Federal Office
for Information Security

Technical Guideline BSI TR-03111

Elliptic Curve Cryptography

Version 2.10
Date: 2018-06-01

History

Version	Date	Comment
1.00	2007-02-14	Initial public version.
1.10	2009-02-03	Enhancements, corrections, and clarifications.
1.11	2009-04-17	Bug fixes.
2.00	2012-06-28	Extension by further algorithms and protocols, corrections and updates.
2.10	2018-06-01	Adaption of EC Schnorr, enhancements, corrections, and clarifications

Bundesamt für Sicherheit in der Informationstechnik
Postfach 20 03 63, 53133 Bonn, Germany
Email: `EllipticCurveCrypto@bsi.bund.de`
Internet: `http://www.bsi.bund.de`

Contents

List of Figures

List of Tables

1. Introduction

Elliptic curve cryptography (ECC) is a very efficient technology to realise public key cryptosystems and public key infrastructures (PKI). The security of a public key system using elliptic curves is based on the difficulty of computing discrete logarithms in the group of points on an elliptic curve defined over a finite field. The elliptic curve discrete logarithm problem (ECDLP), described in Section 2.3.3, is currently believed to be asymptotically harder than the factorization of integers or the computation of discrete logarithms in the multiplicative group of a finite field (DLP), described in Section 2.2.5. As a matter of fact key sizes of cryptosystems based on elliptic curves are short compared to cryptosystems based on integer factorization at the same level of security.

The aim of this technical guideline is to facilitate the application of elliptic curve cryptography by giving recommendations on the secure deployment of elliptic curve cryptography in commercial applications. For that purpose, this guideline compiles

- mathematical foundations of elliptic curves and
- algorithms based on elliptic curves in one document.

Furthermore, this guideline sets requirements on the suitable deployment of ECC in applications. Examples include but are not limited to official German documents, smart metering systems or cash registers.

The algorithms described here are the elliptic curve based signature algorithms ECDSA, ECGDSA, ECSDSA and EC-KCDSA for generating and verifying digital signatures, the Elliptic Curve Key Agreement Algorithm (ECKA) for key establishment and the Password Authenticated Connection Establishment (PACE).

Specific requirements on parameters and key lengths to be used in German government projects are defined in [11]. General recommendations and suitable key lengths can be found in [9]. The deployment of ECC to classified information is not in the scope of this guideline.

1.1. Patents and side-channel attacks

In implementations, patents and side-channel attacks play an important role.

The algorithms described in this guideline have been carefully selected to allow patent-free and/or license-free implementations. Nevertheless, some of the described algorithms or its particular implementations may be subject of patent rights. The BSI shall not be held responsible for identifying any or all such patent rights.

Implementors and security evaluators shall also pay attention to [4], which gives a general guidance to assess the side-channel resistance of implementations on smartcards.

1.2. Standards

This document refers to a number of international standards related to elliptic curve cryptography. Many national and international organizations have standardized the use of elliptic curves in cryptography. The most important organizations and the corresponding standards are:

1. The International Organization for Standardization (ISO) has issued the following relevant standards:

- ISO 14888 (*Information technology – Security techniques – Digital signatures with appendix*) – Part 3, including Amendment 1 (*Discrete logarithm based techniques*) [19], [20]
- ISO 11770 (*Information technology – Security techniques – Key Management*) – Part 3 (*Mechanisms using asymmetric techniques*) [18]

2. The American National Standards Institute (ANSI) has standardized protocols for digital signatures and for key agreement. The following standards are relevant:
   - X9.62 (*Public Key Cryptography For The Financial Services Industry – The Elliptic Curve Digital Signature Algorithm (ECDSA))* [2].
   - X9.63 (*Public Key Cryptography For The Financial Services Industry – Key Agreement and Key Transport Using Elliptic Curve Cryptography*) [3].
3. The Institute of Electrical and Electronics Engineers (IEEE) has issued the standard P1363 (*Standard Specifications for Public Key Cryptography*) [14] and its amendment P1363a [15]. The standards describe commonly used cryptosystems like RSA, DSA, and cryptosystems based on elliptic curves.
4. The IETF published in RFC 5639 (*Elliptic Curve Cryptography (ECC) Brainpool Standard Curves and Curve Generation*) [31] an set of domain parameters defining cryptographically strong groups on elliptic curves.

1.3. Symbols and Abbreviations

The following notations and abbreviations are used in this document:

Symbol	**Comments**
$\mathbb{N}$	The set of all natural numbers (without 0).
$\mathbb{Z}$	The set of all integers.
$\mathbb{Z}_m$	The set of all integers modulo m.
p	A prime number.
$\mathbb{F}_p$	The finite field of p elements.
$\mathbb{F}_{2^m}$	The finite field of 2^m elements, with $m \in \mathbb{N}$.
E	An elliptic curve defined by a Weierstraß equation. If E is defined over a finite field of characteristic $p > 3$, then the Weierstraß equation is of the form $$y^2 = x^3 + ax + b, \qquad a, b \in \mathbb{F}_p \, , 4a^3 + 27b^2 \neq 0. \qquad (1.1)$$ Essentially, this technical guideline considers elliptic curves over prime fields of characteristic $p > 3$.

continued on next page

continued from page 8	
Symbol	**Comments**
$E(\mathbb{F}_p)$	An elliptic curve group over the field $\mathbb{F}_p$ consisting of all points $(x, y) \in \mathbb{F}_p^2$ solving the Weierstraß equation of E together with the point at infinity $\mathcal{O}$.
$\#E(\mathbb{F}_p)$	The order (or cardinality) of the group $E(\mathbb{F}_p)$.
$\mathcal{O}$	The point at infinity. It is the identity element of the group $E(\mathbb{F}_p)$ and can not be described in affine coordinates.
P, Q	Points on the elliptic curve $E(\mathbb{F}_p)$.
x_P, y_P	The x- and y-coordinates of P in affine representation, if P is different from $\mathcal{O}$.
$P + Q$	The sum of two points P and Q in $E(\mathbb{F}_p)$.
$[k]P$	The k-th multiple of a point $P \in E(\mathbb{F}_p)$, i.e. $[k]P = P + P + \cdots + P$, k addends.
G	The base point is a generator of a subgroup of $E(\mathbb{F}_p)$.
n	The order of the base point G. Typically, n is a prime of bit length ≥ 224.
A	The sender of a cryptographic message.
B	The receiver of a cryptographic message.
d_{A}	The private key of entity A. This is an integer in the set $\{1, \ldots, n-1\}$.
P_{A}	The public key of entity A. This is a point on $E(\mathbb{F}_p)$. The relation between d_{A} and P_{A} is given by the equation $P_{\mathsf{A}} = [d_{\mathsf{A}}]G$ except for the signature schemes ECGDSA and ECKCDSA, where the relation is $P'_{\mathsf{A}} = [d_{\mathsf{A}}^{-1} \mod n]G$.
$\widetilde{P_{\mathsf{A}}}$	An ephemeral public key of entity A.
$\mathrm{H}(M)$	Hash value (digest) of the message M.
$\mathrm{H}_l(M)$	Truncated hash value of the message M. The hash value is cropped to the l leftmost bits of $\mathrm{H}(M)$.
ℓ	Bit length of the output of a hash function.
κ	Bit length of a symmetric key or key stream.
τ	Bit length of the order of the base point, i.e. $\tau = \lceil \log_2 n \rceil$.
$R \oplus S$	Bitwise sum of two octet or bit strings R, S.

Table 1.1.: Symbols and abbreviations.

1.4. Terminology

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in RFC 2119 [8].

2. Mathematical Foundations

This section introduces the mathematical foundations required to understand elliptic curve cryptography: First an overview on modular arithmetic is given in Section 2.1. Then the basic properties of groups and finite fields are introduced in Section 2.2. Finally, elliptic curves over finite fields $\mathbb{F}_p$ are described in Section 2.3.

2.1. Modular Arithmetic

The following description is based on P1363 [14]. However, this guideline makes use of a slightly different notation.

Modular arithmetic fixes an integer $m > 1$ called the *modulus*. The fundamental operation in the context of modular arithmetic is the reduction modulo m. Given an integer a, one divides a by m and takes the remainder r as the result of the reduction. Therefore, r is in the range $0 \leq r \leq m-1$. The operation is written as

$$r = a \bmod m. \tag{2.1}$$

Let a and b be two integers with remainders r_1 and r_2, respectively. Then a and b are said to be *congruent* modulo m, if and only if $r_1 = r_2$. This relationship is written as $a \equiv b \bmod m$. The following two properties of congruences can easily be seen:

1. Integers a and b are congruent modulo m if and only if $b - a$ is divisible by m.

2. If $r = a \bmod m$ then $r \equiv a \bmod m$.

The integers modulo m are the possible remainders modulo m. They are denoted by $\mathbb{Z}_m$. Thus the set of integers modulo m is

$$\mathbb{Z}_m = \{0, 1, \ldots, m-1\}.$$

Next, we enumerate properties of addition, subtraction, multiplication, and division in $\mathbb{Z}_m$. Let a_0, b_0, a_1, b_1 be integers with $a_0 \equiv b_0 \bmod m$ and $a_1 \equiv b_1 \bmod m$. Thus, $a_0 \bmod m$ and $b_0 \bmod m$ represent the same element in Z_m. The same holds for a_1 and b_1.

$$a_0 + a_1 \equiv b_0 + b_1 \bmod m \tag{2.2}$$
$$a_0 - a_1 \equiv b_0 - b_1 \bmod m \tag{2.3}$$
$$a_0 \cdot a_1 \equiv b_0 \cdot b_1 \bmod m \tag{2.4}$$

Equation (2.2) shows that the order of adding and reducing modulo m may be exchanged. Equations (2.3) and (2.4) show the same property for subtraction and multiplication modulo m, respectively.

Typically, one performs addition, subtraction, and multiplication in $\mathbb{Z}_m$ by performing the corresponding integer operation and reducing the result modulo m. Then, all computations take place in the set $\{0, 1, \ldots, (m-1)^2\}$, i.e. the largest number appearing in an intermediate result before reduction is $(m-1)^2$.

2.2. Groups and Finite Fields

2.2.1. Groups

A *group* $(\mathsf{G}, \circ)$ is a set G with a binary operation $\circ : \mathsf{G} \times \mathsf{G} \to \mathsf{G}$ such that the following four axioms are satisfied:

Associativity: For all $a, b, c \in \mathsf{G}$ the equation $(a \circ b) \circ c = a \circ (b \circ c)$ holds.

Identity element: There is an element $e \in \mathsf{G}$ such that for all $a \in \mathsf{G}$ the equation $e \circ a = a \circ e = a$ holds.

Inverse element: For each $a \in \mathsf{G}$ there exists an element $b \in \mathsf{G}$ such that $a \circ b = b \circ a = e$.

It is easy to see that for a group the identity element e is unique – and so is the inverse for each element of the group.

If $(\mathsf{G}, \circ)$ is a group, then $\circ$ is called the *group law*. Often a group satisfies $a \circ b = b \circ a$ for all $a, b \in \mathsf{G}$. Then G is said to be *commutative* or *Abelian*.

In practice, the group law is commonly written as an addition or a multiplication:

Additive Notation. The identity is denoted by 0. If g is an element of G, the inverse element is denoted by $-g$. We define $[k]g = \sum_1^k g$, $k \in \mathbb{N}$, as the sum of k times the element g.

Multiplicative Notation. The identity is denoted by 1. If g is an element of G, the inverse element is denoted by g^{-1}. We define $g^k = \prod_1^k g$, $k \in \mathbb{N}$, as the product of k times the element g.

2.2.2. Group Order and Generators

Let G be a finite group, i.e. G contains n elements, $n \in \mathbb{N}$. The number n is called the *group order* and $\#\mathsf{G} = n$. As G is finite, for every $g \in \mathsf{G}$ an integer s with $1 \leq s \leq n$ exists such that $[s]g = 0$. The smallest such number is written as $\#g$, called the *order* of g in G. If s denotes the order of g in G, then the following properties hold:

1. The order of the group is a multiple of the order of all its elements, i.e. s divides n.
2. For $g \neq 1$, the representation $[k]g$ is unique, i.e. $[k_1]g = [k_2]g$ if and only if $k_1 \equiv k_2 \bmod s$.

A finite group $(\mathsf{G}, +)$ of order n is called *cyclic*, if there is a group element $g \in \mathsf{G}$ with

$$\mathsf{G} = \{g, [2]g, [3]g, \ldots, [n-1]g, [n]g\}.$$

In this case, the element g is called a *generator* of $(\mathsf{G}, +)$.

2.2.3. Subgroups

Let $(\mathsf{G}, +)$ be a finite group. A non-empty subset $\mathsf{S} \subseteq \mathsf{G}$ is called a *subgroup*, if for any two elements $a, b \in \mathsf{S}$ it holds that $a - b \in \mathsf{S}$. Due to Lagrange's theorem, $\#\mathsf{S}$ is a divisor of $\#\mathsf{G}$.

For every $a \in \mathsf{G}$ the set

$$\langle a \rangle = \{[k]a; 1 \leq k \leq \#a\}$$

is a cyclic subgroup of G.

2.2.4. Finite Fields

A field $(F, +, \cdot)$ is a set F together with two operations $+$ and $\cdot$ such that

1. $+ : F \times F \to F$ and $\cdot : F \times F \to F$,
2. $(F, +)$ is an Abelian group,
3. $(F \setminus \{0\}, \cdot)$ is an Abelian group,
4. $(a + b) \cdot c = a \cdot c + b \cdot c$ holds for all $a, b, c \in F$.

A finite field is a field with finitely many elements. It is a fundamental theorem of the theory of finite fields that a finite field of q elements exists if and only if q is a prime power, i.e. $q = p^m$ where p is a prime and m is an integer with $m \geq 1$. In addition, for a given prime power q there exists up to isomorphism only one finite field consisting of q elements. In the following, this field is denoted by $GF(q)$ or $\mathbb{F}_q$.

As of today, two families of finite fields are used for elliptic curve cryptography in practice:

Prime fields: Finite fields $\mathbb{F}_p$ of p elements with p prime. In this case $\mathbb{F}_p$ is isomorphic to $(\mathbb{Z}_p, +, \cdot)$ (cf. Section 2.1), therefore in this Technical Guideline elements of $\mathbb{F}_p$ will be regarded as integers in $\{0, 1, \ldots, p-1\}$.

Extension fields of characteristic 2: Finite fields $\mathbb{F}_{2^m}$ of 2^m elements.

In this technical guideline, we focus mainly on elliptic curve cryptography over prime fields.

2.2.5. The Discrete Logarithm Problem (DLP)

The *discrete logarithm problem* (DLP) is defined as follows: Let G be a cyclic group of order n with generator g. The discrete logarithm of $h \in \mathsf{G}$ to the base g, denoted by $\log_g h$, is the unique integer k, $0 \leq k \leq n-1$, such that $[k]g = h$.

Given g and h, the discrete logarithm problem is to find k, which is assumed to be computationally intractable for the relevant groups in ECC for large n (cf. Section 2.3.3).

2.3. Elliptic Curves over prime fields

According to this guideline, it is RECOMMENDED to use elliptic curves over prime fields $\mathbb{F}_p$ where $p \equiv 3 \bmod 4$ (cf. Section 3.2.2).

The security of elliptic curve cryptography is based on the hardness of the elliptic curve discrete logarithm problem (cf. Section 2.3.3).

2.3.1. Elliptic Curve Groups

We introduce the basic facts of elliptic curves over a finite field $\mathbb{F}_p$. Let E be an elliptic curve over $\mathbb{F}_p$. In this section it is assumed, that $p \neq 2, 3$. Then E may be described in terms of the Weierstraß equation

$$y^2 = x^3 + ax + b, \qquad a, b \in \mathbb{F}_p, \quad 4a^3 + 27b^2 \neq 0. \tag{2.5}$$

The requirement $4a^3 + 27b^2 \neq 0$ ensures that E is non-singular, this means in particular that one may compute the tangent in every point on the curve.

Several different representations for elliptic curves exist. Within this guideline only the *affine representation* (cf. Equation (2.5)) is used.

The set of *rational points* in E over $\mathbb{F}_p$ denoted by $E(\mathbb{F}_p)$ is

$$E(\mathbb{F}_p) = \{(x, y) \in \mathbb{F}_p^2 : y^2 = x^3 + ax + b\} \cup \{\mathcal{O}\}, \tag{2.6}$$

where $\mathcal{O}$ is the point at infinity. It is the projective closure of the equation $y^2 = x^3 + ax + b$ and may not be described in terms of two coordinates in $\mathbb{F}_p$.

$E(\mathbb{F}_p)$ carries a group structure with the point at infinity acting as the identity element. The binary operation of rational points in $E(\mathbb{F}_p)$ is commonly denoted as an addition. It turns out that the addition of points in $E(\mathbb{F}_p)$ has a simple geometric interpretation, as shown in Figure 2.1, visualizing the operations on a elliptic curve defined over $\mathbb{R}$.

Let $P \in E(\mathbb{F}_p)$ and $Q \in E(\mathbb{F}_p)$ be points on the elliptic curve. The addition law uses the *chord-tangent process* where the following different cases have to be distinguished:

1. Let $P + \mathcal{O} = \mathcal{O} + P = P$ for all $P \in E(\mathbb{F}_p)$. Thus $\mathcal{O}$ acts as the identity element in the group $E(\mathbb{F}_p)$.

2. Let $P \neq \mathcal{O}$ and $P = (x_P, y_P)$. The point $(x_P, -y_P)$ is an element of $E(\mathbb{F}_p) \setminus \{\mathcal{O}\}$ and one defines $-P = (x_P, -y_P)$. Additionally, one sets $-\mathcal{O} = \mathcal{O}$. The identity $P + (-P) = \mathcal{O}$ holds for all $P \in E(\mathbb{F}_p)$.

3. Let $P \neq \mathcal{O}$, and $Q \neq \mathcal{O}$ such that $P \neq \pm Q$, i.e. P and Q have different x-coordinates. The line through P and Q intersects $E(\mathbb{F}_p)$ in a third point $R \in E(\mathbb{F}_p) \setminus \{\mathcal{O}\}$. One sets $P + Q = -R$.

   This definition leads to the following addition rule: Set $\lambda = (y_Q - y_P)/(x_Q - x_P)$ and $P + Q = (x_R, y_R)$ (the denominator is different from zero, as $x_P \neq x_Q$). Then x_R and y_R may be computed by the formulae

   $$x_R = \lambda^2 - x_P - x_Q, \qquad y_R = \lambda(x_P - x_R) - y_P.$$

4. Let $P \neq \mathcal{O}$, $P \neq -P$. The tangent to $E(\mathbb{F}_p)$ in P intersects $E(\mathbb{F}_p)$ in $R \in E(\mathbb{F}_p) \setminus \{\mathcal{O}\}$, and we set $[2]P = -R$.

   This description leads to the following doubling rule: Set $\lambda = (3x_P^2 + a)/(2y_P)$ and $[2]P = (x_R, y_R)$. Then x_R and y_R may be computed by the formulae

   $$x_R = \lambda^2 - 2x_P, \qquad y_R = \lambda(x_P - x_R) - y_P.$$

The chord-tangent process for an elliptic curve over real numbers is shown in Figure 2.1. With the definitions above $(E(\mathbb{F}_p), +)$ is an Abelian group.

The order of $E(\mathbb{F}_p)$ may be estimated due to a theorem of Hasse:

$$p + 1 - 2\sqrt{p} \leq \#E(\mathbb{F}_p) \leq p + 1 + 2\sqrt{p}. \tag{2.7}$$

Hasse's theorem shows $\#E(\mathbb{F}_p) \approx p$, i.e. p and $\#E(\mathbb{F}_p)$ are of same order of magnitude.

2.3.2. Elliptic Curve Domain Parameters

Elliptic curve domain parameters yield a set of information for communicating parties to identify a certain elliptic curve group for use in cryptography. The domain parameters comprise the finite field $\mathbb{F}_p$, the coefficients a and b of the Weierstraß equation, a base point $G \in E(\mathbb{F}_p)$, its order n, and finally the cofactor $h = \frac{\#E(\mathbb{F}_p)}{n}$. The base point G generates a cyclic subgroup of order n in $E(\mathbb{F}_p)$ denoted by $\langle G \rangle$, i.e.:

$$\langle G \rangle = \{G, [2]G, [3]G, \ldots, [n-1]G, [n]G\}.$$

Table 2.1 summarizes the domain parameters of an elliptic curve defined over $\mathbb{F}_p$ with $p > 3$.

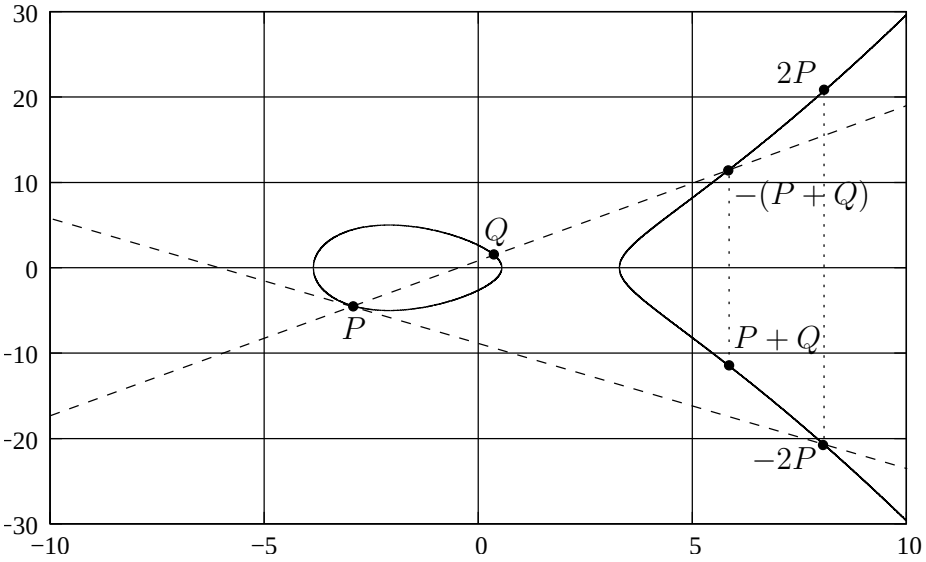

Figure 2.1.: Operations on an elliptic curve $E(\mathbb{R})$.

Parameter	Comment
p	A prime number specifying the underlying field $\mathbb{F}_p$.
a	The first coefficient of the Weierstraß equation E.
b	The second coefficient of the Weierstraß equation E.
G	A base point in $E(\mathbb{F}_p)$.
n	The order of G in $E(\mathbb{F}_p)$.
h	The cofactor of G in $E(\mathbb{F}_p)$.

Table 2.1.: Elliptic curve domain parameters over $\mathbb{F}_p$.

2.3.3. Elliptic Curve Discrete Logarithm Problem

The *elliptic curve discrete logarithm problem* (ECDLP) is defined as follows: Given the elliptic curve domain parameters as described above and a point $P \in \langle G \rangle$, find the unique integer k, $0 \leq k \leq n-1$ such that $P = [k]G$.

This is a special case of the general discrete logarithm problem as explained in Section 2.2.5.

An elliptic curve group is called *cryptographically strong* if the underlying ECDLP is considered to be computationally intractable for the application in use.

Cryptographically strong elliptic curve groups for different security levels are published by various standardization bodies (e.g. ANSI, ISO, IETF, NIST).

2.3.4. Cryptographically Strong Elliptic Curve Domain Parameters over $\mathbb{F}_p$

Cryptographically strong elliptic curve domain parameters SHALL be used (see also [5], [7] and [31] for the generation of suitable curves). The ECDLP is currently considered to be intractable, if at least the following conditions hold :

1. The order n of the base point G MUST be a prime of a suitable bit length. Cf. [11] and [9] for corresponding requirements.
2. To avoid the elliptic curve to be anomalous the order n MUST be different from p.
3. The ECDLP MUST NOT be reducible to the DLP in a multiplicative group $\mathbb{F}_{p^r}$ for a 'small' integer r. Thus, it is REQUIRED that $p^r \not\equiv 1 \bmod n$ for all $1 \leq r \leq 10^4$.

4. The class number of the principal order belonging to the endomorphism ring of E SHOULD be at least 200. [1]

However, as the generation and validation of domain parameters is non-trivial (cf. for example [5]), it is RECOMMENDED to use standardized domain parameters, generated by trusted third parties. Cryptographically strong domain parameters can be found in Section 6.

[1] If an elliptic curve is generated at random, this curve respects this requirement with a very high probability (cf. [5], [7] and [31] for the generation and validation of domain parameters, and the calculation of class numbers).

3. Data Types and Data Conversion

The cryptographic algorithms specified in this guideline involve operations using several different data types. These data types are considered as abstract data types consisting of distinct sets of elements, e.g. an octet string is regarded as distinct from a bit string. This distinction helps to clarify the requirements placed on implementations and helps to avoid subtle coding errors.

In all, five data types are employed in this document:

1. Octet strings (`OS`)
2. Bit strings (`BS`)
3. Integers (`I`)
4. Field elements (`FE`)
5. Elliptic curve points (`ECP`)

It is often necessary to convert one data type into another one. Section 3.1 describes the conversion routines as summarized in Table 3.1. Section 3.2 describes two encoding mechanisms for elliptic curve points.

Conversion routine	**Abbreviation**	**Section**
Bit String to Octet String	`BS2OS`	3.1.1
Octet String to Bit String	`OS2BS`	3.1.1
Integer to Octet String	`I2OS`	3.1.2
Octet String to Integer	`OS2I`	3.1.2
Finite Field Element to Octet String	`FE2OS`	3.1.3
Octet String to Finite Field Element	`OS2FE`	3.1.3

Table 3.1.: Conversion routines for data types used in this guideline.

3.1. Conversion Routines

The big endian notation is assumed to be used in the following conversion routines.

3.1.1. Conversion between Bit Strings and Octet Strings

`BS2OS`

The data conversion primitive that converts a bit string to an octet string is called *Bit String to Octet String Conversion Primitive* or `BS2OS`. It takes a bit string of length d as input and outputs the corresponding octet string of length $l = \lceil d/8 \rceil$. The bit string and the octet string are written as $b_{d-1}\, b_{d-2}\, \cdots\, b_1\, b_0$ and $M_{l-1}\, M_{l-2}\, \cdots\, M_1\, M_0$, respectively.

The conversion is quite simple: One simply pads enough zeros on the left of the bit string to make its length a multiple of 8, and then chops the padded bit string up into octets. More precisely, one proceeds as follows:

1. $l = \lceil d/8 \rceil$.
2. For $0 \leq i \leq l-2$ do $M_i = b_{8i+7}\, b_{8i+6}\, \cdots b_{8i+2}\, b_{8i+1}\, b_{8i}$.
3. $M_{l-1} = 0\, \cdots\, 0\, b_{d-1}\, \cdots\, b_{8(l-1)+1}\, b_{8(l-1)}$, where the number of zeros at the left of M_{l-1} is equal to $8l - d$.
4. Output $M_{l-1}\, M_{l-2}\, \cdots M_1\, M_0$.

`OS2BS`

The data conversion primitive that converts an octet string to a bit string is called *Octet String to Bit String Conversion Primitive* or `OS2BS`. It takes an octet string of length l as input and outputs the corresponding bit string of length $d = 8l$. Assume that the octet string and the bit string are written as $M_{l-1}\, M_{l-2}\, \cdots\, M_1\, M_0$ and $b_{d-1}\, b_{d-2}\, \cdots\, b_1\, b_0$, respectively.

Each octet is interpreted as a bit string of length 8. The result is then the concatenation of these bit strings. More precisely, one proceeds as follows:

1. $d = 8l$.
2. For $0 \leq i \leq l-1$ do $b_{8i+7}\, b_{8i+6}\, \cdots b_{8i+2}\, b_{8i+1}\, b_{8i} = M_i$.
3. Output $b_{d-1}\, b_{d-2}\, \cdots b_1\, b_0$.

3.1.2. Conversion between Integers and Octet Strings

`I2OS`

The data conversion primitive that converts integers to octet strings is called *Integer to Octet String Conversion Primitive* or `I2OS`. It takes a non-negative integer x and the desired length l of the octet string as input. The length l has to satisfy $256^l > x$. `I2OS` outputs the corresponding octet string. If $256^l \leq x$, the conversion algorithm SHALL output *error*.

The idea is to write a non-negative integer x in its unique l-digit representation to the base 256:

$$x = x_{l-1} \cdot 256^{l-1} + x_{l-2} \cdot 256^{l-2} + \cdots + x_1 \cdot 256 + x_0, \quad 0 \leq x_i < 256 \text{ for } 0 \leq i \leq l-1.$$

As usual, the leftmost bit in each digit x_i is the most significant bit. We denote the octet string by $M_{l-1}\, M_{l-2}\, \cdots\, M_1\, M_0$. One sets $M_i = x_i$ for $0 \leq i \leq l-1$.

Note: One or more leading digits will be zero if $x < 256^{l-1}$.

`OS2I`

The primitive that converts octet strings to integers is called *Octet String to Integer Conversion Primitive* or `OS2I`. It takes a non-empty octet string of length $l \in \mathbb{N}$ as input and outputs the corresponding integer x as explained below. In addition, for empty octet strings, i.e. $l = 0$, this guideline defines $x = 0$.

Let the octet string be $M_{l-1}\, M_{l-2}\, \cdots\, M_1\, M_0$. Each octet is interpreted as a non-negative integer to the base 256, where the leftmost bit is the most significant one, i.e. one sets $x_i = M_i$ for $0 \leq i \leq l-1$. Then

$$x = x_{l-1} \cdot 256^{l-1} + x_{l-2} \cdot 256^{l-2} + \cdots + x_1 \cdot 256 + x_0.$$

Note: The octet string of length zero (the empty octet string) is converted to the integer 0.

3.1.3. Conversion between Field Elements and Octet Strings

`FE2OS`

The primitive that converts field elements to octet strings is called *Field Element to Octet String Conversion Primitive* or `FE2OS`. It takes a field element as input and outputs the corresponding octet string.

A field element $x \in \mathbb{F}_p$ is converted to an octet string of length $l = \lceil \log_{256} p \rceil$ by applying the conversion function `I2OS` as described in Section 3.1.2 with parameter l, i.e. $\texttt{FE2OS}(x) = \texttt{I2OS}(x, l)$. Here the element $x \in \mathbb{F}_p$ is represented as an integer $x \in \{0, 1, \ldots, p-1\}$ (cf. Section 2.2.4).

`OS2FE`

The primitive that converts octet strings to field elements is called *Octet String to Field Element Conversion Primitive* or `OS2FE`. It takes an octet string as input and outputs the corresponding field element.

An octet string X is converted to a field element by applying the conversion function `OS2I` as described in Section 3.1.2 and reducing the output modulo p, i.e. $\texttt{OS2FE}(X) = \texttt{OS2I}(X) \bmod p$.

3.2. Encoding Elliptic Curve Points

Let p be a prime $p \neq 2, 3$ and let E be an elliptic curve over $\mathbb{F}_p$ given by its Weierstraß equation $y^2 = x^3 + ax + b$. Let $P \in E(\mathbb{F}_p)$ be a point on the elliptic curve. This guideline represents the point P by an octet string:

- If $P \neq \mathcal{O}$, the point is represented by its affine coordinate(s). Either a *compressed* (P_C) or an *uncompressed* (P_U) encoding is used.
- If $P = \mathcal{O}$, the point is always represented by the single octet $0x00$ independent of the encoding.

3.2.1. Uncompressed Encoding

In uncompressed encoding the point P is represented by two field elements, its x-coordinate denoted by x_P and its y-coordinate denoted by y_P. If b is the bit length of p, storing (x_P, y_P) requires $2b$ bits (excluding additional data required for the encoding).

Encoding

The uncompressed encoding P_U is defined as $P_U = C \parallel X \parallel Y$, where

- $C = 0x04$
- $X = \texttt{FE2OS}(x_P)$
- $Y = \texttt{FE2OS}(y_P)$

Decoding

Given P_U the point P is recovered as $P = (\texttt{OS2FE}(X), \texttt{OS2FE}(Y))$. Before using P it MUST be validated that P is indeed a point on the elliptic curve E by checking that $y_P^2 = x_P^3 + ax_P + b$.

3.2.2. Compressed Encoding

In compressed encoding the point P is represented by its x-coordinate x_P and an additional bit to uniquely identify the y-coordinate y_P. More precisely, the bit y'_P is defined to be the rightmost bit of y_P, i.e. $y'_P = 0$ if and only if y_P is even.

Encoding

The compressed encoding P_C is defined as $P_C = C \parallel X$, where

- If $y'_P = 0$, set $C = 0x02$
- If $y'_P = 1$, set $C = 0x03$
- $X = \texttt{FE2OS}(x_P)$

Decoding

Given P_C the point P is recovered as $P = (\texttt{OS2FE}(X), y_P)$, where the following algorithm is used to calculate y_P:

1. Set $\alpha = x_P^3 + ax_P + b$.
2. Check whether α is a square in $\mathbb{F}_p$. If α is a non-square, output error and terminate.
3. If $\alpha = 0$, then $y_P = 0$. Output y_P and terminate.
4. Compute a square root $\beta \in \mathbb{F}_p$ of α in $\mathbb{F}_p$.
5. If the rightmost bit of β is equal to y'_P, then $y_P = \beta$. Otherwise, $y_P = p - \beta$. Output y_P and terminate.

To efficiently check whether α is a square in $\mathbb{F}_p$, the Legendre-Symbol

$$\left(\frac{\alpha}{p}\right) \equiv \alpha^{(p-1)/2} \bmod p$$

can be used. More precisely, α is a square in $\mathbb{F}_p$ if and only if $\left(\frac{\alpha}{p}\right) = 1$.

Note: According to this guideline primes $p \equiv 3 \bmod 4$ are RECOMMENDED. In this case, the square roots $\pm\beta$ of α can be efficiently computed as $\beta = \alpha^{(p+1)/4} \bmod p$.

4. Elliptic Curve Cryptography Algorithms

This section specifies cryptographic algorithms for elliptic curves: Section 4.1 provides definitions for auxiliary functions, i.e. random number generators, hash functions, and key generation. In Section 4.2, the elliptic curve based digital signature algorithms ECDSA, ECGDSA and ECSDSA are specified. Subsequently, the key agreement algorithm ECKA and PACE are described in Sections 4.3 and 4.4, respectively.

4.1. Auxiliary Functions and Algorithms

4.1.1. Random and Pseudo-Random Number Generators

Random number generators are often based on physical processes like radioactive decay or unpredictable events like the time between two strikes on a keyboard.

In practice, pseudo-random number generators (non-physical) are often used for efficiency reasons. Roughly speaking, the output of a pseudo-random number generator should be indistinguishable from the output of a true random number generator.

In this technical guideline, the notation $\mathrm{RNG}(\{1, 2, \ldots, n-1\})$ is used to denote both a random number generator and a pseudo-random number generator. The input of the function RNG is a finite set of positive integers, its output is a number randomly or pseudo-randomly chosen from this set.

The outputs of $\mathrm{RNG}(\{1, 2, \ldots, n-1\})$ SHALL be (almost) uniformly distributed within $\{1, 2, \ldots, n-1\}$. Functionality classes and evaluation methodologies for pseudo-random number generators and physical random number generators are published in an appendix to AIS 20/31 [1]. The selection of suitable classes is application-specific and defined by [11] or [9], respectively.

In many applications $\mathrm{RNG}(\{1, 2, \ldots, n-1\})$ is derived from $\mathrm{RNG}(\{0, 1, 2, \ldots, 2^k - 1\})$ with $2^k \geq n$. In this case the implementor MUST ensure that the (almost) uniform distribution of $\mathrm{RNG}(\{0, 1, 2, \ldots, 2^k - 1\})$ is maintained in $\mathrm{RNG}(\{1, 2, \ldots, n-1\})$. The following two algorithms are provided as an example. More information and additional algorithms can be found in TR-02102 [9].

Algorithm 1: This algorithm maintains uniform distribution but has probabilistic run-time.

1. $r = \mathrm{RNG}(\{0, 1, 2, \ldots, 2^k - 1\})$
2. If $(r < n)$ and $(r > 0)$, output r else goto 1.

Algorithm 2: This algorithm has deterministic run-time but does not fully maintain uniform distribution.

1. $r = \mathrm{RNG}(\{0, 1, 2, \ldots, 2^{k+64} - 1\})$
2. Output $(r \bmod (n-1)) + 1$

Note: The usage of a non-uniformly distributed $\mathrm{RNG}(\{1, 2, \ldots, n-1\})$ can enable an attack on signature algorithms (cf. Bleichenbacher's attack on DSA, described e.g. in [28]). Algorithm 2 does not provide uniform distribution. It is however assumed that the deviation from uniform distribution produced by Algorithm 2 is too small to be exploited by an attacker.

4.1.2. Cryptographically Strong Hash Functions

A hash function H maps a message M to a hash value (digest) $D = \mathrm{H}(M)$. The message M is an octet string of arbitrary length[1] and the hash value D is an octet string of fixed length $\lceil \ell/8 \rceil$, where ℓ is the bit length of the hash values produced by H():

$$\mathrm{H}\colon \{0,\ldots,255\}^* \mapsto \{0,\ldots,255\}^{\lceil \ell/8 \rceil}$$

In some cases the hash values have to be truncated. Let $\mathrm{H}_l(M)$ be the *truncated hash value* of M, i.e. the hash value $\mathrm{H}(M)$ is cropped to the l leftmost bits. $\mathrm{H}_l(M)$ SHALL be encoded as octet string using the `BS2OS` conversion.

A hash function suitable for cryptography, has to satisfy the following requirements:

Preimage resistance: For any hash value D, it is computationally infeasible to find a message M with $\mathrm{H}(M) = D$.

Second preimage resistance: For any message M, it is computationally infeasible to find a message M' with $M \neq M'$ and $\mathrm{H}(M) = \mathrm{H}(M')$.

Collision resistance: It is computationally infeasible to find arbitrary messages M and M' with $M \neq M'$ and $\mathrm{H}(M) = \mathrm{H}(M')$.

If H fulfills all these requirements, it is said to be *cryptographically strong*.

Hash functions with an output length $\ell \geq 224$ SHALL be used. Some hash functions are weaker than previously believed (cf. [34]). The hash functions listed in Table 4.1 are supported by this specification. Specific requirements/recommendations are given in [11] and [9].

Hash Function	Hash Length (bit)	Reference
SHA-224	224	[12]
SHA-256	256	[12]
SHA-384	384	[12]
SHA-512	512	[12]
SHA3-224	224	[13]
SHA3-256	256	[13]
SHA3-384	384	[13]
SHA3-512	512	[13]

Table 4.1.: Supported hash functions.

For session key derivation (cf. section 4.3.3), also the hash functions SHA-1 and RIPEMD-160 (cf. [12], [17]) are supported.

4.1.3. Elliptic Curve Key Pair Generation – ECKeyPair

An elliptic curve key pair consists of a public key P and a private key d. A key pair is generated as follows.

Input: Cryptographically strong elliptic curve domain parameters (p, a, b, G, n, h).

Output: The key pair (d, P).

Actions: The following actions are performed:

[1] Most hash functions have a restriction on the length of M.

1. $d = \mathrm{RNG}(\{1, 2, \ldots, n-1\})$
2. $P = [d]G$
   (If P is generated for ECGDSA or ECKCDSA, set $P = [d^{-1} \mod n]G$ instead).
3. Output (d, P)

4.2. Elliptic Curve Based Signature Algorithms

This section specifies the signature algorithms ECDSA, ECGDSA and ECSDSA. For that purpose, it is assumed in the following that A sends B a message M and the corresponding signature (r, s). Furthermore, it is assumed that the message M also includes information identifying the public key, the signature algorithm and the hash function H() to be used for the verification of the signature. An example for such a signature format is given in Section 5.2.

Note: It is RECOMMENDED to use a hash function H() (cf. Section 4.1.2) with an output length $\ell = \tau$, i.e. the output length of the hash function and the bit length of the order of the base point G SHOULD be equal. If for any reason the hash function has to be chosen such that $\ell > \tau$, the hash value SHALL be truncated to $\mathrm{H}_\tau(M)$, the τ leftmost bits of $\mathrm{H}(M)$.

The hash function SHOULD NOT be chosen such that $\ell < \tau$.

4.2.1. The Elliptic Curve Digital Signature Algorithm – ECDSA

This section describes the Elliptic Curve Digital Signature Algorithm abbreviated by ECDSA. The description is in conformance with [2].

4.2.1.1. Signature Algorithm

A proceeds as follows to generate the ECDSA signature (r, s) on the message M.

Input: The following inputs are needed:

1. A's private key d_A and the elliptic curve domain parameters (p, a, b, G, n, h).
2. The message M to be signed.

Output: The ECDSA signature (r, s) over M.

Actions: The following actions are performed:

1. $k = \mathrm{RNG}(\{1, 2, \ldots, n-1\})$
2. $Q = [k]G$
3. $r = \mathtt{OS2I}(\mathtt{FE2OS}(x_Q)) \bmod n$
   If $r = 0$ goto 1.
4. $k_{inv} = k^{-1} \bmod n$
5. $s = k_{inv} \cdot (r \cdot d_\mathsf{A} + \mathtt{OS2I}(\mathrm{H}_\tau(M))) \bmod n$
   If $s = 0$ goto 1.
6. Output (r, s)

Note: The signature depends on the random number k. If A signs the same document M twice, both signatures differ with a very high probability.

4.2.1.2. Verification Algorithm

B proceeds as follows to verify the received ECDSA signature (r, s) on M.

Input: The following inputs are needed:

1. A's authentic public key P_{A} and the domain parameters (p, a, b, G, n, h).
2. The signed message M.
3. The ECDSA signature (r, s).

Output: `True`, if the signature is valid, and `False` otherwise.

Actions: The following actions are performed:

1. Verify that $r, s \in \{1, 2, \ldots, n-1\}$
   If the check fails, output `False` and terminate.
2. $s_{inv} = s^{-1} \bmod n$
3. $u_1 = s_{inv} \cdot \mathtt{OS2I}(\mathrm{H}_\tau(M)) \bmod n$
   $u_2 = s_{inv} \cdot r \bmod n$
4. $Q = [u_1]G + [u_2]P_{\mathsf{A}}$
   If $Q = \mathcal{O}$, output `Error` and terminate.
5. $v = \mathtt{OS2I}(\mathtt{FE2OS}(x_Q)) \bmod n$
6. Output `True` if $v = r$, and `False` otherwise.

4.2.2. The Elliptic Curve German Digital Signature Algorithm - ECGDSA

This section introduces the Elliptic Curve German Digital Signature Algorithm. The specification matches with [22]. In the ECGDSA scheme, the elliptic curve point $P'_{\mathsf{A}} := [d_{\mathsf{A}}^{-1} \bmod n]G$ is used as public key. As a consequence, the signature creation requires no computation of a multiplicative inverse mod n.

4.2.2.1. Signature Algorithm

A proceeds as follows to generate the ECGDSA signature (r, s) on a message M.

Input: The following information is required as input:

1. A's private key d_{A} and the elliptic curve domain parameters (p, a, b, G, n, h).
2. The message M to be signed.

Output: The ECGDSA signature (r, s) over M.

Actions: The following actions are performed:

1. $k = \mathrm{RNG}(\{1, 2, \ldots, n-1\})$
2. $Q = [k]G$
3. $r = \mathtt{OS2I}(\mathtt{FE2OS}(x_Q)) \bmod n$
   If $r = 0$ goto 1.
4. $s = (k \cdot r - \mathtt{OS2I}(\mathrm{H}_\tau(M))) \cdot d_{\mathsf{A}} \bmod n$
   If $s = 0$ goto 1.
5. Output (r, s)

Note: The signature depends on the random number k. If A signs the same document M twice, both signatures differ with a very high probability.

4.2.2.2. Verification Algorithm

To verify the received ECGDSA signature (e, s) on a message M, B has to proceed as follows.

Input: The following inputs are needed:

1. A's authentic public key P_{A} and the domain parameters (p, a, b, G, n, h).
2. The signed message M.
3. The ECGDSA signature (r, s).

Output: `True`, if the signature is valid, and `False` otherwise.

Actions: The following actions are performed:

1. Verify that $r, s \in \{1, 2, \ldots, n-1\}$
   If the check fails, output `False` and terminate.
2. $r_{inv} = r^{-1} \bmod n$
3. $u_1 = r_{inv} \cdot \texttt{OS2I}(\mathrm{H}_\tau(M)) \bmod n$
   $u_2 = r_{inv} \cdot s \bmod n$
4. $Q = [u_1]G + [u_2]P_{\mathsf{A}}$
   If $Q = \mathcal{O}$, output `Error` and terminate.
5. $v = \texttt{OS2I}(\texttt{FE2OS}(x_Q)) \bmod n$
6. Output `True` if $v = r$, and `False` otherwise.

4.2.3. The Elliptic Curve Based Schnorr Digital Signature Algorithm - ECSDSA

This section describes the Elliptic Curve Based Schnorr Digital Signature Algorithm (ECSDSA)[2], which is described in the following[3]. The scheme requires no computation of a multiplicative inverse modulo n during the signature creation.

4.2.3.1. Signature Algorithm

A proceeds as follows to generate the ECSDSA signature (r, s) on a message M.

Input: The following information is required as input:

1. A's private key d_{A} and the elliptic curve domain parameters (p, a, b, G, n, h).
2. The message M to be signed.

Output: The ECSDSA signature (r, s) over M.

Actions: The following actions are performed:

1. $k = \mathrm{RNG}(\{1, 2, \ldots, n-1\})$
2. $Q = [k]G$

[2] For the Schnorr signature, see also [32].

[3] Please note that the algorithm description was slightly modified since the last version of this Technical Guideline, cf. [20] and [33].

3. $r = \texttt{OS2I}(\mathrm{H}_\tau(\texttt{FE2OS}(x_Q) \parallel \texttt{FE2OS}(y_Q) \parallel M))$
   If $r = 0 \bmod n$, goto 1.
4. $s = k + r \cdot d_\mathsf{A} \bmod n$
   If $s = 0$ goto 1.
5. Output (r, s)

Note: The signature depends on the random number k. If A signs the same document M twice, both signatures differ with a very high probability.

4.2.3.2. Verification Algorithm

Given a ECSDSA signature (r, s) on a message M, the verification procedure is the following

Input: The following inputs are needed:

1. A's authentic public key P_A and the domain parameters (p, a, b, G, n, h).
2. The signed message M.
3. The ECSDSA signature (r, s).

Output: `True`, if the signature is valid, and `False` otherwise.

Actions: The following actions are performed:

1. Verify that $r \in \{1, \ldots, 2^\tau - 1\}$ and $s \in \{1, 2, \ldots, n-1\}$.
   If the check fails, output `False` and terminate.
2. $Q = [s]G - [r]P_\mathsf{A}$
   If $Q = \mathcal{O}$, output `Error` and terminate.
3. $v = \texttt{OS2I}(\mathrm{H}_\tau(\texttt{FE2OS}(x_Q) \parallel \texttt{FE2OS}(y_Q) \parallel M))$
4. Output `True` if $v = r$, and `False` otherwise.

4.3. The Elliptic Curve Key Agreement Algorithm – ECKA

This section describes the Elliptic Curve Key Agreement Algorithm (ECKA), key derivation functions, and the key agreement protocols of Diffie-Hellman (ECKA-DH) and ElGamal (ECKA-EG). The description of ECKA is in conformance with [3].

Note: To prevent attacks based on invalid (ephemeral) public keys it MUST be checked that a received public key is indeed a point on the elliptic curve. This validation is already part of the point decoding algorithms (cf. Section 3.2). In addition to this, small subgroup attacks are prevented by using (compatible) cofactor multiplication in the key agreement algorithms.

4.3.1. Key Agreement Algorithm

A and B proceed as follows to generate a shared secret point S_AB:

Input: The private key $\widehat{d}$, the public key $\widehat{P}$, and the elliptic curve domain parameters (p, a, b, G, n, h). The private key $\widehat{d}$ and the public key $\widehat{P}$ SHALL be either both ephemeral (ECKA-DH, cf. Section 4.3.2.1) or ephemeral-static (ECKA-EG, cf. Section 4.3.2.2).

Output: The output consists of:

1. The shared secret point S_{AB}.
2. The shared secret value Z_{AB} (OPTIONAL).

Actions: The following actions are performed:

1. $l = h^{-1} \bmod n$
2. $Q = [h]\widehat{P}$
3. $S_{\mathsf{AB}} = [\widehat{d} \cdot l \bmod n]Q$

   If $S_{\mathsf{AB}} = \mathcal{O}$, output `Error` and terminate.
4. $Z_{\mathsf{AB}} = \texttt{FE2OS}(x_{S_{\mathsf{AB}}})$ (OPTIONAL)
5. Output S_{AB} and conditionally Z_{AB}

Note: To derive keys for symmetric encryption and/or integrity protection the OPTIONAL generation of Z_{AB} MUST be performed. The shared secret value Z_{AB} MUST NOT be used directly for encryption or integrity protection, key derivation functions are described in Section 4.3.3.

4.3.2. The Key Agreement Protocols ECKA-DH and ECKA-EG

To interactively generate a shared secret point S_{AB} (or a shared secret value Z_{AB}), A and B may use one of the following protocols.

4.3.2.1. Anonymous Diffie-Hellman Key Agreement (ECKA-DH).

Both A and B agree on the domain parameters (p, a, b, G, n, h), the key derivation algorithm, the cipher and/or message authentication code to be used and perform the following steps:

Initiator A		**Recipient B**
$(\widetilde{d_{\mathsf{A}}}, \widetilde{P_{\mathsf{A}}}) = \text{ECKeyPair}(p, a, b, G, n, h)$	$\widetilde{P_{\mathsf{A}}} \rightleftharpoons \widetilde{P_{\mathsf{B}}}$	$(\widetilde{d_{\mathsf{B}}}, \widetilde{P_{\mathsf{B}}}) = \text{ECKeyPair}(p, a, b, G, n, h)$
$S_{\mathsf{AB}} = \text{ECKA}(\widetilde{d_{\mathsf{A}}}, \widetilde{P_{\mathsf{B}}}, (p, a, b, G, n, h))$		$S_{\mathsf{AB}} = \text{ECKA}(\widetilde{d_{\mathsf{B}}}, \widetilde{P_{\mathsf{A}}}, (p, a, b, G, n, h))$

Table 4.2.: ECKA-DH.

4.3.2.2. ElGamal Key Agreement (ECKA-EG).

The recipient B must make the static public key P_{B} including the corresponding domain parameters (p, a, b, G, n, h) publicly available in an authentic form and performs the steps of table 4.3.

To send B an encrypted and/or integrity protected message M, A MUST include the ephemeral public key $\widetilde{P_{\mathsf{A}}}$ and information identifying the key derivation algorithm, the cipher and/or the message authentication code to be used. An example for a message format is given in Section 5.3.1.1.

4.3.3. Key Derivation Functions

The following algorithms are RECOMMENDED to derive keys from the shared secret value Z_{AB}:

Initiator A		**Recipient B**
$(\widetilde{d_{\mathsf{A}}}, \widetilde{P_{\mathsf{A}}}) = \text{ECKeyPair}(p, a, b, G, n, h)$ $S_{\mathsf{AB}} = \text{ECKA}(\widetilde{d_{\mathsf{A}}}, P_{\mathsf{B}}, (p, a, b, G, n, h))$	$\overset{M=(\widetilde{P_{\mathsf{A}}},\ldots)}{\longrightarrow}$	$S_{\mathsf{AB}} = \text{ECKA}(d_{\mathsf{B}}, \widetilde{P_{\mathsf{A}}}, (p, a, b, G, n, h))$

Table 4.3.: ECKA-EG.

X9.63 Key Derivation Function. ANSI X9.63 [3] describes a method for converting a shared secret to a cryptographic key. The algorithm $\text{KDF}_{X9.63}()$ requires to select a hash function H() from Section 4.1.2. Let ℓ denote the bit length of the hash value.

Input: The following inputs are needed:

1. An octet string Z_{AB}, which is the shared secret value.
2. An integer $\kappa < \ell \cdot (2^{32} - 1)$, which is the bit length of the keying data to be generated.
3. An octet string *SharedInfo*, which consists of some information shared between A and B (OPTIONAL).

Output: The octet string *KeyData* of length $k = \lceil \kappa/8 \rceil$.

Actions: The following actions are performed:

1. Let *counter* be a 32 bit, big-endian integer, initialized with $0x00000001$.
2. $j = \lceil \kappa/\ell \rceil$
3. For $i = 1$ to $j - 1$ do the following:
   a) $H_i = \text{H}(Z_{\mathsf{AB}} \parallel \textit{counter} \parallel [\textit{SharedInfo}])$
   b) $\textit{counter} = \textit{counter} + 1$
   c) $i = i + 1$
4. $l = \kappa - (\ell \cdot (j - 1))$
5. $H_j = \text{H}_l(Z_{\mathsf{AB}} \parallel \textit{counter} \parallel [\textit{SharedInfo}])$
6. $\textit{KeyData} = H_1 \parallel H_2 \parallel \cdots \parallel H_{j-1} \parallel H_j$
7. Output *KeyData*

Key Derivation Function for Session Keys. This paragraph describes a method for deriving cryptographic session keys of bit length κ, i.e. keys for symmetric encryption and for computing message authentication codes (MAC). The algorithm $\text{KDF}_{Session}$ requires to select a hash function H() from Section 4.1.2 with bit length $\ell \geq \kappa$.

Input: The following inputs are needed:

1. An octet string Z_{AB}, which is a shared secret value.
2. A 32-bit, big-endian integer *counter*, which is initiated as follows:
   a) Default key used for encryption:
   $\textit{counter} = 0x00000001$
   b) Default key used for authentication:
   $\textit{counter} = 0x00000002$

c) Alternative key used for encryption:

$counter = 0x00000003$

d) Alternative key used for authentication:

$counter = 0x00000004$

e) ...

3. A nonce r encoded as octet string (OPTIONAL).

Output: An octet string *KeyData*.

Actions: The following actions are performed:

1. $D = Z_{\mathsf{AB}} \parallel r \parallel counter$
2. $KeyData = \mathrm{H}_{\kappa}(D)$
3. Output *KeyData*

4.3.3.1. Key Derivation for DES.

To derive 112-bit 3DES keys the hash function SHA-1 SHALL be used with $\kappa = 112$. The parity bits of *KeyData* MAY be adjusted to form correct DES keys.

4.3.3.2. Key Derivation for AES.

- To derive 128-bit AES keys the hash function SHA-1 with $\kappa = 128$ SHALL be used.
- To derive 192-bit AES keys the hash function SHA-256 with $\kappa = 192$ SHALL be used.
- To derive 256-bit AES keys the hash function SHA-256 with $\kappa = 256$ SHALL be used.

4.4. The Password Authenticated Connection Establishment – PACE

This section describes the Password Authenticated Connection Establishment protocol, abbreviated by PACE. The protocol establishes a secure channel with strong session keys based on an authentication by means of a secret password (which MAY have low entropy).

A and B choose a key derivation function $\mathrm{KDF}_{Session}$ (e.g. the one of the section 4.3.3). Furthermore, they agree on a suitable mapping function Map() (e.g. the mapping function GMap() of section 4.4.1), a symmetric cipher (with encryption and decryption denoted by E() and $\mathrm{E}^{-1}()$, respectively), the message authentication code MAC(). Keys, input and output values of E() and MAC() are assumed to be octet strings. Let v be a fixed multiple of the block size of E().

As input, the PACE protocol requires the shared password π and the elliptic curve domain parameters $D = (p, a, b, G, n, h)$. The following actions are performed to establish the secure channel:

	Initiator A		**Recipient B**
0.	$K_\pi = \mathrm{KDF}_{Session}(\pi, 3)$		$K_\pi = \mathrm{KDF}_{Session}(\pi, 3)$
1.	$s = \mathrm{RNG}(\{0, \ldots, 2^v - 1\})$		
2.	$z = \mathrm{E}(K_\pi, \mathtt{FE2OS}(s))$		
3.	Send z	$\xrightarrow{z}$	
4.			$s = \mathtt{OS2FE}(\mathrm{E}^{-1}(K_\pi, z))$
5.	$\widetilde{D} = (p, a, b, \widetilde{G}, n, h) = \mathrm{Map}(D, s)$	$\rightleftharpoons$	$\widetilde{D} = (p, a, b, \widetilde{G}, n, h) = \mathrm{Map}(D, s)$
6.	$(\widetilde{y_\mathsf{A}}, \widetilde{Y_\mathsf{A}}) = \mathrm{ECKeyPair}(\widetilde{D})$	$\overset{\widetilde{Y_\mathsf{A}}}{\underset{\widetilde{Y_\mathsf{B}}}{\rightleftharpoons}}$	$(\widetilde{y_\mathsf{B}}, \widetilde{Y_\mathsf{B}}) = \mathrm{ECKeyPair}(\widetilde{D})$
7.	$(S_\mathsf{AB}, Z_\mathsf{AB}) = \mathrm{ECKA}(\widetilde{y_\mathsf{A}}, Y_\mathsf{B}, \widetilde{D})$		$(S_\mathsf{AB}, Z_\mathsf{AB}) = \mathrm{ECKA}(y_\mathsf{B}, \widetilde{Y_\mathsf{A}}, \widetilde{D})$
8.	$K_{Enc} = \mathrm{KDF}_{Session}(Z_\mathsf{AB}, 1)$		$K_{Enc} = \mathrm{KDF}_{Session}(Z_\mathsf{AB}, 1)$
9.	$K_{Mac} = \mathrm{KDF}_{Session}(Z_\mathsf{AB}, 2)$		$K_{Mac} = \mathrm{KDF}_{Session}(Z_\mathsf{AB}, 2)$
10.			$T_\mathsf{B} = \mathrm{MAC}(K_{Mac}, \widetilde{Y_\mathsf{A}})$
11.		$\xleftarrow[T_\mathsf{B}]{}$	Send T_B
12.	If $T_\mathsf{B} \neq \mathrm{MAC}(K_{Mac}, \widetilde{Y_\mathsf{A}})$, output `Error` and terminate.		
13.	$T_\mathsf{A} = \mathrm{MAC}(K_{Mac}, \widetilde{Y_\mathsf{B}})$		
14.	Send T_A	$\xrightarrow[T_\mathsf{A}]{}$	
15.			If $T_\mathsf{A} \neq \mathrm{MAC}(K_{Mac}, \widetilde{Y_\mathsf{B}})$, output `Error` and terminate.

Table 4.4.: PACE.

A detailed specification for an implementation of PACE on smartcards is contained in [10], a security proof of the protocol can be found in [6].

Note: For the generation of nonces in the PACE protocol, a (pseudo-)random number generator belonging to the classes K4, DRG.3, DRG.4 or PTG.3 MUST be used.

4.4.1. The Generic Mapping – GMap()

To map a nonce s to a point of the elliptic curve A and B SHOULD use the generic mapping $GMap()$. It is based on an anonynous Diffie-Hellman key agreement. The required input for the generic mapping are the domain parameters (p, a, b, G, n, h) of the curve and the nonce s that shall be mapped. Then, the protocol produces ephemeral domain parameters $\widetilde{D} = GMap(D, s)$ by computing a new base point $\widetilde{G}$ of the curve.

	Initiator A		**Recipient B**
1.	$(\widetilde{d_\mathsf{A}}, \widetilde{P_\mathsf{A}}) = \mathrm{ECKeyPair}(p, a, b, G, n, h)$	$\overset{\widetilde{P_\mathsf{A}}}{\underset{\widetilde{P_\mathsf{B}}}{\rightleftharpoons}}$	$(\widetilde{d_\mathsf{B}}, \widetilde{P_\mathsf{B}}) = \mathrm{ECKeyPair}(p, a, b, G, n, h)$
2.	$H = \mathrm{ECKA}(\widetilde{d_\mathsf{A}}, \widetilde{P_\mathsf{B}}, (p, a, b, G, n, h))$		$H = \mathrm{ECKA}(\widetilde{d_\mathsf{B}}, \widetilde{P_\mathsf{A}}, (p, a, b, G, n, h))$
3.	$\widetilde{D} = GMap(D, s) = (p, a, b, [s]G + H, n, h)$		$\widetilde{D} = GMap(D, s) = (p, a, b, [s]G + H, n, h)$

Table 4.5.: Generic Mapping.

5. Input and Output Formats

This section specifies data structures and object identifiers for in- and output of public keys, signatures, and key agreement.

The object identifier `bsi-de` represents the root of the subtree containing all objects defined in this specification:

```
bsi-de OBJECT IDENTIFIER ::= {
  itu-t(0) identified-organization(4) etsi(0)
  reserved(127) etsi-identified-organization(0) 7
}
```

The root identifier for elliptic curve cryptography is:

```
id-ecc OBJECT IDENTIFIER ::= { bsi-de algorithms(1) 1 }
```

This guideline also supports the data structures and object identifiers specified in ANSI X9.62 [2]. The root identifier for ANSI X9.62 is:

```
ansi-X9-62 OBJECT IDENTIFIER ::= {
  iso(1) member-body(2) us(840) 10045 }
```

5.1. Public Key Format

It is RECOMMENDED to store and exchange elliptic curve public keys in X9.62 format. In this case the data structures and object identifiers specified by X9.62 [2] SHALL be used.

If, however, elliptic curve cryptography is performed on smartcards, public keys SHALL be encoded as data objects as defined in ISO 7816-8 [25].

5.1.1. X9.62 Format

Public keys represented in X.509 syntax have the following ASN.1 structure:

```
SubjectPublicKeyInfo ::= SEQUENCE {
  algorithm         AlgorithmIdentifier,
  subjectPublicKey  BIT STRING
}
```

The component `algorithm` of type `AlgorithmIdentifier` specifies the type of the public key and its associated parameters. The component `subjectPublicKey` of type `BIT STRING` specifies the actual value of the public key.

The elliptic curve public key is a value of type `ECPoint`, which is simply an `OCTET STRING` as defined in Section 3.2. The conversion routine `OS2BS` SHALL be used to map the value to a `BIT STRING`.

Public keys in X9.62 format are identified by the object identifier `id-ecPublicKey` which is specified as follows:

```
id-publicKeyType  OBJECT IDENTIFIER ::= { ansi-X9-62 keyType(2) }

id-ecPublicKey    OBJECT IDENTIFIER ::= { id-publicKeyType 1 }
```

The public key parameters contained in the `AlgorithmIdentifier` are defined as a choice of three alternatives:

```
Parameters ::= CHOICE {
  ecParameters  ECParameters,
  namedCurve    OBJECT IDENTIFIER,
  implicitlyCA  NULL
}
```

ecParameters: The domain parameters are explicitly described.

namedCurve: Standardized domain parameters identified by an object identifier are used.

implicitlyCA: The domain parameters are inherited or implicitly known.

It is RECOMMENDED to use the alternatives `ecParameters` or `namedCurve` unless ephemeral public keys are exchanged. In this case `implicitlyCA` SHOULD be used instead.

> **Note:** These recommendations deviate from [30], 2.1.1. Implementations that strive for conformance to [30] MUST only support `namedCurve`.

The structure `ECParameters` is used to describe domain parameters explicitly. Version 1 MUST be used. It is specified as follows:

```
ECParameters ::= SEQUENCE {
  version   INTEGER{ecpVer1(1)} (ecpVer1),
  fieldID   FieldID,
  curve     Curve,
  base      ECPoint,
  order     INTEGER,
  cofactor  INTEGER OPTIONAL,
  ...
}

Curve ::= SEQUENCE {
  a     FieldElement,
  b     FieldElement,
  seed  BIT STRING OPTIONAL
}

FieldElement ::= OCTET STRING

ECPoint ::= OCTET STRING

FieldID ::= SEQUENCE {
  fieldType   OBJECT IDENTIFIER,
  parameters  ANY DEFINED BY fieldType
}
id-fieldType  OBJECT IDENTIFER ::= { ansi-X9-62 fieldType(1) }

prime-field   OBJECT IDENTIFIER ::= { id-fieldType 1 }

Prime-p ::= INTEGER
```

If `FieldID` refers to a `prime-field`, `Prime-p` SHALL be used as parameter.

5.1.2. ISO 7816 Format

For smartcards public keys and domain parameters MUST be exchanged as TLV (Tag-Length-Value) encoded data objects as described in ISO 7816-8 [25]. The tags and the encodings for data objects relevant to elliptic curves are given in Table 5.1.

According to this guideline the data object for domain parameters MUST either be all present or all absent. Especially for ephemeral public keys the domain parameters are usually implicitly known and MUST be all absent.

Public keys and domain parameters MUST be encapsulated in a constructed public key data object identified by tag $0x7F49$ unless the domain parameters and the key type are implicitly known and omitted.

Unrestricted public keys data objects are identified by the object identifier `id-ecTLVPublicKey` which is specified as follows:

```
id-ecTLVKeyFormat  OBJECT IDENTIFIER ::= { id-ecc keyType(2) 2}
id-ecTLVPublicKey  OBJECT IDENTIFIER ::= { id-ecTLVKeyFormat unrestricted(1) }
```

A restricted public key type for usage with dedicated ECC algorithms (for instance ECDSA, ECGDSA) may be defined in a later version of this specification.

Object	**Type**	**Symbol**	**Tag**
Algorithm	Object Identifier	–	$0x06$
Prime modulus	Integer	p	$0x81$
First coefficient	Integer	a	$0x82$
Second coefficient	Integer	b	$0x83$
Base point	Point	G	$0x84$
Order of the base point	Integer	n	$0x85$
Public Key	Point	P	$0x86$
Cofactor	Integer	h	$0x87$

Table 5.1.: Tags for elliptic curve public keys and domain parameters over $\mathbb{F}_p$.

Note: This guideline deviates from ISO 7816 in the encoding of elliptic curve points.

5.2. Signature Format

While this guideline supports the signature format specified by ANSI X9.62 [2] for ECDSA, the usage of the plain format is RECOMMENDED for all applications. The plain format MUST be used if the signature is generated or verified by a smartcard.

The signature algorithm and the signature format are identified by an `AlgorithmIdentifier`, the hash function to be used is either referenced directly by the object identifier or by the parameters of the `AlgorithmIdentifier`.

It is RECOMMENDED to reference the hash function to be used directly in the object identifier. In this case the parameters MAY be either absent or `null`. The recipient MUST be able to interpret both variants.[1]

[1] According to RFC 5480 [30], the parameters MUST be absent for algorithms identified by `ecdsa-with-SHAxxx`. According to ANSI X9.62 [2] the parameters SHOULD be absent, but implementations SHALL accept NULL parameters. This guideline explicitly allows both variants.

Note: Signature algorithms with hash functions SHA-1 or RIPEMD-160 SHALL NOT be used anymore and are only included for backwards compatibility. For current requirements and recommendations on suitable parameters, cf [11] and [9].

5.2.1. Plain Format

In plain format the signature (r, s) is encoded as octet string $R \parallel S$, i.e. as concatenation of the octet strings $R = \texttt{I2OS}(r, l)$ and $S = \texttt{I2OS}(s, l)$ with $l = \lceil \log_{256} n \rceil$. Thus, the signature has a fixed length of $2l$ octets.

To embed the signature in a `BIT STRING` the conversion routine `OS2BS` SHALL be used.

The signature algorithm including the hash function to be used and the signature format is identified by the following object identifiers:

5.2.1.1. ECDSA

```
ecdsa-plain-signatures  OBJECT IDENTIFIER ::= { id-ecc signatures(4) 1 }
ecdsa-plain-SHA1        OBJECT IDENTIFIER ::= { ecdsa-plain-signatures 1 }
ecdsa-plain-SHA224      OBJECT IDENTIFIER ::= { ecdsa-plain-signatures 2 }
ecdsa-plain-SHA256      OBJECT IDENTIFIER ::= { ecdsa-plain-signatures 3 }
ecdsa-plain-SHA384      OBJECT IDENTIFIER ::= { ecdsa-plain-signatures 4 }
ecdsa-plain-SHA512      OBJECT IDENTIFIER ::= { ecdsa-plain-signatures 5 }
ecdsa-plain-RIPEMD160   OBJECT IDENTIFIER ::= { ecdsa-plain-signatures 6 }
ecdsa-plain-SHA3-224    OBJECT IDENTIFIER ::= { ecdsa-plain-signatures 8 }
ecdsa-plain-SHA3-256    OBJECT IDENTIFIER ::= { ecdsa-plain-signatures 9 }
ecdsa-plain-SHA3-384    OBJECT IDENTIFIER ::= { ecdsa-plain-signatures 10 }
ecdsa-plain-SHA3-512    OBJECT IDENTIFIER ::= { ecdsa-plain-signatures 11 }
```

5.2.1.2. ECGDSA

```
ecgdsa-plain-signatures  OBJECT IDENTIFIER ::= { id-ecc signatures(4) 2 }
ecgdsa-plain-SHA224      OBJECT IDENTIFIER ::= { ecgdsa-plain-signatures 1 }
ecgdsa-plain-SHA256      OBJECT IDENTIFIER ::= { ecgdsa-plain-signatures 2 }
ecgdsa-plain-SHA384      OBJECT IDENTIFIER ::= { ecgdsa-plain-signatures 3 }
ecgdsa-plain-SHA512      OBJECT IDENTIFIER ::= { ecgdsa-plain-signatures 4 }
ecgdsa-plain-SHA3-224    OBJECT IDENTIFIER ::= { ecgdsa-plain-signatures 5 }
ecgdsa-plain-SHA3-256    OBJECT IDENTIFIER ::= { ecgdsa-plain-signatures 6 }
ecgdsa-plain-SHA3-384    OBJECT IDENTIFIER ::= { ecgdsa-plain-signatures 7 }
ecgdsa-plain-SHA3-512    OBJECT IDENTIFIER ::= { ecgdsa-plain-signatures 8 }
```

5.2.1.3. ECSDSA

```
ecsdsa-plain-signatures OBJECT IDENTIFIER ::= { id-ecc signatures(4) 4 }
ecsdsa-plain-SHA224     OBJECT IDENTIFIER ::= { ecsdsa-plain-signatures 1 }
ecsdsa-plain-SHA256     OBJECT IDENTIFIER ::= { ecsdsa-plain-signatures 2 }
ecsdsa-plain-SHA384     OBJECT IDENTIFIER ::= { ecsdsa-plain-signatures 3 }
ecsdsa-plain-SHA512     OBJECT IDENTIFIER ::= { ecsdsa-plain-signatures 4 }
ecsdsa-plain-SHA3-224   OBJECT IDENTIFIER ::= { ecsdsa-plain-signatures 5 }
ecsdsa-plain-SHA3-256   OBJECT IDENTIFIER ::= { ecsdsa-plain-signatures 6 }
ecsdsa-plain-SHA3-384   OBJECT IDENTIFIER ::= { ecsdsa-plain-signatures 7 }
ecsdsa-plain-SHA3-512   OBJECT IDENTIFIER ::= { ecsdsa-plain-signatures 8 }
```

5.2.2. X9.62 Format

In X9.62 format the ECDSA-signature (r, s) is encoded as ASN.1 structure with the following syntax:

```
ECDSA-Sig-Value ::= SEQUENCE {
  r  INTEGER,
  s  INTEGER
}
```

To embed the signature in a `BIT STRING` the DER encoded `ECDSA-Sig-Value` SHALL be the value of the bit string (including tag and length field).

The following object identifiers are defined in X9.62 [2].

```
id-ecSigType          OBJECT IDENTIFIER ::= { ansi-x9-62 signatures(4) }
ecdsa-with-Sha1       OBJECT IDENTIFIER ::= { id-ecSigType 1 }
ecdsa-with-Specified  OBJECT IDENTIFIER ::= { id-ecSigType 3 }
ecdsa-with-Sha224     OBJECT IDENTIFIER ::= { ecdsa-with-Specified 1 }
ecdsa-with-Sha256     OBJECT IDENTIFIER ::= { ecdsa-with-Specified 2 }
ecdsa-with-Sha384     OBJECT IDENTIFIER ::= { ecdsa-with-Specified 3 }
ecdsa-with-Sha512     OBJECT IDENTIFIER ::= { ecdsa-with-Specified 4 }
```

For `ecdsa-with-Specified` the object identifier of the hash function to be used MUST be provided as parameter in the `AlgorithmIdentifier`.

Note: X9.62 also provides `ecdsa-with-Recommended` which refers to ECDSA with "the natural size" hash function. This alternative is not supported by this guideline and MUST NOT be used.

5.3. Key Agreement

This section gives the object identifiers that SHALL be used for key agreement. Notice that the key agreement algorithms with block cipher 3DES SHOULD NOT be used in new applications and are only included for backward compatibility. For current requirements and recommendations on suitable parameters, cf [11] and [9].

5.3.1. ElGamal Key Agreement

The object identifier for the ElGamal key agreement protocol (ECKA-EG) is:

```
ecka-eg  OBJECT IDENTIFIER ::= { id-ecc key-establishment(5) 1 }
```

The object identifiers for ECKA-EG with specified key derivation functions are:

```
ecka-eg-X963KDF            OBJECT IDENTIFIER ::= { ecka-eg 1 }
ecka-eg-X963KDF-SHA1       OBJECT IDENTIFIER ::= { ecka-eg-X963KDF 1 }
ecka-eg-X963KDF-SHA224     OBJECT IDENTIFIER ::= { ecka-eg-X963KDF 2 }
ecka-eg-X963KDF-SHA256     OBJECT IDENTIFIER ::= { ecka-eg-X963KDF 3 }
ecka-eg-X963KDF-SHA384     OBJECT IDENTIFIER ::= { ecka-eg-X963KDF 4 }
ecka-eg-X963KDF-SHA512     OBJECT IDENTIFIER ::= { ecka-eg-X963KDF 5 }
ecka-eg-X963KDF-RIPEMD160  OBJECT IDENTIFIER ::= { ecka-eg-X963KDF 6 }
```

```
ecka-eg-SessionKDF          OBJECT IDENTIFIER ::= { ecka-eg 2 }
ecka-eg-SessionKDF-3DES     OBJECT IDENTIFIER ::= { ecka-eg-SessionKDF 1 }
ecka-eg-SessionKDF-AES128   OBJECT IDENTIFIER ::= { ecka-eg-SessionKDF 2 }
ecka-eg-SessionKDF-AES192   OBJECT IDENTIFIER ::= { ecka-eg-SessionKDF 3 }
ecka-eg-SessionKDF-AES256   OBJECT IDENTIFIER ::= { ecka-eg-SessionKDF 4 }
```

5.3.1.1. Message Format

The object identifiers beneath `SessionKDF` SHALL be associated with the message format as specified in Table 5.2 using the respective block cipher for encryption and authentication.

Object	Type	Symbol	Tag
Ephemeral Public Key	ECPoint	$\widetilde{P_{\mathsf{A}}}$	$0x97$
Padding indicator and ciphertext	Octet String	$\mathrm{E}(K_{Enc}, M)$	$0x87$
Integrity Protection	Octet String	$\mathrm{MAC}(K_{Mac}, M)$	$0x8\mathrm{E}$

Table 5.2.: Tags for messages protected by ECKA-EG.

Note: The message format is compatible to ISO 7816-4 Secure Messaging [24].

5.3.1.2. Authentication

For message authentication the block cipher SHALL be used in CMAC-mode [27] with $K_{Mac} = \mathrm{KDF}_{Session}(Z_{\mathsf{AB}}, 2)$.

5.3.1.3. Encryption

For message encryption the block cipher SHALL be used in CBC-mode [16] with key $K_{Enc} = \mathrm{KDF}_{Session}(Z_{\mathsf{AB}}, 1)$ and $IV = \mathrm{E}(K_{Enc}, 0)$.

5.3.2. Anonymous Diffie-Hellman Key Agreement

The object identifier for the anonymous Diffie-Hellman key agreement protocol (ECKA-DH) is:

```
ecka-dh  OBJECT IDENTIFIER ::= { id-ecc key-establishment(5) 2 }
```

The object identifiers for ECKA-DH with specified key derivation functions are:

```
ecka-dh-X963KDF             OBJECT IDENTIFIER ::= { ecka-dh 1 }
ecka-dh-X963KDF-SHA1        OBJECT IDENTIFIER ::= { ecka-dh-X963KDF 1 }
ecka-dh-X963KDF-SHA224      OBJECT IDENTIFIER ::= { ecka-dh-X963KDF 2 }
ecka-dh-X963KDF-SHA256      OBJECT IDENTIFIER ::= { ecka-dh-X963KDF 3 }
ecka-dh-X963KDF-SHA384      OBJECT IDENTIFIER ::= { ecka-dh-X963KDF 4 }
ecka-dh-X963KDF-SHA512      OBJECT IDENTIFIER ::= { ecka-dh-X963KDF 5 }
ecka-dh-X963KDF-RIPEMD160   OBJECT IDENTIFIER ::= { ecka-dh-X963KDF 6 }

ecka-dh-SessionKDF          OBJECT IDENTIFIER ::= { ecka-dh 2 }
ecka-dh-SessionKDF-3DES     OBJECT IDENTIFIER ::= { ecka-dh-SessionKDF 1 }
ecka-dh-SessionKDF-AES128   OBJECT IDENTIFIER ::= { ecka-dh-SessionKDF 2 }
ecka-dh-SessionKDF-AES192   OBJECT IDENTIFIER ::= { ecka-dh-SessionKDF 3 }
ecka-dh-SessionKDF-AES256   OBJECT IDENTIFIER ::= { ecka-dh-SessionKDF 4 }
```

5.4. PACE

This section gives the object identifiers that SHALL be used for PACE. Notice that the object identifier with block cipher 3DES SHOULD NOT be used in new applications and is only included for backward compatibility. For current requirements and recommendations on suitable parameters, cf [11] and [9].

The object identifiers for the PACE protocol with generic mapping are:

```
id-PACE-KA  OBJECT IDENTIFIER ::= { id-ecc key-establishment(5) 3 }

id-PACE-KA-GM                     OBJECT IDENTIFIER ::= { id-PACE-KA 1 }
id-PACE-KA-GM-SessionKDF-3DES     OBJECT IDENTIFIER ::= { id-PACE-KA-GM 1 }
id-PACE-KA-GM-SessionKDF-AES-128  OBJECT IDENTIFIER ::= { id-PACE-KA-GM 2 }
id-PACE-KA-GM-SessionKDF-AES-192  OBJECT IDENTIFIER ::= { id-PACE-KA-GM 3 }
id-PACE-KA-GM-SessionKDF-AES-256  OBJECT IDENTIFIER ::= { id-PACE-KA-GM 4 }
```

Here, public keys SHALL be structured according to section 5.1 and nonces SHALL be encoded as octet strings. The used encoding for the elliptic curve points MUST be negotiated in advance by the protocol parties.

5.4.1. PACE on Smartcards

For smartcards, Part 2 of the Technical Guideline [10] gives a detailed specification of PACE. Implementations according to that specification SHALL use the corresponding object identifiers.

```
id-PACE  OBJECT IDENTIFIER ::= { bsi-de protocols(2) smartcards(2) 4 }
```

For PACE with generic mapping and specified symmetric ciphers, [10] defines the following object identifiers:

```
id-PACE-ECDH-GM                   OBJECT IDENTIFIER ::= { id-PACE 2 }
id-PACE-ECDH-GM-3DES-CBC-CBC      OBJECT IDENTIFIER ::= { id-PACE-ECDH-GM 1 }
id-PACE-ECDH-GM-AES-CBC-CMAC-128  OBJECT IDENTIFIER ::= { id-PACE-ECDH-GM 2 }
id-PACE-ECDH-GM-AES-CBC-CMAC-192  OBJECT IDENTIFIER ::= { id-PACE-ECDH-GM 3 }
id-PACE-ECDH-GM-AES-CBC-CMAC-256  OBJECT IDENTIFIER ::= { id-PACE-ECDH-GM 4 }
```

Here, elliptic curve points are always represented in uncompressed encoding.

6. Standardized Domain Parameters

While this guideline supports domain parameters standardized by X9.62 [2] it is RECOMMENDED to use the domain parameters of the ECC Brainpool working group, which are standardized by the IETF RFC 5639 [31]:

```
BrainpoolCurveNames CURVES ::= {
  {ID brainpoolP160r1} | {ID brainpoolP160t1} |
  {ID brainpoolP192r1} | {ID brainpoolP192t1} |
  {ID brainpoolP224r1} | {ID brainpoolP224t1} |
  {ID brainpoolP256r1} | {ID brainpoolP256t1} |
  {ID brainpoolP320r1} | {ID brainpoolP320t1} |
  {ID brainpoolP384r1} | {ID brainpoolP384t1} |
  {ID brainpoolP512r1} | {ID brainpoolP512t1}
  ...
}
```

The identifier `brainpoolP`Lrj and `brainpoolP`Ltj depend on two parameters:

1. The integer L denotes the bit length of the prime p which is also the bit length of the order n of the base point.

2. The integer j denotes the j-th elliptic curve defined by Brainpool. Currently, only curves for $j = 1$ are specified.

The curve with curve identifier name `brainpoolP`Lrj is $\mathbb{F}_p$-isomorphic to the twisted curve with curve name `brainpoolP`Ltj with coefficient $a = -3 \bmod p$.

Note: The subset of domain parameters with $L \leq 224$ is only included for backward-compatibility with former versions of this Technical Guideline. For current requirements and recommendations on suitable parameters and key lengths, cf [11] and [9].

The object identifier `versionOne` represents the tree containing the object identifiers for each set of elliptic curve domain parameters as specified in [31]. The object identifier has the following value:

```
ecStdCurvesAndGeneration OBJECT IDENTIFIER ::= {
  iso(1) identified-organization(3) teletrust(36) algorithm(3)
  signature-algorithm(3) ecSign(2) ecStdCurvesAndGeneration(8)
}

ellipticCurve    OBJECT IDENTIFIER ::= { ecStdCurvesAndGeneration 1 }

versionOne       OBJECT IDENTIFIER ::= { ellipticCurve 1 }

brainpoolP160r1  OBJECT IDENTIFIER ::= { versionOne 1 }
brainpoolP160t1  OBJECT IDENTIFIER ::= { versionOne 2 }
brainpoolP192r1  OBJECT IDENTIFIER ::= { versionOne 3 }
```

```
brainpoolP192t1  OBJECT IDENTIFIER ::= { versionOne 4 }
brainpoolP224r1  OBJECT IDENTIFIER ::= { versionOne 5 }
brainpoolP224t1  OBJECT IDENTIFIER ::= { versionOne 6 }
brainpoolP256r1  OBJECT IDENTIFIER ::= { versionOne 7 }
brainpoolP256t1  OBJECT IDENTIFIER ::= { versionOne 8 }
brainpoolP320r1  OBJECT IDENTIFIER ::= { versionOne 9 }
brainpoolP320t1  OBJECT IDENTIFIER ::= { versionOne 10 }
brainpoolP384r1  OBJECT IDENTIFIER ::= { versionOne 11 }
brainpoolP384t1  OBJECT IDENTIFIER ::= { versionOne 12 }
brainpoolP512r1  OBJECT IDENTIFIER ::= { versionOne 13 }
brainpoolP512t1  OBJECT IDENTIFIER ::= { versionOne 14 }
```

Appendix

A. The Signature Algorithm – EC-KCDSA (Informative)

For informative reasons, this appendix describes the Elliptic Curve Korean Certificate Based Digital Signature Algorithm in conformance to [19]. The algorithm uses the public key $P'_{\mathsf{A}} := [d_{\mathsf{A}}^{-1} \bmod n]G$. Let the value z_{A} be defined as l leftmost bits of the sequence $\texttt{FE2OS}(x_{P_{\mathsf{A}}}) \parallel \texttt{FE2OS}(y_{P_{\mathsf{A}}})$. In the following, $l(b)$ denotes the length of a bit string b.

A.1. Signature Algorithm

A proceeds as follows to generate the EC-KCDSA signature (r, s) on a message M.

Input: The following information is required as input:

1. A's private key d_{A} and the elliptic curve domain parameters (p, a, b, G, n, h).
2. The hash value z_{A} of the A's certification data.
3. The message M to be signed.

Output: The EC-KCDSA signature (r, s) over M.

Actions: The following actions are performed:

0. $e = \mathrm{H}_\tau(z_{\mathsf{A}} \parallel M)$
1. $k = \mathrm{RNG}(\{1, 2, \ldots, n-1\})$
2. $Q = [k]G$
3. $c = \mathrm{H}_\tau(\texttt{FE2OS}(x_Q))$
4. $r = \texttt{OS2I}(c)$
5. $w = \texttt{OS2I}(c \oplus e)$
   If $w \geq n$, set $w = w - n$.
6. $s = (k - w)d_{\mathsf{A}} \bmod n$
   If $s = 0$ goto 1.
7. Output (r, s)

Note: The signature depends on the random number k. If A signs the same document M twice, both signatures differ with a very high probability.

A.2. Verification Algorithm

B proceeds as follows to verify the received EC-KCDSA signature (r, s) on M.

Input: The following input is necessary:

1. A's authentic public key P'_{A} and the domain parameters (p, a, b, G, n, h).
2. The hash value z_{A} of the A's certification data.
3. The signed message M.

4. The EC-KCDSA signature (r, s).

Output: `True`, if the signature is valid, and `False` otherwise.

Actions: The following actions are performed:

1. Verify that $s \in \{1, 2, \ldots, n-1\}$ and $l(\texttt{OS2BS}(\texttt{I2OS}(r))) \leq \tau$.
   If this is not the case, output `False` and terminate.
2. $e = \mathrm{H}_\tau(z_\mathsf{A} \parallel M)$
3. $w = \texttt{OS2I}(\texttt{I2OS}(r) \oplus e) \mod n$
4. $Q = [w]G + [s]P_\mathsf{A}$
5. $c = \texttt{FE2OS}(x_Q))$
6. $v = \mathrm{H}_\tau(c)$
7. Output `True` if $v = r$, and `False` otherwise.

Bibliography

[1] BSI. Appendix to AIS 20/31. *A proposal for: Functionality classes for random number generators*. Federal Office for Information Security, 2011. Available at `https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Zertifizierung/Interpretationen/AIS20_Functionality_classes_for_random_numbers_generators.pdf`

[2] ANSI X9.62. *Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA)*, 2005.

[3] ANSI X9.63. *Public Key Cryptography for the Financial Services Industry: Key Agreement and Key Transport Using Elliptic Curve Cryptography*, 2011.

[4] BSI. Appendix to AIS 46. *Minimum Requirements for Evaluating Side-channel Attack Resistance of Elliptice Curve Implementations*. Available at `https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Zertifizierung/Interpretationen/AIS_46_ECCGuide_e_pdf.pdf`

[5] R. M. Avanzi, H. Cohen, C. Doche, G. Frey, T. Lange, K. Nguyen, and F. Vercauteren. *Handbook of Elliptic and Hyperelliptic Curve Cryptography*. Chapman & Hall/CRC, 2005.

[6] J. Bender, M. Fischlin, D. Kügler. Security Analysis of the PACEv2 Key Agreement Protocol. ISC 2009. *Lecture Notes in Computer Science*. Springer Verlag, 2009.

[7] I. F. Blake, G. Seroussi, N. P. Smart. Elliptic Curve Cryptography. Cambridge University Press, 1999.

[8] S. Bradner. *Key words for use in RFCs to indicate requirement levels (RFC2119)*, 1999. Available at `http://www.ietf.org/rfc/rfc2119.txt`.

[9] BSI TR-02102. *Kryptographische Verfahren: Empfehlungen und Schlüssellängen*, Version 1.0. Federal Office for Information Security, Available at `https://www.bsi.bund.de/EN/eID-TR`.

[10] BSI TR-03110. *Advanced Security Mechanisms for Machine Readable Travel Documents and eIDAS Token*, Parts 1-4. Federal Office for Information Security. Available at `https://www.bsi.bund.de/EN/eID-TR`.

[11] BSI TR-03116. *Kryptographische Vorgaben für Projekte der Bundesregierung* Federal Office for Information Security. Available at `https://www.bsi.bund.de/EN/eID-TR`.

[12] Federal Information Processing Standards Publication 180-4 (FIPS PUB 180-4). *Secure Hash Standard (SHS)*. Available at `https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf`.

[13] Federal Information Processing Standards Publication 202 (FIPS PUB 202). *SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions*. Available at `https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf`.

[14] IEEE P1363. *Standard Specifications for Public Key Cryptography*, 2000.

[15] IEEE P1363a. *Standard Specifications for Public Key Cryptography - Amendment 1: Additional Techniques*, 2004.

[16] ISO/IEC 10116-2006. *Information technology – Security techniques – Modes of operation for an n-bit block cipher*, 2006.

[17] ISO/IEC 10118-3-2004. *Information technology – Security techniques – Hash functions – Part 3: Dedicated hash functions*, 2003.

[18] ISO/IEC 11770-3-2008. *Information technology – Security techniques – Key management – Part 3: Mechanisms using asymmetric techniques*, 2008.

[19] ISO/IEC 14888-3-2006. *Information technology – Security techniques – Digital signatures with appendix – Part 3: Discrete logarithm based Mechanisms*, 2006.

[20] ISO/IEC 14888-3-2006/Amd 1-2010. *Elliptic Curve Russian Digital Signature Algorithm, Schnorr Digital Signature Algorithm, Elliptic Curve Schnorr Digital Signature Algorithm, and Elliptic Curve Full Schnorr Digital Signature Algorithm*, 2010.

[21] ISO/IEC 15946-1-2008. *Information technology – Security techniques – Cryptographic techniques based on elliptic curves – Part 1: General*, 2008.

[22] ISO/IEC 15946-2-2002 (Withdrawn). *Information technology – Security techniques – Cryptographic techniques based on elliptic curves – Part 2: Digital signatures*, 2002.

[23] ISO/IEC 15946-3-2002 (Withdrawn). *Information technology – Security techniques – Cryptographic techniques based on elliptic curves – Part 3: Key establishment*, 2002.

[24] ISO/IEC 7816-4-2013. *Identification cards – Integrated circuit cards – Part 4: Organization, security and commands for interchange.*

[25] ISO/IEC 7816-8-2004. *Identification cards – Integrated circuit cards – Part 8: Commands for security operations*, 2004.

[26] A. Menezes. *Elliptic Curve Public Key Cryptosystems*. Kluwer Academic Publishers, 1993.

[27] NIST. *Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication*, Special Publication 800-38B, 2005. Available at `http://csrc.nist.gov/publications/nistpubs/800-38B/SP_800-38B.pdf`.

[28] P. Nguyen and I. Shparlinski. The insecurity of the elliptic curve signature algorithm with partially known nonces. *Designs, Codes and Cryptography*, 30(2):201–217, 2003.

[29] W. Polk, R. Housley, and L. Bassham. *Algorithms and Identifiers for the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile (RFC3279)*, 2002. Available at `http://www.ietf.org/rfc/rfc3279.txt`.

[30] S. Turner, D. Brown, K. Yiu, R. Housley and T. Polk. *Elliptic Curve Cryptography Subject Public Key Information (RFC5480)*, 2009. Available at `http://www.ietf.org/rfc/rfc5480.txt`.

[31] M. Lochter, J. Merkle. *Elliptic Curve Cryptography (ECC) Brainpool Standard Curves and Curve Generation (RFC5639)*, 2010. Available at `http://www.ietf.org/rfc/rfc5639.txt`.

[32] C. P. Schnorr. Efficient Signature Generation on Smart Cards. Journal of Cryptology 4(4), pp. 161-174, 1991.

[33] G. Neven, N.P. Smart, B. Warinschi. Hash function requirements for Schnorr signatures. Journal of Mathematical Cryptology 3(1), 2009.

[34] X. Wang, Y.L. Yin, and H. Yu. Collision search attacks on SHA-1. In *Proceedings of Crypto 2005*. Springer Verlag, 2005.