information-processing system without instance authentication or without data authentication. Therefore, in this chapter adequate mechanisms for both encryption with data authentication and authenticated key agreement are provided.

A.1. Encryption Methods with Data Authentication

In general, all mechanisms recommended in Chapter 3 or Section 5.2 can be used for the combination of encryption and data authentication.

The following two aspects must be observed:

- Encrypted data must always be transmitted authenticated. In addition, it is possible to transmit non-confidential data authenticated but unencrypted. All other data of the same data transmission is *not* authenticated.
- Encryption and authentication keys must be different and should not be derivable from each other.

Remark A.1 For the authenticated transmission of encrypted data, the use of a MAC in Encrypt-then-MAC mode is recommended.

Remark A.2 It is possible to derive encryption and authentication keys from a shared key; this does not contradict the second aspect above. Recommended mechanisms are summarised in Section B.1.

Remark A.3 If the security objective of non-repudiation of the plaintext is also sought in a transmission of encrypted data, the plaintext should be secured by a digital signature. In this case, the plaintext is thus first signed, then encrypted, and finally the encrypted transmission is protected by a MAC against modification on the transmission path. Moreover, a signature on the ciphertext can be reasonable if in addition the encrypted message should be non-repudiable or only the sender (and not the legitimate recipient) should be able to change the ciphertext.

A.2. Key Agreement with Instance Authentication

As already mentioned, a key agreement must always be combined with an instance authentication. After some general preliminary remarks, we provide schemes based either entirely on symmetric algorithms or entirely on asymmetric algorithms.

A.2.1. Preliminary Remarks

Objectives The objective of a key exchange scheme with instance authentication is that the parties involved share a common secret afterwards and are sure with whom they share it at the end of the protocol execution. For the derivation of symmetric keys for encryption and data authentication schemes from this secret, see Section B.1.

Requirements on the Environment Symmetric schemes for an authenticated key exchange always assume the existence of pre-distributed secrets. In the case of asymmetric schemes it is usually assumed that a public key infrastructure (PKI) that is able to reliably bind keys to identities and to authenticate the origin of a key through appropriate certificates exists. Further, it is assumed that the root certificates of the PKI have been made known to all parties involved via reliable channels and that all parties are able to check the validity of all relevant certificates at any time.

For the practical implementation of the schemes presented, the following two conditions must be fulfilled:

- The random values used for the authentication have to be different with high probability each time the protocol is executed. This can be achieved, for example, by choosing each time a random value with respect to the uniform distribution on $\{0,1\}^{120}$.
- The random values used for key agreement must at least reach an entropy that corresponds to the desired key lengths of the keys to be to be agreed upon. It is assumed at this point that only symmetric keys are negotiated. In addition, each party involved in the key agreement should contribute at least 120 bits of min-entropy to the key to be negotiated.

A.2.2. Symmetric Schemes

In principle, any scheme for instance authentication from Section 6.1 can be combined with any scheme for key agreement from Section 3.3. The combination must be done in such a way that the exchanged keys are actually authenticated and thus in particular man-in-the-middle attacks can be excluded. The following mechanism is recommended for this application:

Key Establishment Mechanism 5 from [62].

Table A.1: Recommended Symmetric Scheme for Key Agreement with Instance Authentication.

Remark A.4 Any of the authenticated encryption schemes recommended in this Technical Guideline may be used as encryption methods in Key Establishment Mechanism 5 from [62] (see Section A.1).

A.2.3. Asymmetric Schemes

Analogous to symmetric schemes, any scheme for instance authentication from Section 6.2 can be combined with any mechanism for key agreement from Chapter 2. However, in order to prevent errors in self-designed protocols the key agreement schemes listed in Table A.2 are recommended for key agreement with instance authentication based on asymmetric schemes.

All recommended schemes require the existence of a mechanism for authentic distribution of public keys as a precondition. This mechanism must have the following properties:

- The public key generated by a user must be reliably bound to the user's identity.

- The associated private key should be reliably bound to the identity of the user (it should not be possible for a user to register a public key under his identity to which he cannot use the associated private key).

There are several ways to achieve this. An authentic key distribution can be achieved by means of a PKI. The requirement that the owners of all certificates issued by the PKI should actually be users of the associated private keys can be verified by the PKI performing one of the instance authentication protocols described in Section 6.2 with the applicant using his public key before issuing the certificate.

If the PKI does not perform such a check, it is recommended to add a key confirmation step to the schemes recommended below, in which it is verified that both parties have determined the same shared secret K and in which this secret is bound to the identities of the two parties. For key confirmation, the scheme described in [93, Section 5.6.2] is recommended. The second recommended mechanism (KAS2-bilateral-confirmation according to [93]) already includes this step.

- Elliptic Curve Key Agreement of ElGamal Type (ECKA-EG), see [37],
- Instance Authentication with RSA and Key Agreement with RSA, see KAS2-bilateral-confirmation after [93, section 8.3.3.4],
- MTI(A0), see [66, Annex D.7].

Table A.2: Recommended asymmetric schemes for key agreement with instance authentication.

Remark A.5 In order to comply with the present Technical Guideline, care must be taken in the specific implementation of the protocols that only the cryptographic components recommended in this document are used.

Remark A.6 In the case of the ECKA-EG scheme, no mutual authentication takes place. Here, one party merely proves to the other that it is in possession of a private key, and even this is done only implicitly, through the possession of the negotiated secret after the execution of the protocol.

Appendix B.

Additional Functions and Algorithms

For some of the cryptographic methods recommended in this Technical Guideline, additional functions and algorithms are required, for example, to generate system parameters or to derive symmetric keys from the output obtained by random number generators or key agreement schemes. These functions and algorithms must be carefully chosen in order to achieve the level of security required in this Technical Guideline and to prevent cryptanalytic attacks.

B.1. Key Derivation

B.1.1. Key Derivation after Key Exchange

After a key agreement, both parties hold a shared secret. Im many applications, several symmetric keys, for example for encryption and data authentication, have to be derived from this secret (see also Remark A.2), which can be realised with the help of a key derivation function. In addition, the following objectives can also be achieved by using a key derivation function:

- Binding of key material to protocol data (for example name of the sender, name of the recipient,...) by using the protocol data in the key derivation function.
- Derivation of session keys or keys for different purposes from a master key also in purely symmetric cryptosystems.
- Post-processing of random data to remove statistical biases in cryptographic key generation.

The following methods are recommended for all applications of key derivation functions:

• Two-Step KDF, see [95, Abschnitt 5]
• HKDF, see [71].

Table B.1: Recommended method for key derivation..

In the context of this Technical Guideline, it is in case of Two-Step KDF recommended to use HMAC or AES-CMAC as MAC function, see in Section 5.2. The output of the MAC algorithm must not be shortened. For key lengths of 128 bits, according to [95, Table 4,5] both methods can be used equally, for longer keys only the use of HMAC is recommended. If the key derivation is performed with HMAC, the mechanism recommended here essentially corresponds to HKDF [71]. The only difference to HKDF is the order of the message components, which is not relevant from a cryptographic point of view, but may lead to interoperability problems.

B.1.2. Key Derivation and Hybridisation

In general, the quantum-safe methods recommended in this Technical Guideline are not yet given the same confidence as the established classical methods, as they among others have not been equally well investigated with regard to side-channel resistance or implementation security. This Technical Guideline therefore recommends the use of a quantum-safe mechanism only in combination with a classical key derivation mechanism. Hybrid key agreement should be secure as long as one of the methods used is secure. It is hereby important to consider in detail how the shared secrets are combined with each other and how context-dependent information is included so that the desired property mentionend before is actually achieved.

The following mechanisms are recommended for hybridisation in this Technical Guideline:

CatKDF and CasKDF, see [32].

Table B.2: Recommended hybridisation mechanisms.

B.1.3. Password-Based Key Derivation

In password-based key derivation, a cryptographic key (such as for hard disk encryption) is derived directly from a password entered by a user. When using user-generated passwords, a security level of 120 bits is usually unreachable due to the lack of entropy in human-generated passwords.

In such situations, this Technical Guideline primarily recommends to use a MAC with a secret key used only for this purpose to derive the required secret from the password entered by the user. The MAC should be computed on a cryptographically secure hardware element that is locally available in the system that checks the password. As MAC, a CMAC or HMAC with at least 128 bits key length should be used and the password should be combined with a salt value of at least 32 bits length. If authentication or key derivation fails, the hardware component should implement a delayed response in order to prevent local brute force attacks. In this case, the quality of the passwords must meet the requirements from Section 6.3.1, whereby offline attacks can be considered as inapplicable.

If the use of a a cryptographic hardware token for password-based key derivation is not possible, the hash function `Argon2id` [13] should be used. The security parameters of `Argon2id` and the requirements for the passwords depend on the application scenario and should be discussed with an expert.

B.2. Generation of Unpredictable Initialisation Vectors

As already mentioned in Section 3.1.2, initialisation vectors for symmetric encryption encryption schemes that use the Cipher Block Chaining (CBC) mode of operation. must be unpredictable. This does not mean that the initialisation vectors must be kept confidential, but only that a possible attacker must not be able to practically guess initialisation vectors that will be used in the future. Furthermore, the attacker must not be able to influence the choice of initialisation vectors either.

This Technical Guideline recommends the following two mechanisms for generating unpredictable initialisation vectors, where n is the block size of the block cipher used: with the block cipher and key currently in use and use the ciphertext as an initialisation vector.

Random Initialisation Vectors:	Generation of a random bit sequence of length n using a suitable random number generator (see Chapter 8) and using this bit sequence as an initialisation vector.
Encrypted Initialisation Vectors:	Use of a deterministic mechanism for the generation of pre-initialisation vectors (for example a counter). Encryption of the pre-initialisation vector with the block cipher and key currently in use and use of the ciphertext as the initialisation vector. pre-initialisation vectors (for example a counter). Encryption of the pre-initialisation vector with the block cipher to be used and a key that is different from the actual key (e.g. hash value of the encryption key to be used), and use of the ciphertext as the initialisation vector.

Table B.3: Recommended methods for the generation of unpredictable initialisation vectors.

In the second method, it has to be ensured that the pre-initialisation vectors are not repeated during the lifetime of the system. If a counter is used as a pre-initialisation vector, this means that counter overflows must not occur during the entire lifetime of the system.

B.3. Generation of EC System Parameters

The security of asymmetric mechanisms based on elliptic curves is derived from the assumed difficulty of computing discrete logarithms in these groups.

The following ingredients are needed to define EC system parameters:

1.) Prime number p,

2.) Curve parameters $a, b \in \mathbb{F}_p$ with $4a^3 + 27b^2 \neq 0$, which define an elliptic curve

$$E(\mathbb{F}_p) = \{(x, y) \in \mathbb{F}_p \times \mathbb{F}_p;\ y^2 = x^3 + ax + b\} \cup \{\mathcal{O}_E\},$$

3.) Base point P on $E(\mathbb{F}_p)$.

The *EC system parameters* are then given by the values (p, a, b, P, q, i), where $q := \mathrm{ord}(P)$ denotes the order of the base point P in $E(\mathbb{F}_p)$, $p > 3$, and $i := \mathrm{Card}(E(\mathbb{F}_p))/q$ is the is the so-called *cofactor*.

Not all EC system parameters are suitable for the asymmetric, elliptic curve-based schemes recommended in this Technical Guideline, since for some parameter constellations the discrete logarithm problem is efficiently solvable in the groups generated by these elliptic curves. Besides a sufficient bit length of q, the following conditions must also be fulfilled, see [76] for more information:

- The order $q = \mathrm{ord}(P)$ of the base point P is a prime number different from p.
- $p^r \neq 1 \bmod q$ for all $1 \leq r \leq 10^4$.
- The class number of the maximal order belonging to the quotient field of the endomorphism ring of E is larger than 10^7.

EC system parameters that satisfy the above conditions are also referred to as *cryptographically strong*.

Remark B.1 It is recommended not to generate the EC system parameters on one's own, but instead to use standardised values that are provided by a trustworthy authority.

The system parameters listed in Table B.4 are recommended:

- brainpoolP256r1, see [76],
- brainpoolP320r1, see [76],
- brainpoolP384r1, see [76],
- brainpoolP512r1, see [76].

Table B.4: Recommended EC system parameters for asymmetric schemes that are based on elliptic curves.

B.4. Generation of Random Numbers for Probabilistic Asymmetric Schemes

This Technical Guideline discusses several asymmetric mechanisms that require random numbers $k \in \{0, \ldots, q-1\}$ (for example, as ephemeral keys), where q is usually not a power of 2. Already in the Remarks 2.11, 2.9, 5.8 and 5.10 it was pointed out that k should be chosen to be (at least nearly) uniformly distributed if possible. In contrast, the random number generators presented in Chapter 8 generate uniformly distributed random numbers on $\{0, 1, \ldots, 2^n - 1\}$ („random n-bit strings"). The task is thus to derive (at least nearly) uniformly distributed random numbers on $\{0, 1, \ldots, q-1\}$ from these random numbers.

In Algorithms B.1 and B.2 two methods are presented that achieve this objective, where $n \in \mathbb{N}$ is chosen such that $2^{n-1} \leq q \leq 2^n - 1$ holds, in other words q has bit length n.

Algorithmus B.1: Method 1 for computing random values on $\{0, \ldots, q-1\}$.

Input: $q \in \mathbb{N}$ with $2^{n-1} \leq q \leq 2^n - 1$
Output: $k \in \{0, 1, \ldots, q-1\}$ equally distributed

1: Choose $k \in \{0, 1, \ldots, 2^n - 1\}$ equally distributed.
2: **while** $k \geq q$ **do**
3: Choose $k \in \{0, 1, \ldots, 2^n - 1\}$ equally distributed.
4: **end while**

Algorithmus B.2: Method 2 for computing random values on $\{0, \ldots, q-1\}$.

Input: $q \in \mathbb{N}$ with $2^{n-1} \leq q \leq 2^n - 1$
Output: $k \in \{0, 1, \ldots, q-1\}$ (almost) equally distributed.

1: Choose $k' \in \{0, 1, \ldots, 2^{n+64} - 1\}$ equally distributed.
2: Set $k = k' \bmod q$.

Remark B.2 (i) Method 1 in Algorithm B.1 converts a uniform distribution on $\{0, \ldots, 2^n - 1\}$ into a uniform distribution on $\{0, \ldots, q-1\}$. More precisely, Method 1 yields the conditional distribution on $\{0, \ldots, q-1\} \subset \{0, \ldots, 2^n - 1\}$. In contrast, Method 2 in Algorithm B.2 does not

produce a (perfect) uniform distribution on $\{0, \ldots, q-1\}$ even for ideal random number generators with values in $\{0, \ldots, 2^{n+64}-1\}$. However, the deviations are so small that, according to current knowledge, they cannot be exploited by an attacker.

(ii) The second method has the advantage that any existing skewnesses on $\{0, \ldots, 2^{n+64}-1\}$ are generally reduced. Therefore, only this method is recommended for PTG.2-compliant random number generators. However, it should be noted that the direct use of PTG.2 generators is no longer recommended.

(iii) The disadvantage of Method 1 is that the number of iterations (and thus the runtime) is not constant. For some applications, however, it may be necessary to guarantee an upper bound on the runtime. At this point it should be noted that the probability that a random number uniformly distributed on $k \in \{0, 1, \ldots, 2^n-1\}$ is less than q is greater than $q/2^n \geq 2^{n-1}/2^n = 1/2$.

B.5. Generation of Prime Numbers

B.5.1. Preliminary Remarks

When defining the system parameters for RSA-based asymmetric schemes, two prime numbers p and q must be chosen. For the security of the schemes, it is necessary that these prime are kept secret. This requires, in particular, that p and q are chosen randomly. With regard to the usability of an application in which RSA-based schemes are used, it is also important that the prime number generation can be performed efficiently. It should be noted that proprietary speed optimisations in key generation can cause significant cryptographic weaknesses, see for example [99]. It is therefore strongly recommended to use mechanisms that are publicly known and have been examined with regard to their security.

Routines for the generation of random prime numbers are also needed for the generation of system parameters for EC- or finite field arithmetic-based cryptosystems without special properties. The requirements for these primes differ from those for the RSA mechanism in that primes need not be kept secret, but instead it may be relevant that their generation is *verifiable random*. Further details and references on this topic can be found in Section B.3.

B.5.2. Methods for Generating Prime Numbers

Three mechanisms are acceptable for generating random primes lying in a given interval $[a, b] \cap \mathbb{N}$, which can be briefly summarised as follows:

1.) Uniform generation of random primes by rejection sampling;

2.) Uniform generation of an invertible residue class r with respect to $B\#$, where $B\#$ is the *primorial* of B, that means the product of all primes smaller than B, followed by the choice of a prime of suitable size with residue $r \bmod B\#$ by rejection sampling;

3.) Generation of a random number s of suitable size which is coprime to $B\#$ and search for the next prime in the arithmetic sequence given by $s, s + B\#, s + 2 \cdot B\#, \ldots$.

The first two methods are equally recommended; the third method produces certain statistical biases in the distribution of the generated primes, which are generally undesirable. However, it is widely used in practice (see, for example, [109, Table 1]) and there is currently no evidence that the induced statistical biases can be used for attacks. Therefore, this method is accepted as a legacy method in this Technical Guideline.

The following tables provide a more detailed description of the three methods supported by this Technical Guideline:

Algorithmus B.3: Recommended Method 1 for prime number generation by rejection sampling.

Input: Interval $I \coloneqq [a, b] \cap \mathbb{N}$
Output: $p \in I$ prime

1: Choose $p \in I$ odd and uniformly distributed on I.
2: **while** p composite **do**
3: Choose $p \in I$ odd and uniformly distributed on I.
4: **end while**

Algorithmus B.4: Recommended Method 2 for prime number generation by efficiency-optimised rejection sampling.

Input: Interval $I \coloneqq [a, b] \cap \mathbb{N}$, $B \in \mathbb{N}$ with $S \coloneqq B\# \ll b - a$
Output: $p \in I$ prim

1: Choose r in $(\mathbb{Z}/S)^*$ uniformly distributed (equivalently, choose $r < S$ randomly with $\gcd(r, S) = 1$).
2: Choose $k \in \mathbb{N}$ randomly such that $p \coloneqq kS + r \in I$ (equivalently, choose k equally distributed on $[\lceil (a - r)/S \rceil, \lfloor (b - r)/S \rfloor]$).
3: **while** p composite **do**
4: Choose $k \in \mathbb{N}$ randomly such that $p \coloneqq kS + r \in I$ (equivalently, choose k equally distributed on $[\lceil (a - r)/S \rceil, \lfloor (b - r)/S \rfloor]$).
5: **end while**

Algorithmus B.5: Legacy Method for prime number generation by incremental search.

Input: Interval $I \coloneqq [a, b] \cap \mathbb{N}$, $B \in \mathbb{N}$ with $S \coloneqq B\# \ll b - a$
Output: $p \in I$ prim

1: **repeat**
2: Choose r in $(\mathbb{Z}/S)^*$ uniformly distributed (equivalently, choose $r < S$ randomly with $\gcd(r, S) = 1$).
3: Choose $k \in \mathbb{N}$ randomly such that $p \coloneqq kS + r \in I$ (equivalently: choose k equally distributed on $[\lceil (a - r)/S \rceil, \lfloor (b - r)/S \rfloor]$).
4: **while** p composite, $p \in I$ **do**
5: $p \leftarrow p + S$
6: **end while**
7: **until** p prime

For reasons of efficiency, a probabilistic primality test is usually used as a primality test in the algorithms described above. The following algorithm is recommended in this Technical Guideline:

Miller-Rabin, see [80, Algorithmus 4.24].

Table B.5: Recommended probabilistic primality test.

Remark B.3 (Miller-Rabin Algorithm) The Miller-Rabin algorithm requires, in addition to the number p to be examined, a random value $x \in \{2, 3, \ldots, p - 2\}$, the so-called basis. If x is uniformly

distributed on $\{2, 3, \ldots, p-2\}$, the probability that p is composite although the Miller-Rabin algorithm outputs that p is a prime, is at most $1/4$.

Worst Case: To bound the probability that a fixed number p is output as a prime number by means of the Miller-Rabin algorithm even though it is composite to 2^{-120}, the algorithm must be repeated 60 times, each time with bases $x_1, \ldots, x_{60} \in \{2, 3, \ldots, p-2\}$ chosen independently of each other with respect to the uniform distribution, see further Section B.4 for recommended mechanisms for computing uniformly distributed random numbers on $\{2, 3, \ldots, p-2\}$.

Average Case: In order to test a randomly with respect to the uniform distribution chosen odd number number $p \in [2^{b-1}, 2^b - 1]$ with the desired certainty for its prime property, far fewer iterations of the Miller-Rabin algorithm are sufficient than the estimate given above would suggest, compare [28], [98, Appendix C] and [64, Annex A]. For example, for $b = 1536$, only four iterations are needed to exclude, with a remaining error probability of 2^{-128}, that p is is composite, although the Miller-Rabin algorithm identifies p as a prime number [64]. Again, the bases must be chosen independently of each other at random with respect to the uniform distribution on $\{2, 3, \ldots, p-2\}$. The concrete number of necessary operations depends on the bit length of p, since the numbers for which the worst-case estimates apply decrease significantly in density as the size of the numbers increases.

Optimisations: To optimise the runtime of, for example, Algorithm B.3, it can be helpful to eliminate composite numbers with very small factors by trial division or sieving techniques before applying the probabilistic primality test. Such a preliminary test has only minor impact on the probability that numbers classified as prime by the test are composite. The recommendations on the required number of repetitions of the Miller-Rabin test therefore apply unchanged to variants of the algorithm optimised in this way.

Other Comments: When generating prime numbers which are to be used in particularly security-critical functions of a cryptosystem or whose generation is not very time-critical, it is recommended in this Technical Guideline to perform a verification of the prime number property with 60 rounds of the Miller-Rabin test, see also [88, 98]. This applies, for example, to prime numbers that are generated once as permanent parameters of a cryptographic mechanism and are then not changed for a long period of time and are possibly used by many users.

A random bit generator of the functionality class PTG.3, DRG.4, DRG.3, or NTG.1 may be used to generate the required random numbers. When using a deterministic random number generator, the generation of uniformly distributed prime numbers is from an information-theoretical point of view not possible, but this does not create a security risk: Under cryptographic standard assumptions, a random number generator of the functionality class DRG.3 or DRG.4 generates random numbers with a distribution that cannot be distinguished from an ideal distribution by any known attack with realistic practical effort when using classical computers.

However, it should be noted in this context that the security level of the generated RSA moduli may in this case be limited by the security level of the random bit generation. This would be the case, for example, if a random bit generator with a security level of 120 bits were used to generate RSA keys of a length of 4096 bits.

Alternative Primality Tests: The choice of a primality test is not security critical from a cryptanalytic point of view as long as the selected test does not misclassify prime numbers as composite and as long as the probability that composite numbers pass the test is negligible. Therefore, other tests for which these properties have been demonstrated in the literature may be used in place of the Miller-Rabin test without loss of conformity to this Technical Guideline. However, the use of the very widely known Miller-Rabin algorithm is advantageous with regard to,

among other things, a verification of the correctness of an implementation as well as a check of side-channel resistance.

B.5.3. Generation of Prime Number Pairs

To ensure the security of key pairs for which the underlying RSA moduli have been calculated by multiplying two prime numbers generated independently of each other using one of the appropriate methods, it is important that the interval $I := [a, b] \cap \mathbb{N}$ is not too narrow. If key pairs are to be generated whose modulus N has a predetermined bit length n, it is natural to choose $I = [\lceil \frac{2^{(n/2)}}{\sqrt{2}} \rceil, \lfloor 2^{(n/2)} \rfloor] \cap \mathbb{N}$. A different choice of I is compliant with this Technical Guideline if the same interval I is used for p and q and $\mathrm{Card}(I) \geq 2^{-8}b$.

B.5.4. Notes on the Security of the Recommended Methods

In the following, π denotes the prime number function, which is defined as $\pi(x) := \mathrm{Card}(\{n \in \mathbb{N}\colon n \leq x, n \text{ prime}\})$. According to the prime number theorem, $\pi(x)$ is asymptotically equivalent to $x/\ln(x)$, that means,

$$\lim_{x \to \infty} \frac{\pi(x) \cdot \ln(x)}{x} = 1.$$

The security of the methods for prime number generation recommended here is based on the following observations:

the incidence of prime numbers is independent of the chosen residue class

- All three methods can generate any prime number contained in the given interval if the underlying random bit generator can generate all candidates from the respective interval.
- The first two methods generate primes whose distribution is practically indistinguishable from a uniform distribution when the recommended security parameters are used. This is obvious for the first method; for the second method it follows heuristically from *Dirichlet's prime number theorem*: The relative frequency of primes is asymptotically the same in all invertible residue classes modulo S, and the residue class modulo S of the prime to be generated is chosen according to the uniform distribution on $(\mathbb{Z}/S)^*$.
- Strictly speaking, the argument for the security of the second method just given provides no guarantee that for a concrete S and a concrete given interval I the frequency of primes during the search does not in fact depend on the chosen residue class $r \bmod S$. Indeed, it is clear that this asymptotic statement will not be valid when S approaches the order of magnitude of $b-a$. However, it is reasonable to assume that there are no significant differences in terms of prime density between the different residue classes when the number of primes in each residue class is large. The interval I contains $\pi(b) - \pi(a)$ primes, so for each residue class $\mathrm{mod} S$ $\frac{\pi(b)-\pi(a)}{\varphi(S)}$ primes are expected. For numbers of the order of about 1000 bits, this expected value can be estimated with a small relative error to $\frac{b\ln(a)-a\ln(b)}{\ln(a)\ln(b)\varphi(S)}$ as long as $\varphi(S)$ is small compared to the numerator of the fraction. It is recommended to choose S such that $\frac{b\ln(a)-a\ln(b)}{\ln(a)\ln(b)\varphi(S)} \geq 2^{64}$ holds.
- The qualitative reasoning given above are sufficient to assess the second method as suitable. In the literature, there are more detailed investigations of closely related methods for prime number generation, see for instance [51].
- The third method generates primes that are not uniformly distributed, even though the biases in the distribution of the generated primes are – according to current knowledge – considered to be practically unexploitable by an attacker. The probability of a prime p in the interval I being output by this method is proportional to the length of the prime-free section in the

arithmetic sequence $p - kS$, $p - (k-1)S, \ldots, p - S$, p terminated by p. Since the prime density in these arithmetic sequence tends to increase for large S, this effect is expected to be most pronounced for $S = 2$. Again, however, in practice it means only a very limited loss of entropy. One can limit the distribution bias upwards by terminating and restarting the search with a new starting value if no prime has been found after a reasonable number T of steps: In this case, all prime numbers following a gap of length $\geq T$ are output with equal probability.

Appendix C.

Protocols for Special Cryptographic Applications

This chapter deals with protocols that can be used as building blocks of cryptographic solutions. In the current version of this Technical Guideline, this only concerns the protocols Secure Real-Time Transport Protocol (SRTP) and Messaging Layer Security (MLS), as corresponding information for TLS [33], IPsec [34] and SSH [35] has been moved to parts two to four of this Technical Guideline, see [33, 34, 35].

In general, the use of established protocols in the development of cryptographic systems has the advantage of being able to fall back on comprehensive public analysis. In contrast, in-house developments can easily contain vulnerabilities that are difficult for a developer to detect. It is therefore recommended that, wherever possible, to prefer generally available, possibly standardised and extensively evaluated protocols to in-house protocol developments.

C.1. SRTP

SRTP is a protocol that supplements the audio and video protocol RTP (Real-Time Transport Protocol) with functions to ensure confidentiality and integrity of the transmitted messages. It is defined in RFC 3711 [8]. SRTP must be combined with a key management protocol as it does not provide its own mechanisms for negotiating a cryptocontext.

Within this Technical Guideline, the following specifications are recommended when using SRTP:

- As a symmetric encryption scheme with combined integrity protection, AES in Galois/Counter Mode as in [79] is recommended.
- As an alternative encryption method, both AES in counter mode and in f8 mode as in [8] are recommended. A SHA1-based HMAC may be used here as integrity protection, since the use of hash functions of the SHA2 or SHA3 family is not specified in [8]. This HMAC may be reduced to 80 bits in the context of the protocol.
- MIKEY [4] should be used as the key management system. The following key management procedures from [4] are recommended: DH key exchange with authentication via PKI, RSA with PKI, and pre-shared keys. In general, only cryptographic mechanisms recommended in this Technical Guideline should be used within MIKEY and SRTP as components.
- zRTP should only be used if it would involve disproportionate high effort to solve the problem of key distribution by a public key mechanism using a PKI or by pre-distributing secret keys.
- It is strongly recommended to use the mechanisms provided in [8] for replay and integrity protection in SRTP. The „sliding window approach“ from [8] should be used. For integrity protection, mechanisms from Chapter 5 should be used.

In applications for the secure transmission of audio and video data in real time, particular attention should be paid to minimise the creation of side channels, for example through data transmission rate, the chronological order of different signals or other traffic analysis. Otherwise, attacks such as those presented in [6] are possible.

C.2. MLS

Messaging Layer Security (MLS) is a protocol for exchanging keys for secure communication in the context of (group) messaging. It was developed to meet the need for end-to-end encryption and confidentiality even in large group chats and represents a further development of the double ratchet protocol. MLS provides a protocol framework based on an asynchronous key encapsulation method for tree structures, the so-called TreeKEM, which enables the members of a group to derive and update shared keys. The associated computational complexity scales with the logarithm of the group size, allowing efficient and asynchronous distribution of group keys with forward secrecy and post-compromise security.

The protocol requires the specification of a cipher suite, which consists of the protocol version and the set of cryptographic algorithms to be used. In addition to the security level (in bits), the following cryptographic algorithms must be specified:

- KEM algorithm used for HPKE in ratchet tree operations,
- AEAD algorithm used for HPKE and message protection,
- Hash algorithm used for HPKE and the MLS transcript hash,
- Signature algorithm used for message authentication.

For MLS 1.0, the following cipher suites are recommended in this Technical Guideline:

- MLS_128_DHKEMP256_AES128GCM_SHA256_P256, see [7, Section 17.1],
- MLS_256_DHKEMP384_AES256GCM_SHA384_P384, see [7, Section 17.1],
- MLS_256_DHKEMP521_AES256GCM_SHA512_P521, see [7, Section 17.1].

Table C.1: Recommended cipher suites for MLS 1.0.

Bibliography

[1] M. Abdalla, M. Bellare, and P. Rogaway. DHIES: An encryption scheme based on the Diffie-Hellman problem. 2001. https://cseweb.ucsd.edu/~mihir/papers/dhies.pdf.

[2] M. R. Albrecht, , D. J. Bernstein, T. Chou, C. Cid, J. Gilcher, T. Lange, V. Maram, I. von Maurich, R. Misoczki, R. Niederhagen, K. G. Paterson, E. Persichetti, C. Peters, P. Schwabe, N. Sendrier, J. Szefer, C. J. Tjhai, M. Tomlinson, and W. Wang. Classic McEliece: conservative code-based cryptography: cryptosystem specification, 2022. https://classic.mceliece.org/mceliece-spec-20221023.pdf.

[3] E. Alkim, J. W. Bos, L. Ducas, P. Longa, I. Mironov, M. Naehrig, V. Nikolaenko, C. Peikert, A. Raghunathan, and D. Stebila. FrodoKEM: Learning With Errors Key Encapsulation. Einreichung zur dritten Runde des NIST PQC-Standardisierungswettbewerbs, 2021. https://frodokem.org/files/FrodoKEM-specification-20210604.pdf.

[4] J. Arkko, E. Carrara, F. Lindholm, M. Naslund, and K. Norrman. MIKEY: Multimedia Internet KEYing. RFC 3830, 2004. https://datatracker.ietf.org/doc/html/rfc3830.

[5] atsec information security GmbH. Documentation and Analysis of the Linux Random Number Generator. Recurring study under the authority of Federal Office for Information Security (BSI). https://www.bsi.bund.de/LinuxRNG.

[6] L. Ballard, S. Coull, F. Monrose, G. Masson, and C. Wright. Spot me if you can: recovering spoken phrases in encrypted VoIP conversations. *IEEE Symposium on Security and Privacy*, 2008.

[7] R. Barnes, B. Beurdouche, R. Robert, J. Millican, E. Omara, and K. Cohn-Gordon. The Messaging Layer Security (MLS) Protocol. RFC 9420, 2023. https://datatracker.ietf.org/doc/html/rfc9420.

[8] M. Baugher, D. McGrew, M. Naslund, E. Carrara, and K. Norrman. The Secure Real-time Transport Protocol (SRTP). RFC 3711, 2004. https://datatracker.ietf.org/doc/html/rfc3711.

[9] M. Bellare, R. Canetti, and H. Krawczyk. Keying Hash Functions for Message Authentication. In *Advances in Cryptology – CRYPTO 1996*, volume 1109 of *LNCS*, pages 1–15. Springer, 1996.

[10] M. Bellare, R. Canetti, and H. Krawczyk. HMAC: Keyed-Hashing for Message Authentication. RFC 2104, 1997. https://datatracker.ietf.org/doc/html/rfc2104.

[11] M. Bellare and S. K. Miner. A Forward-Secure Digital Signature Scheme. In *Advances in Cryptology – CRYPTO 1999*, volume 1666 of *LNCS*, pages 431–448, 1999.

[12] D. J. Bernstein. Cost analysis of hash collisions: Will quantum computers make SHARCS obsolete? *SHARCS*, 2009.

[13] A. Biryukov, D. Dinu, D. Khovratovich, and S. Josefsson. Argon2 Memory-Hard Function for Password Hashing and Proof-of-Work Applications. RFC 9106, 2021. https://www.rfc-editor.org/info/rfc9106.

[14] A. Biryukov, O. Dunkelman, N. Keller, D. Khovratovich, and A. Shamir. Key Recovery Attacks of Practical Complexity on AES-256 Variants With Up to 10 Rounds. In *Advances in Cryptology – EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 299–319, 2010.

[15] A. Biryukov and D. Khovratovich. Related-Key Cryptanalysis of the Full AES-192 and AES-256. In *Advances in Cryptology – ASIACRYPT 2009*, volume 5912 of *LNCS*, pages 1–18, 2009.

[16] S. Blake-Wilson and A. Menezes. Unknown Key-Share Attacks on the Station-to-Station (STS) Protocol. In *Public Key Cryptography*, Volume 1560 of LNCS, pages 154–170. Springer, 1999.

[17] D. Bleichenbacher. Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS# 1. In *Advances in Cryptology – CRYPTO 1998*, volume 1462 of *LNCS*, pages 1–12. Springer, 1998.

[18] H. Böck, J. Somorovsky, and C. Young. Return Of Bleichenbacher's Oracle Threat (ROBOT). In *27th USENIX Security Symposium (USENIX Security 18)*, pages 817–849. USENIX Association, 2018.

[19] A. Bogdanov, D. Khovratovich, and C. Rechberger. Biclique cryptanalysis of the full AES. In *Advances in Cryptology – ASIACRYPT 2011*, Volume 7073 of LNCS, pages 344–371. Springer, 2011.

[20] D. Boneh and G. Durfee. Cryptanalysis of RSA with private key d less than $N^{0.292}$. *IEEE transactions on Information Theory*, 46(4):1339–1349, 2000.

[21] G. Brassard, P. Høyer, and A. Tapp. Quantum cryptanalysis of hash and claw-free functions. *ACM SIGACT News*, 28(2).

[22] D. R. L. Brown. Generic Groups, Collision Resistance, and ECDSA. *Designs, Codes and Cryptography*, 35(1):119–152, 2005.

[23] D. R. L. Brown and R. P. Gallant. The Static Diffie-Hellman Problem. Cryptology ePrint Archive, Report 2004/306, 2004. https://ia.cr/2004/306.

[24] J. Buchmann, E. Dahmen, and A. Hülsing. XMSS – A Practical Forward Secure Signature Scheme Based on Minimal Security Assumptions. In *Post-Quantum Cryptography*, volume 7071 of *LNCS*, pages 117–129. Springer, 2011.

[25] S. Chen, R. Wang, X. Wang, and K. Zhang. Side-Channel Leaks in Web Applications: A Reality Today, a Challenge Tomorrow. In *2010 IEEE Symposium on Security and Privacy*, pages 191–206. IEEE, 2010. http://research.microsoft.com/pubs/119060/WebAppSideChannel-final.pdf.

[26] J. H. Cheon. Security analysis of the strong Diffie-Hellman problem. In *Advances in Cryptology – EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 1–11. Springer, 2006.

[27] J.-S. Coron, D. Naccache, and J. P. Stern. On the security of RSA padding. In *Advances in Cryptology – CRYPTO 1999*, volume 1666 of *LNCS*, pages 1–18. Springer, 1999.

[28] I. Damgård, P. Landrock, and C. Pomerance. Average Case Error Estimates for the Strong Probable Prime Test. *Mathematics of computation*, 61(203):177–194, 1993.

[29] W. Diffie, P. C. Van Oorschot, and M. J. Wiener. Authentication and Authenticated Key Exchanges. *Designs, Codes and Cryptography*, 2(2):107–125, 1992.

[30] ECRYPT – CSA. Algorithms, Key Size and Protocols Report, 2018. https://www.ecrypt.eu.org/csa/documents/D5.4-FinalAlgKeySizeProt.pdf.

[31] ECRYPT – II. Algorithms, Key Size and Protocols Report, 2012. https://www.ecrypt.eu.org/ecrypt2/documents/D.SPA.20.pdf.

[32] ETSI. ETSI TS 103 744: CYBER; Quantum-safe Hybrid Key Exchanges. V1.1.1, 2020. https://www.etsi.org/deliver/etsi_ts/103700_103799/103744/01.01.01_60/ts_103744v010101p.pdf.

[33] Federal Office for Information Security (BSI). BSI TR-02102-2: Cryptographic Mechanisms: Recommendations and Key Lengths, part 2 – Use of Transport Layer Security (TLS). https://www.bsi.bund.de/EN/Themen/Unternehmen-und-Organisationen/Standards-und-Zertifizierung/Technische-Richtlinien/TR-nach-Thema-sortiert/tr02102/tr02102_node.html.

[34] Federal Office for Information Security (BSI). BSI TR-02102-3: Cryptographic Mechanisms: Recommendations and Key Lengths, part 3 – Use of Internet Protocol Security (IPsec) and Internet Key Exchange (IKEv2). https://www.bsi.bund.de/EN/Themen/Unternehmen-und-Organisationen/Standards-und-Zertifizierung/Technische-Richtlinien/TR-nach-Thema-sortiert/tr02102/tr02102_node.html.

[35] Federal Office for Information Security (BSI). BSI TR-02102-4: Cryptographic Mechanisms: Recommendations and Key Lengths, part 4 – Use of Secure Shell (SSH). https://www.bsi.bund.de/EN/Themen/Unternehmen-und-Organisationen/Standards-und-Zertifizierung/Technische-Richtlinien/TR-nach-Thema-sortiert/tr02102/tr02102_node.html.

[36] Federal Office for Information Security (BSI). BSI TR-03110-2: Advanced Security Mechanisms for Machine Readable Travel Documents and eIDAS Token, Part 2 – Protocols for electronic IDentification, Authentication and trust Services (eIDAS). https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/TechGuidelines/TR03110/BSI_TR-03110_Part-2-V2_2.pdf.

[37] Federal Office for Information Security (BSI). BSI TR-03111: Elliptic Curve Cryptography. https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/TechGuidelines/TR03111/BSI-TR-03111_V-2-1_pdf.pdf.

[38] Federal Office for Information Security (BSI). BSI TR-03116-2: Kryptographische Vorgaben für Projekte der Bundesregierung, Teil 2 – Hoheitliche Ausweisdokumente. https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR03116/BSI-TR-03116-2.pdf.

[39] Federal Office for Information Security (BSI). BSI TR-03116-4: Kryptographische Vorgaben für Projekte der Bundesregierung, Teil 4 – Kommunikationsverfahren in Anwendungen. https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR03116/BSI-TR-03116-4.pdf.

[40] Federal Office for Information Security (BSI). BSI TR-03116: Kryptographische Vorgaben für Projekte der Bundesregierung. https://www.bsi.bund.de/DE/Themen/Unternehmen-und-Organisationen/Standards-und-Zertifizierung/Technische-Richtlinien/TR-nach-Thema-sortiert/tr03116/TR-03116_node.html.

[41] Federal Office for Information Security (BSI). BSI TR-03125 – TR-ESOR: Preservation of Evidence of Cryptographically Signed Document. https://www.bsi.bund.de/EN/Themen/Unternehmen-und-Organisationen/Standards-und-Zertifizierung/Technische-Richtlinien/TR-nach-Thema-sortiert/tr03125/TR-03125_node.html.

[42] Federal Office for Information Security (BSI). A proposal for: Functionality classes for random number generators. Version 2, 2011. https://www.bsi.bund.de/dok/ais-20-31-appx-2011.

[43] Federal Office for Information Security (BSI). AIS 20: Funktionalitätsklassen und Evaluationsmethodologie für deterministische Zufallszahlengeneratoren. Version 3, 2013. https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Zertifizierung/Interpretationen/AIS_20_pdf.pdf.

[44] Federal Office for Information Security (BSI). AIS 31: Funktionalitätsklassen und Evaluationsmethodologie für physikalische Zufallszahlengeneratoren. Version 3, 2013. https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Zertifizierung/Interpretationen/AIS_31_pdf.pdf.

[45] Federal Office for Information Security (BSI). Appendix to AIS 46: Minimum Requirements for Evaluating Side-Channel Attack Resistance of RSA, DSA and Diffie-Hellman Key Exchange Implementations. Version 1, 2013. https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Zertifizierung/Interpretationen/AIS_46_BSI_guidelines_SCA_RSA_V1_0_e_pdf.pdf.

[46] Federal Office for Information Security (BSI). Appendix to AIS 46: Minimum Requirements for Evaluating Side-Channel Attack Resistance of Elliptic Curve Implementations. Version 2, 2016. https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Zertifizierung/Interpretationen/AIS_46_ECCGuide_e_pdf.pdf.

[47] Federal Office for Information Security (BSI). Quantum-safe cryptography – fundamentals, current developments and recommendations. 2021. https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/Brochure/quantum-safe-cryptography.pdf.

[48] Federal Office for Information Security (BSI). Appendix to AIS 46: Fundamentals of Evaluating Side-Channel and Fault Attack Resistance. Version 1, 2023.

[49] Federal Office for Information Security (BSI). Appendix to AIS 46: Machine-Learning based Side-Channel Attack Resistance. Version 1, 2023.

[50] N. Ferguson. Authentication weaknesses in GCM. *Comments submitted to NIST Modes of Operation Process*, 2005. https://csrc.nist.gov/csrc/media/projects/block-cipher-techniques/documents/bcm/comments/cwc-gcm/ferguson2.pdf.

[51] P.-A. Fouque and M. Tibouchi. Close to uniform prime number generation with fewer random bits. *IEEE Transactions on Information Theory*, 65(2):1307–1317, 2019.

[52] M. Gebhardt, G. Illies, and W. Schindler. A note on the practical value of single hash collisions for special file formats. In *Sicherheit 2006, Sicherheit – Schutz und Zuverlässigkeit*, pages 333–344. Gesellschaft für Informatik e.V., 2006. https://dl.gi.de/bitstream/handle/20.500.12116/24792/GI-Proceedings-77-41.pdf.

[53] D. Gillmor. Negotiated Finite Field Diffie-Hellman Ephemeral Parameters for Transport Layer Security (TLS). RFC 7919, 2016. https://datatracker.ietf.org/doc/html/rfc7919.

[54] L. K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, pages 212–219. Association for Computing Machinery, 1996.

[55] S. Gueron, A. Langley, and Y. Lindell. AES-GCM-SIV: Nonce Misuse-Resistant Authenticated Encryption.

[56] N. Heninger, Z. Durumeric, E. Wustrow, and J. A. Halderman. Mining Your Ps and Qs: Detection of Widespread Weak Keys in Network Devices. In *21st USENIX Security Symposium (USENIX Security 12)*, pages 205–220. USENIX Association, 2012. https://www.usenix.org/system/files/conference/usenixsecurity12/sec12-final228.pdf.

[57] R. Housley. Cryptographic Message Syntax (CMS). RFC 5652, 2009. https://datatracker.ietf.org/doc/html/rfc5652.

[58] G. Illies, M. Lochter, and O. Stein. Behördliche Vorgaben zu kryptografischen Algorithmen. *Datenschutz und Datensicherheit-DuD*, 35(11):807–811, 2011.

[59] International Organization for Standardization. ISO/IEC 18033-2:2006 Information technology – Security techniques – Encryption algorithms – Part 2: Asymmetric ciphers, 2006.

[60] International Organization for Standardization. ISO/IEC 14888-2:2008 Information technology – Security techniques – Digital signatures with appendix – Part 2: Integer factorization based mechanisms, 2008.

[61] International Organization for Standardization. ISO/IEC 9797-1:2011 Information technology – Security techniques – Message Authentication Codes (MACs) – Part 1: Mechanisms using a block cipher, 2011.

[62] International Organization for Standardization. ISO/IEC 11770-2:2018 Information security – Key management – Part 2: Mechanisms using symmetric techniques, 2018.

[63] International Organization for Standardization. ISO/IEC 14888-3:2018 Information technology – Digital signatures with appendix – Part 3: Discrete logarithm based mechanisms, 2018.

[64] International Organization for Standardization. ISO/IEC 18032:2020 Information security – Prime number generation, 2020.

[65] International Organization for Standardization. ISO/IEC 9796-2:2010 Information technology – Security techniques – Digital signature schemes giving message recovery – Part 2: Integer factorization based mechanisms, 2020.

[66] International Organization for Standardization. ISO/IEC 11770-3:2021 Information security – Key management – Part 3: Mechanisms using asymmetric techniques, 2021.

[67] T. Iwata, K. Ohashi, and K. Minematsu. Breaking and Repairing GCM Security Proofs. In *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *LNCS*, pages 31–49. Springer, 2012.

[68] K. Jang, A. Baksi, H. Kim, G. Song, H. Seo, and A. Chattopadhyay. Quantum analysis of AES. Cryptology ePrint Archive, Paper 2022/683, 2022. https://eprint.iacr.org/2022/683.

[69] S. Kent. IP Encapsulating Security Payload (ESP). RFC 4303, 2005. https://datatracker.ietf.org/doc/html/rfc4303.

[70] T. Kivinen and M. Kojo. More Modular Exponential (MODP) Diffie-Hellman Groups for Internet Key Exchange (IKE). RFC 3526, 2003. https://datatracker.ietf.org/doc/html/rfc3526.

[71] H. Krawczyk and P. Eronen. HMAC-based Extract-and-Expand Key Derivation Function (HKDF). RFC5869, 2010. https://www.rfc-editor.org/info/rfc5869.

[72] A. K. Lenstra. Key lengths. In *Handbook of Information Security*, volume II, 2006.

[73] A. K. Lenstra and E. R. Verheul. Selecting Cryptographic Key Sizes. *Journal of Cryptology*, 14(4):255–293, 2001.

[74] G. Leurent and T. Peyrin. From Collisions to Chosen-Prefix Collisions Application to Full SHA-1. In *Advances in Cryptology – EUROCRYPT 2019*, volume 11478 of *LNCS*, pages 527–555. Springer, 2019.

[75] G. Leurent and T. Peyrin. SHA-1 is a Shambles: First Chosen-Prefix Collision on SHA-1 and Application to the PGP Web of Trust. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 1839–1856. USENIX Association, 2020. https://www.usenix.org/system/files/sec20-leurent.pdf.

[76] M. Lochter and J. Merkle. Elliptic Curve Cryptography (ECC) Brainpool Standard Curves and Curve Generation. RFC 5639, 2010. https://datatracker.ietf.org/doc/html/rfc5639.

[77] S. Lucks and M. Daum. The Story of Alice and her Boss: Hash Functions and the Blind Passenger Attack. Presentation. https://www.cits.rub.de/imperia/md/content/magnus/rumpec05.pdf.

[78] V. G. Martínez, F. H. Álvarez, L. H. Encinas, and C. S. Ávila. A Comparison of the Standardized Versions of ECIES. In *Sixth International Conference on Information Assurance and Security, IAS 2010*, pages 1–4. IEEE, 2010.

[79] D. McGrew and K. Igoe. AES-GCM Authenticated Encryption in the Secure Real-Time Transport Protocol (SRTP). RFC 7714, 2015. https://datatracker.ietf.org/doc/html/rfc7714.

[80] A. J. Menezes, P. C. V. Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.

[81] R. C. Merkle. Secure Communications over Insecure Channels. *Communications of the ACM*, 21(4):294–299, 1978.

[82] K. Moriarty, B. Kaliski, J. Jonsson, and A. Rusch. PKCS #1: RSA Cryptography Specifications Version 2.2. RFC 8017, 2016. https://datatracker.ietf.org/doc/html/rfc8017.

[83] National Institute of Standards and Technology. Special Publication NIST SP 800-38A: Recommendation for Block Cipher Modes of Operation: Methods and Techniques, 2001. https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38a.pdf.

[84] National Institute of Standards and Technology. Federal Information Processing Standards FIPS PUB 197: Advanced Encryption Standard (AES), 2001, updated 2023. https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197-upd1.pdf.

[85] National Institute of Standards and Technology. Special Publication NIST SP 800-38C: Recommendation for Block Cipher Modes of Operation: The CCM Mode for Authentication and Confidentiality, 2007. https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-38c.pdf.

[86] National Institute of Standards and Technology. Special Publication NIST SP 800-38D: Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC, 2007. https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38d.pdf.

[87] National Institute of Standards and Technology. Special Publication NIST SP 800-38E: Recommendation for Block Cipher Modes of Operation: The XTS-AES Mode for Confidentiality on Storage Devices, 2010. https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-38e.pdf.

[88] National Institute of Standards and Technology. Federal Information Processing Standards FIPS PUB 186-4: Digital Signature Standard (DSS), 2013. https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf.

[89] National Institute of Standards and Technology. Federal Information Processing Standards FIPS PUB 180-4: Secure Hash Standard, 2015. https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf.

[90] National Institute of Standards and Technology. Federal Information Processing Standards FIPS PUB 202: SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions, 2015. https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf.

[91] National Institute of Standards and Technology. Special Publication NIST SP 800-185: SHA-3 Derived Functions: cSHAKE, KMAC, TupleHash, and ParallelHash, 2016. https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-185.pdf.

[92] National Institute of Standards and Technology. Special Publication NIST SP 800-38B: Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication, 2016. https://nvlpubs.nist.gov/nistpubs/specialpublications/nist.sp.800-38b.pdf.

[93] National Institute of Standards and Technology. Special Publication NIST SP 800-56B Revision 2: Recommendation for Pair-Wise Key Establishment Using Integer Factorization Cryptography, 2019. https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-56Br2.pdf.

[94] National Institute of Standards and Technology. Special Publication NIST SP 800-208: Recommendation for Stateful Hash-Based Signature Schemes, 2020. https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-208.pdf.

[95] National Institute of Standards and Technology. Special Publication NIST SP 800-56C Revision 2: Recommendation for Key-Derivation Methods in Key-Establishment Schemes, 2020. https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-56Cr2.pdf.

[96] National Institute of Standards and Technology. Special Publication NIST SP 800-57 Part 1 Revision 5: Recommendation for Key Management: Part 1 – General, 2020. https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57pt1r5.pdf.

[97] National Institute of Standards and Technology. Special Publication NIST SP 800-108r1: Recommendation for Key Derivation Using Pseudorandom Functions, 2022. https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-108r1.pdf.

[98] National Institute of Standards and Technology. Federal Information Processing Standards FIPS PUB 186-5: Digital Signature Standard (DSS), 2023. https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-5.pdf.

[99] M. Nemec, M. Sys, P. Svenda, D. Klinec, and V. Matyas. The Return of Coppersmith's Attack: Practical Factorization of Widely Used RSA Moduli. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1631–1648. Association for Computing Machinery, 2017.

[100] P. Q. Nguyen and I. E. Shparlinski. The Insecurity of the Elliptic Curve Digital Signature Algorithm with Partially Known Nonces. *Designs, Codes and Cryptography*, 30(2):201–217, 2003.

[101] R. Perlner, J. Kelsey, and D. Cooper. Breaking category five SPHINCS+ with SHA-256. In *Post-Quantum Cryptography*, pages 501–522. Springer International Publishing, 2022.

[102] J.-F. Raymond and A. Stiglic. Security Issues in the Diffie-Hellman Key Agreement Protocol. 2000.

[103] T. Ristenpart and S. Yilek. When Good Randomness Goes Bad: Virtual Machine Reset Vulnerabilities and Hedging Deployed Cryptography. In *Network and Distributed System Security Symposium (NDSS) 2010*, 2010.

[104] A. Shamir. How to Share a Secret. *Communications of the ACM*, 22(11):612–613, 1979.

[105] P. W. Shor. Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. *SIAM J. Comput.*, 26(5):1484–1509, 1997.

[106] D. X. Song, D. A. Wagner, and X. Tian. Timing Analysis of Keystrokes and Timing Attacks on SSH. In *10th USENIX Security Symposium (USENIX Security 01)*. USENIX Association, 2001. https://www.usenix.org/legacy/events/sec2001/full_papers/song/song.pdf.

[107] M. Stevens, E. Bursztein, P. Karpman, A. Albertini, and Y. Markov. The First Collision for Full SHA-1. In *Advances in Cryptology – CRYPTO 2017*, volume 10401 of *LNCS*, pages 570–596. Springer, 2017.

[108] S. Vaudenay. Security Flaws Induced by CBC Padding – Applications to SSL, IPSEC, WTLS, ... In *Advances in Cryptology – EUROCRYPT 2002*, volume 2332 of *LNCS*, pages 534–545. Springer, 2002.

[109] P. Švenda, M. Nemec, P. Sekan, R. Kvašňovskỳ, D. Formánek, D. Komárek, and V. Matyáš. The Million-Key Question – Investigating the Origins of RSA Public Keys. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 893–910. USENIX Association, 2016. https://www.usenix.org/system/files/conference/usenixsecurity16/sec16_paper_svenda.pdf.

[110] M. Zhandry. A note on the quantum collision and set equality problems. *Quantum Info. Comput.*, 15(7–8):557–567, 2015.