ETSI TS 103 532 V1.1.1 (2018-03)

TECHNICAL SPECIFICATION

CYBER; Attribute Based Encryption for Attribute Based Access Control

Reference
DTS/CYBER-0025

Keywords
access control, privacy

Contents

Intellectual Property Rights

Foreword

This Technical Specification (TS) has been produced by ETSI Technical Committee Cyber Security (CYBER).

Modal verbs terminology

In the present document "**shall**", "**shall not**", "**should**", "**should not**", "**may**", "**need not**", "**will**", "**will not**", "**can**" and "**cannot**" are to be interpreted as described in clause 3.2 of the ETSI Drafting Rules (Verbal forms for the expression of provisions).

"**must**" and "**must not**" are **NOT** allowed in ETSI deliverables except when used in direct citation.

1 Scope

The present document specifies trust models, functions and protocols using attribute based encryption as a foundation of an attribute based access control scheme. It covers both the Ciphertext-Policy (CP-ABE) and Key-Policy (KP-ABE) variants of Attribute-Based Encryption.

The specifications address the following aspects:

- Identification of an ABE scheme covering both the Ciphertext-Policy and Key-Policy variants
- Definition of interactions between the data sources, the service providers and the authority releasing attributes and key material
- Mechanisms for keys, policies, and attributes distribution
- Mechanisms for secret key expiration and revocation
- Definition of semantics for a basic set of attributes to ensure interoperability
- Mapping to a standard Public Key Infrastructure X.509
- Mapping to a standard assertion protocol (SAML)
- Definition of a policy schema for data access control
- Identification of limitations compared to traditional ABAC features
- Translation rules to XACML
- Definition of new protocol bindings when existing bindings do not cover the deployment scenario (e.g. a CoAP binding for the IoT case)

2 References

2.1 Normative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

Referenced documents which are not found to be publicly available in the expected location might be found at https://docbox.etsi.org/Reference.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long term validity.

The following referenced documents are necessary for the application of the present document.

[1] Bureau international des poids et mesures, "Le Système international d'unités (SI) / The International System of Units (SI)".

[2] National Institute of Standards and Technology: NIST SP 800-56B Revision 1 "Recommendation for Pair-Wise Key-Establishment Schemes Using Integer Factorization Cryptography".

[3] Federal Information Processing Standards Publication (FIPS) 197: "Advanced Encryption Standard".

[4] IETF RFC 4648: "The Base16, Base32, and Base64 Data Encodings".

[5] OASIS: "Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0".

[6] OASIS: "Bindings for the OASIS Security Assertion Markup Language (SAML) V2.0".

[7] OASIS: "XACML v3.0 Core and Hierarchical Role Based Access Control (RBAC) Profile Version 1.0".

[8] W3C: "XML Schema Part 2: Datatypes".

Available at: https://www.w3.org/TR/xmlschema-2/.

[9] OASIS: "eXtensible Access Control Markup Language (XACML) Version 3.0".

[10] ANSI INCITS 4-1986[R2012]: "Information Systems - Coded Character Sets - 7-Bit American National Standard Code for Information Interchange (7-Bit ASCII)".

[11] ISO/IEC 8601:2004: "Data elements and interchange formats - Information interchange - Representation of dates and times".

[12] W3C: "XML Encryption Syntax and Processing Version 1.1".

Available at: https://www.w3.org/TR/xmlenc-core1/.

[13] IETF RFC 5234: "Augmented BNF for Syntax Specifications: ABNF".

[14] ANSI/IEEE 754^{TM}-1985: "IEEE Standard for Binary Floating-Point Arithmetic".

[15] IETF RFC 7252: "The Constrained Application Protocol (CoAP)".

[16] IETF RFC 7959: "Block-Wise Transfer in CoAP".

2.2 Informative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long term validity.

The following referenced documents are not necessary for the application of the present document but they assist the user with regard to a particular subject area.

[i.1] ISO/IEC 29100:2011: "Information technology - Security techniques - Privacy framework".

[i.2] National Institute of Standards and Technology NIST SP 800-122: "Guide to Protecting the Confidentiality of Personally Identifiable Information (PII)".

[i.3] ETSI TS 103 458: "CYBER; Application of Attribute-Based Encryption (ABE) for data protection on smart devices, cloud and mobile services".

[i.4] ISO/IEC 18031:2011: "Information technology - Security techniques - Random bit generation".

[i.5] IETF RFC 5280: "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile".

[i.6] J. Bethencourt, A. Sahai, B. Waters: "Ciphertext-policy attribute-based encryption". Proceedings of the 2007 IEEE Symposium on Security and Privacy, SP'07, pages 321-334. Washington, DC, USA, 2007. IEEE Computer Society.

[i.7] ETSI TR 187 010: "Telecommunications and Internet converged Services and Protocols for Advanced Networking (TISPAN); NGN Security; Report on issues related to security in identity imanagement and their resolution in the NGN".

[i.8] ISO/IEC 17789:2014: "Information technology - Cloud computing - Reference architecture".

[i.9] ISO/IEC 19944:2017 "Information technology - Cloud computing - Cloud services and devices: Data flow, data categories and data use".

[i.10] ISO/IEC 17788: "Information technology - Cloud computing - Overview and vocabulary".

[i.11] Herranz, J., Laguillaumie, F., & Ràfols, C. (2010). Constant Size Ciphertexts in Threshold Attribute-Based Encryption. Public Key Cryptography, (p. 19-34).

[i.12] N. Attrapadung, H. Imai: "Attribute-Based Encryption Supporting Direct/Indirect Revocation Modes". Proceedings of the 12th IMA International Conference on Cryptography and Coding. Pages 278 - 300. Cirencester, UK - December 15 - 17, 2009. Springer-Verlag.

[i.13] Ostrovsky, R., Sahai, A., & Waters, B. (2007). Attribute-based encryption with non-monotonic access structures. ACM Conference on Computer and Communications Security, (p. 195-203).

[i.14] Attrapadung, N., Libert, B., & de Panafieu, E. (2011). Expressive Key-Policy Attribute-Based Encryption with Constant-Size Ciphertexts. Public Key Cryptography, (p. 90-108).

[i.15] Lewko, A. B., Sahai, A., & Waters, B. (2010). Revocation Systems with Very Small Private Keys. IEEE Symposium on Security and Privacy, (p. 273-285).

[i.16] ETSI TS 118 112: "oneM2M; Base Ontology (oneM2M TS-0012 version 2.0.0 Release 2)".

[i.17] ISO/IEC 18033-1:2015: "Information technology - Security techniques - Encryption algorithms - Part 1: General".

[i.18] ISO/IEC 18033-5:2015: "Information technology - Security techniques - Encryption algorithms - Part 5: Identity based ciphers".

[i.19] ISO/IEC 24760-1:2011: "Information technology - Security techniques - A framework for identity management - Part 1: Terminology and concepts".

[i.20] Yannis Rouselakis, Brent Waters: "Efficient Statically-Secure Large-Universe Multi-Authority Attribute-Based Encryption". Financial Cryptography 2015: 315-332.

[i.21] IETF RFC 4119: "A Presence-based GEOPRIV Location Object Format".

[i.22] IETF RFC 2253: "Lightweight Directory Access Protocol (v3): UTF-8 String Representation of Distinguished Names".

[i.23] IETF RFC 2821: "Simple Mail Transfer Protocol".

[i.24] IETF RFC 2732: "Format for Literal IPv6 Addresses in URL's".

[i.25] IETF RFC 2396: "Uniform Resource Identifiers (URI): Generic Syntax".

[i.26] W3C: "XML Path Language (XPath)".

Available at: https://www.w3.org/TR/xpath/.

[i.27] IETF RFC 3261: "SIP: Session Initiation Protocol".

[i.28] ISO/IEC 15946-1: 'Information technology - Security techniques - Cryptographic techniques based on elliptic curves - Part 1: General'.

[i.29] IETF RFC 822: 'Standard for the Format of ARPA Internet Text Messages'.

[i.30] Recommendation ITU-T X.520: 'Information technology - Open Systems Interconnection - The Directory: Selected attribute types'.

3 Definitions and abbreviations

3.1 Definitions

For the purposes of the present document, the following terms and definitions apply:

ABE authority: ABE entity that stores the master secret key and gives out secret keys

ABKEM universe: set of attributes in which the number of attributes can be a linear (small-universe) or exponential (large-universe) function of the system's security strength

ABKEM universe regeneration: procedure by which an entirely new ABKEM universe is generated, with redistribution of new public key, new master secret key, and new secret keys

app: "software application", typically running on a user's device platform

assertion: statement made by an authority about a property of an entity, typically for - but not restricted to - access control decisions

asymmetric encryption system: encryption system based on asymmetric cryptographic techniques whose public transformation is used for encryption and whose private transformation is used for decryption, see ISO/IEC 18033-1 [i.17]

attack: algorithm that performs computations and makes queries to the encryption algorithm for the encryption and/or for the decryption of adaptively chosen texts under a single secret key, with the purpose of recovering either the unknown plaintext for a given ciphertext, which may be adaptively chosen but for which a decryption query is not issued, or a secret key, see ISO/IEC 18033-1 [i.17]

attack cost: ratio of the average complexity of the attack algorithm measured in terms of the number of calls to the encryption algorithm made by the attack to the probability of success of the attack, see ISO/IEC 18033-1 [i.17]

attribute: characteristic or property of an entity that can be used to describe its state, appearance, or other aspects, see ISO/IEC 24760-1 [i.19]

Attribute-Based Encryption (ABE) system: asymmetric encryption system which is either a CP-ABE system, a KP-ABE system or a combination of both

attribute universe: set of attributes

cloud platform provider: cloud service provider providing identity management services and interfaces (e.g. API, marketplace, etc.) for third party applications using the platform services

cloud platform user: cloud service user consuming one or more platform services

cloud service customer: individual or organization consuming one or more cloud services provided by a cloud service provider

cloud service partner: individual or organization providing support to the provisioning of cloud services by the cloud service provider, or to the consumption of cloud service by the cloud service customer

cloud service provider: individual or organization providing cloud services to one or more cloud service customers

cloud service user: individual consuming one or more cloud services using a particular device

ciphertext: data which has been transformed to hide its information content, see ISO/IEC 18033-1 [i.17]

Ciphertext-Policy Attribute-Based Encryption (CP-ABE) system: asymmetric encryption system where secret keys are derived from a set of attributes and ciphertexts are derived from a policy on attributes

data consumer: natural or legal person, public authority, agency or any other body accessing data for a given purpose

data controller: natural or legal person, public authority, agency or any other body which alone or jointly with others determines the purposes and means of the processing of personal data

data processor: natural or legal person, public authority, agency or any other body which processes personal data on behalf of the controller

data subject: identifiable person, i.e. a person who can be identified, directly or indirectly, in particular by reference to an identification number or to one or more factors specific to his physical, physiological, mental, economic, cultural or social identity

decryption: reversal of a corresponding encryption, see ISO/IEC 18033-1 [i.17]

decryptor: entity which decrypts ciphertexts, see ISO/IEC 18033-1 [i.17]

encryption: (reversible) transformation of data by a cryptographic algorithm to produce a ciphertext, i.e. to hide the information content of the data, see ISO/IEC 18033-1 [i.17]

encryptor: entity which encrypts ciphertexts, see ISO/IEC 18033-1 [i.17]

entity: item inside or outside an information and communication technology system, such as a person, an organization, a device, a subsystem, or a group of such items that has recognizable distinct existence, see ISO/IEC 24760-1 [i.19]

home network: central source for mobility services to the subscriber. The subscriber has a direct subscription with the home network

identity: set of attributes related to an entity, see ISO/IEC 24760-1 [i.19]

Key-Policy Attribute-Based Encryption (KP-ABE) system: asymmetric encryption system where ciphertexts are derived from a set of attributes and secret keys are derived from a policy on attributes

match: policy statement in the access structure is said to match when the attributes in the annotation allow said statement to be evaluated to true in the logical sense

master public key: public value uniquely determined by the corresponding master secret key [i.18]

master secret key: secret value used by the secret key generator to compute secret keys for an ABE mechanism

personal data: any information relating to an identified or identifiable natural person ('data subject')

Personally Identifiable Information (PII): any information that (a) can be used to identify the PII principal to whom such information relates, or (b) is or might be directly or indirectly linked to a PII principal

NOTE 1: To determine whether a PII principal is identifiable, account can be taken of all the means which can reasonably be used by the privacy stakeholder holding the data, or by any other party, to identify that natural person (ISO/IEC 29100 [i.1]).

NOTE 2: In the US, according to NIST SP 800-122 [i.2]: any information about an individual maintained by an agency, including any information that can be used to distinguish or trace an individual's identity, such as name, social security number, date and place of birth, mother's maiden name, or biometric records; and any other information that is linked or linkable to an individual, such as medical, educational, financial, and employment information.

PII controller: privacy stakeholder that determines the purposes and means for processing personally identifiable information (PII) other than natural persons who use data for personal purposes, see [i.1]

PII principal: natural person to whom the personally identifiable information (PII) relates, see [i.1]

PII processor: privacy stakeholder that processes personally identifiable information (PII) on behalf of and in accordance with the instructions of a PII controller, see [i.1]

plaintext: unencrypted information, see ISO/IEC 18033-1 [i.17]

platform Provider: service provider providing services necessary to support a platform

secret key generator: entity or function which generates a set of secret keys [i.18]

policy on attributes: boolean predicate on attributes combining equality and/or inequality tests

policy statement: elementary test in an access control policy (e.g. "A < B")

processing of PII: operation or set of operations performed upon personally identifiable information (PII), see [i.1]

NOTE: Examples of processing operations of PII include, but are not limited to, the collection, storage, alteration, retrieval, consultation, disclosure, anonymization, pseudonymization, dissemination or otherwise making available, deletion or destruction of PII) [i.1].

random tape: source of (pseudo-)random bits

security strength: number associated with the amount of work (e.g. the number of operations) that is required to break a cryptographic algorithm or system, see ISO/IEC 18033-1 [i.17]

serving network: home network or visited network the user equipment is connected to

set up: process by which the system parameters of an ABE mechanism are selected

set up algorithm: process which generates a master secret key and the corresponding master public key, together with some part of the system parameters, see ISO/IEC 18033-5 [i.18]

setup party: entity that specifies the security parameter and handles the setup of the ABE and ABKEM system

subscriber User Equipment (UE): any device allowing a user access to network services

system parameters: parameters for cryptographic computation including a selection of a particular cryptographic scheme or function from a family of cryptographic schemes of functions, or from a family of mathematical spaces, see ISO/IEC 18033-5 [i.18]

top-level policy statement: policy statement that is evaluated in all branches of the access structure

trust: level of confidence in the reliability and integrity of an entity to fulfil specific responsibilities

Trusted Authority (TA): ABE authority entitled to generate the public key PK and the corresponding secret keys according to a selected large universe ABE scheme

visited network: any network that interacts with the home network to provide mobility services to the subscriber terminal

3.2 Abbreviations

For the purposes of the present document, the following abbreviations apply:

AAD	Additional Authentic Data
ABAC	Attribute-Based Access Control
ABE	Attribute-Based Encryption
ABEA	Attribute-Based Encryption Authority
ABFN	Augmented Backus-Naur Form
ABKEM	Attribute-Based Key-Encapsulation Mechanism
ABNF	Augmented Backus-Naur Form
AES	Advanced Encryption Standard
AP	Access Policy
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange.
BDH	Bilinear Diffie-Hellman
CA	Certification Authority
C_{AE}	Ciphertext for Authenticated Encryption
CCA	Chosen-Ciphertext Attack
CPA	Chosen-Plaintext Attack
CP-ABE	Ciphertext Policy ABE
CP-ABKEM	Ciphertext Policy ABKEM
CRL	Certificate Revocation List
DBDDH	Decisional Bilinear DDH
DBDHE	Decisional Bilinear Diffie-Hellman Exponent
DDH	Decisional Diffie-Hellman
DNS	Domain Name System
DPP	Device Platform Provider

ECDSA	Elliptic Curve Digital Signature Algorithm
GGM	Generic Group Model
GML	Geography Markup Language
IBBE	Identity-Based Broadcast Encryption
K_{AE}	Symmetric Key for Authenticated Encryption
KEM	Key Encapsulation Mechanism
KP	Key-Policy
KP-ABE	Key Policy ABE
KP-ABKEM	Key Policy ABKEM
MEBDH	Multi-Exponent BDH
MPK	Master Public Key
MSK	Master Secret Key
MSP	Monoton Span Program
NIST	National Institute of Standards and Technology
OR	OR (bitwise operator)
PIDF	Presence Information Data Format
PII	Personally Identifiable Information
PKI	Public Key Infrastructure
PKIX	Public-Key Infrastructure (X.509)
PRG	Pseudo Random Generator
RBAC	Role Based Access Control
RSA	Rivest–Shamir–Adleman
SAML	Security Assertion Markup Language
SIP	Session Initiation Protocol
SK_A	Secret Key for A
SM	Standard Model
UINT	Unsigned INTeger
URN	Uniform Resource Name
UTC	Coordinated Universal Time
XACML	eXtensible Access Control Markup Language
XOR	eXclusive OR (bitwise operator)

4 Attribute-Based Encryption Toolkit

4.1 CPA-secure ciphertext-policy and key-policy attribute-based key-encapsulation mechanisms

4.1.1 Overview

Ciphertext-policy attribute-based key-encapsulation and key-policy attribute-based key-encapsulation mechanisms (CP-ABKEM and KP-ABKEM) are specified in clauses 4.1.2 and 4.1.3.

4.1.2 Ciphertext-policy ABKEM

A ciphertext-policy attribute-based key-encapsulation mechanism (CP-ABKEM) shall associate secret keys with attributes from an attribute universe and ciphertexts with access policies.

The following flow is specified:

- **System setup:**
    - The setup party shall specify a security parameter k and shall output system parameters $PARAMS_{CP-ABKEM}$, a master public key $MPK_{CP-ABKEM}$, and a master secret key $MSK_{CP-ABKEM}$. ($MSK_{CP-ABKEM}$ shall contain $MPK_{CP-ABKEM}$ and $MPK_{CP-ABKEM}$ shall contain $PARAMS_{CP-ABKEM}$.)

- **Secret-key generation:**
  - The ABE authority shall take as input the master secret key $MSK_{CP\text{-}ABKEM}$, and an attribute set A from the attribute universe, and shall output a secret key SK_A. (SK_A shall contain $MPK_{CP\text{-}ABKEM}$.)
- **Symmetric-key encapsulation:**
  - The encryptor shall take as input the master public key $MPK_{CP\text{-}ABKEM}$, an access policy AP, and a random tape R, and shall output a symmetric key $K_{CP\text{-}ABKEM}$ and a ciphertext C_{AP} associated with AP. (AP shall be extractable from C_{AP}. R shall be a uniformly random bitstring from $\{0,1\}^k$.)
- **Symmetric-key decapsulation:**
  - The decryptor shall take as input a secret key SK_A associated with an attribute set A and a ciphertext C_{AP} associated with an access policy AP, and shall output a symmetric key $K_{CP\text{-}ABKEM}$ if the A satisfies AP, else output an error *err* that shall indicate that decapsulation has failed.

4.1.3 Key-policy ABKEM

The key-policy attribute-based key-encapsulation mechanism (KP-ABKEM) shall associate secret keys with access policies and ciphertexts with attributes from an attribute universe.

The following flow is specified:

- **System setup:**
  - The setup party shall specify a security parameter k and shall output system parameters $PARAMS_{KP\text{-}ABKEM}$, a master public key $MPK_{KP\text{-}ABKEM}$, and a master secret key $MSK_{KP\text{-}ABKEM}$. ($MSK_{KP\text{-}ABKEM}$ shall contain $MPK_{KP\text{-}ABKEM}$.)
- **Secret-key generation:**
  - The ABE authority shall take as input the master secret key $MSK_{KP\text{-}ABKEM}$ and an access policy AP, and shall output a secret key SK_{AP}. (SK_{AP} shall contain $MPK_{KP\text{-}ABKEM}$.)
- **Symmetric-key encapsulation:**
  - The encryptor shall take as input the master public key $MPK_{KP\text{-}ABKEM}$, an attribute set A from the attribute universe, and a random tape R, and shall output a symmetric key $K_{KP\text{-}ABKEM}$ and a ciphertext C_A associated with A. (A shall explicitly be extractable from C_A. R shall be a uniformly random bitstring from $\{0,1\}^k$.)
- **Symmetric-key decapsulation:**
  - The decryptor shall take as input a secret key SK_{AP} associated with an access policy AP, and a ciphertext C_A associated with an attribute set A, and shall output a symmetric key $K_{KP\text{-}ABKEM}$ if the A satisfy AP, else output an error *err* that shall indicate that decapsulation has failed.

4.2 Specifications of CPA-secure ciphertext-policy and key-policy ABKEMs

4.2.1 General

4.2.1.1 Introduction

The ABKEM security notion of indistinguishability under chosen-plaintext attacks (CPA) is the most basic security notion for ABKEMs. It captures that no efficient adversary can distinguish symmetric keys even in case that the encapsulating ciphertext is present. This also holds when the adversary has access to secret keys of its choice that are not authorized to decrypt the challenge ciphertexts.

Four CPA-secure CP-ABKEMs and KP-ABKEMs dubbed CP-WATERS-KEM, CP-FAME-KEM, KP-FAME-KEM, and KP-GSPW-KEM are specified in clauses 4.2.2, 4.2.3, and 4.2.4.

4.2.1.2 Random bit generation

The mechanisms specified hereafter require a random bit generation algorithm. Random bits should be generated as recommended in ISO/IEC 18031:2011 [i.4], including pseudo-random generation.

Given a parameter p, random integers in Z_p are generated by invoking a source of random values in the space $\{0,1\}^{\delta}$, by interpreting the returned bit-string as an integer and by returning its value modulo p. The size parameter δ shall be at least $|p| + 80$.

4.2.1.3 Formats for attributes and policies

Attributes are encoded as strings in a way that non-ambiguously expresses their nature or refers to a determined referential for interpreting the name, type and value of attributes from their string expression. Self-explanatory examples of such strings can be "City:Berlin", "GenderFemale" or "Access.Level3.True".

A monotone attribute policy is an expression defined recursively as either:

- the binary OR of two or more monotone attribute policies, or
- the binary AND of two or more monotone attribute policies, or
- the OUT-OF$[t, n]$ threshold gate of n monotone attribute policies for some $n > 2$ and $2 \leq t \leq n - 1$, or
- an attribute.

NOTE: A monotone attribute policy can be represented by multiple equivalent expressions.

4.2.1.4 The map2point mapping

4.2.1.4.1 General

This clause describes several possible procedures for mapping an integer to a point on an elliptic curve. The $map2point$ primitive shall be used in combination with a regular hash function H to provide a hash function $\mathcal{H}$ that maps arbitrary strings to elliptic curve points:

$$\mathcal{H}(x) = map2point(H(x) \bmod q)\,.$$

Given an integer u in Z_q, $map2point(u)$ evaluates to a point in the group $G_1 = E(GF(q))[p]$. The prime numbers q, p as well as the coefficients $a, b \in Z_q$ in the Weierstrass equation of the curve:

$$E(GF(q)): y^2 = x^3 + ax + b \bmod q$$

shall also be given as input.

The following implementations of $map2point$ may be used depending on the parameters of the chosen elliptic curve:

- $map2point\_34$ is applicable on any curve but q shall be so that $q = 3 \bmod o$ and q shall not be constant-time.
- $map2point\_ssing23$ is applicable on curves with $a = 0$ (supersingular case) and requires that $q = 2 \bmod 3$. The function is constant-time.
- $map2point\_23$ is applicable on any curve with $q = 2 \bmod o$ and is constant-time.

4.2.1.4.2 map2point_34

Input:

- The prime numbers q, p and the coefficients $a, b \in Z_q$ of the curve. The prime q shall be such that $q = 3 \bmod 4$.
- An integer u in Z_q.

Output:

- A point P in $G_1 = E(GF(q))[p]$.

Algorithm:

1) Compute $v = u^3 + au + b \bmod q$.

2) If v is not a quadratic residue modulo q then set $u = u + 1 \bmod q$ and go to Step 1.

3) Compute the square root $w = v^{(p+1)/4} \bmod q$ of v.

4) Return $P = [f](u, w)$ where fp is the number of points in $E(GF(q))$.

NOTE: Testing for quadratic residuosity at Step 2 can be done by computing the Legendre symbol of v with respect to q by any appropriate method. Another method consists in computing $w = v^{(p+1)/4} \bmod q$ directly and testing whether $w^2 = v \bmod q$, which holds only if v is a quadratic residue.

4.2.1.4.3 map2point_ssing23

Input:

- The prime numbers q, p where $q = 2 \bmod 3$ and the coefficient $b \in Z_q$ of the curve.

Output:

- A point P in $G_1 = E(GF(q))[p]$.

Algorithm:

1) Compute $v = (u^2 - b)^{(2q-1)/3} \bmod q$.

2) Return $P = [f](v, u)$ where fp is the number of points in $E(GF(q))$.

4.2.1.4.4 map2point_23

Input:

- The curve parameters q, p, a, b where $q = 2 \bmod 3$.
- An integer u in Z_q.

Output:

- A point P in $G_1 = E(GF(q))[p]$.

Algorithm:

1) Compute

$$v = \frac{3a - u^4}{6u} \bmod q \ ,$$

$$x = \frac{u^2}{3} + \left(v^2 - 2 - \frac{u^6}{27}\right)^{(2q-1)/3} \bmod q \ ,$$

$$y = ux + v \bmod q \ .$$

2) Return $P = [f](x, y)$ where fp is the number of points in $E(GF(q))$.

4.2.1.5 Monotone span programs

4.2.1.5.1 General

Monotone span programs (MSP) are integer-valued matrices that allow one to encode monotone attribute policies algebraically. This clause specifies two algorithms:

- An algorithm MSP_Encode that takes as input a monotone attribute policy P and returns a pair $(M, label)$ where M is an integer-valued matrix and $label$ is a mapping that labels the lines of M with the attributes occurring in P.
- An algorithm MSP_Decode that takes as input a pair $(M, label)$ and a set of attributes S and returns either an invalidity flag or a set I of line indices in M together with a list of coefficients d_I.

The main property of these functions is that, given $(M, label) = \mathsf{MSP\_Encode}(P)$ and a set of attributes S,

- either S does not satisfy P and then $\mathsf{MSP\_Encode}(M, label, S)$ returns the invalidity flag,
- or S satisfies P and then $\mathsf{MSP\_Decode}(M, label, S)$ returns a set I of lines in M and a list of integer-valued coefficients d_I such that

$$\sum_{i \in I} d_i M[i] = (1,0, \ldots ,0) \ .$$

This property is essential to all the ABKEM mechanisms that are specified later on. These mechanisms handle policies exclusively in the form of their MSP encoding. The MSP_Encode procedure can be used to compute the MSP encoding of any monotone attribute policy. MSP_Encode and MSP_Decode shall be as specified hereafter.

4.2.1.5.2 MSP_Encode

Input:

- A monotone attribute policy P.
- An optional argument $vector$ which is a list of integers with default value (1).

Output:

- A pair $(M, label)$ where M is an integer-valued matrix and $label$ is a list of attributes.

Algorithm:

1) If P is of the form $\mathrm{OR}(P_1, \ldots , P_n)$ for some $n \geq 2$ then:

   1.1) For $i = 1$ to n, compute $(M_i, label_i) = MSP\_Encode(P_i, vector)$.

   1.2) Append as many all-zero columns as necessary to $M_1, \ldots , M_n$ so that they have the same number of columns.

```
    1.3) Concatenate $M_1, \dots, M_n$ as $M$.

    1.4) Concatenate $label_1, \dots, label_n$ as $label$.

    1.5) Return $(M, label)$.

2) If $P$ is of the form AND$(P_1, \dots, P_n)$ for some $n \geq 2$ then:

    2.1) Append 1 to $vector\ n - 1$ times to form $vector_1$. Let $m$ be the length of $vector_1$.

    2.2) For $i = 2$ to $n$, set $vector_i$ to the $m$-element list $(0, \dots, 0, -1, 0, \dots, 0)$ where the index of $-1$ is $m - n + i$.

    2.3) For $i = 1$ to $n$, compute $(M_i, label_i) = MSP\_Encode(P_i, vector_i)$.

    2.4) Append as many all-zero columns as necessary to $M_1, \dots, M_n$ so that they have the same number of columns.

    2.5) Concatenate $M_1, \dots, M_n$ as $M$.

    2.6) Concatenate $label_1, \dots, label_n$ as $label$.

    2.7) Return $(M, label)$.

3) If $P$ is of the form OUT-OF$[t, n](P_1, \dots, P_n)$ for some $n > 2$ and $2 \leq t \leq n$ then:

    3.1) For $i = 1$ to $n$:

        3.1.1) Form $vector_i$ by appending the $(t - 1)$-element list $(i, i^2, \dots, i^{t-1})$ to $vector$.

        3.1.2) Compute $(M_i, label_i) = MSP\_Encode(P_i, vector_i)$.

    3.2) Append as many all-zero columns as necessary to $M_1, \dots, M_n$ so that they have the same number of columns.

    3.3) Concatenate $M_1, \dots, M_n$ as $M$.

    3.4) Concatenate $label_1, \dots, label_n$ as $label$.

    3.5) Return $(M, label)$.

4) If $P$ is an attribute $s$ then:

    4.1) Set $M$ to the single-line matrix whose line is $vector$.

    4.2) Set $label$ to the single-element list whose element is $s$.

    4.3) Return $(M, label)$.
```

4.2.1.5.3 MSP_Decode

Input:

- An integer-valued matrix M,
- A list $label$ of attributes,
- A set of attributes S.

Output:

- An invalidity flag, or a pair of sets of integers (I, d_I).

Algorithm:

```
1) Set $n$ to the number of lines of $M$.
```

2) If the length of $label$ is not equal to n, return an error.

3) Set J to the set of all indices $1 \leq j \leq n$ such that $label[j]$ belongs to S.

4) Collect the lines of M whose index belongs to J to form a sub-matrix M_J.

5) Find an integer-valued vector d such that $dM_J = (1,0, \ldots ,0)$.

6) If there is no such vector d then return the invalidity flag.

7) Deduce from d a subset $I \subseteq J$ such that $\sum_{i \in I} d_i M\,[i] = (1,0, \ldots ,0)$ with $d_i \neq 0$.

8) Set d_I to the list of coefficients d_i for $i \in I$.

9) Return (I, d_I).

4.2.2 Specification of CP-WATERS-KEM

4.2.2.1 General

This mechanism realizes a ciphertext-policy attribute-based key encapsulation mechanism (CP-ABKEM) as defined in clause 4.1.2.

4.2.2.2 Setup

The setup operation creates public system parameters, a master public key and a master secret key.

Input:

- The security parameter κ.

Output:

- The system parameters $params$,
- The master public key mpk,
- The master secret key msk.

Algorithm:

The steps to setup the master public and secret keys and system parameters shall be as follows:

1) Given the security parameter κ, establish the set of base groups G_1, G_2, G_T and a pairing $e: G_1 \times G_2 \rightarrow G_T$. The base groups shall be established together with a generator g_1 of G_1 and a generator g_2 of G_2.

2) Define a hash function $\mathcal{H}$ that hashes strings into elements of G_1 (see clause 4.2.1.4).

3) Set the system parameters to

$$params = (\kappa, p, G_1, G_2, G_T, e, g_1, g_2, \mathcal{H}) \ .$$

4) Generate uniformly random variables a, b in Z_p.

5) Set the master public key to

$$mpk = (mpk_1, mpk_2) = (g_1^b, e(g_1, g_2)^a) \ .$$

6) Set the master secret key to

$$msk = g_1^a \ .$$

7) Make the system parameters and the master public key available. Secure the master secret key.

4.2.2.3 Secret-key generation

The secret-key generation operation takes an arbitrary set of attributes S and generates a corresponding secret decryption key sk_S. The algorithm shall be as follows:

Input:

- The system parameters $params$,
- The master public key mpk,
- The master secret key msk,
- A set of attributes $S = (s_1, s_2, \dots, s_t)$.

Output:

- A secret decryption key sk_S.

Algorithm:

1) Parse $params$, mpk and msk as above.
2) Select $r \in Z_p$ uniformly at random.
3) Compute:

$$x_1 = msk \cdot (mpk_1)^r = g_1^{a+br} \ , \quad x_2 = g_2^r \ .$$

4) For i in $\{1, \dots, t\}$, compute

$$sk_i = \mathcal{H}(s_i)^r \ .$$

5) Return the secret key $sk_S = (x_1, x_2, sk_1, \dots, sk_t)$.

4.2.2.4 Symmetric-key encapsulation

The key encapsulation operation takes the MSP encoding $(M_P, label_P)$ of a monotone attribute policy P and generates a random symmetric key K as well as its encapsulation C_P. The algorithm shall be as follows.

Input:

- The system parameters $params$,
- The master public key mpk,
- An MSP encoding $(M_P, label_P)$,
- A random integer R serving as a seed for pseudo-random number generation.

Output:

- A pair (K, C_P) where K is a random symmetric key and C_P is its encapsulation.

Algorithm:

1) Parse $params$ and mpk as above.
2) Let n be the number of lines of M_P and m the number of its columns. Pseudo-randomly generate $v_1, \dots, v_m$ in Z_p and compute:

$$(\mu_1, \dots, \mu_n) = M_P \cdot (v_1, \dots, v_m) \bmod p \ .$$

3) Compute:

$$z = g_2^{v_1}, \quad K = (mpk_2)^{v_1} = e(g_1, g_2)^{a v_1} \ .$$

4) For i in $\{1, \dots, n\}$,

   1) Pseudo-randomly generate r_i in Z_p.

   2) Compute $c_i = (c_{i,1}, c_{i,2})$ where

$$c_{i,1} = (mpk_1)^{\mu_i} \mathcal{H}(label_P[i])^{-r_i} \ , \qquad c_{i,2} = g_2^{r_i} \ ,$$

5) Set $C_P = (z, c_1, \dots, c_n)$.

6) Return (K, C_P).

4.2.2.5 Symmetric-key decapsulation

The key decapsulation operation takes an encapsulation C_P and a secret decryption key sk_S and returns either the symmetric key K if S satisfies P or an invalidity flag. The algorithm shall be as follows:

Input:

- The system parameters $params$,
- An MSP encoding $(M_P, label_P)$,
- An encapsulation C_P produced for $(M_P, label_P)$,
- A set of attributes S,
- A secret decryption key sk_S produced for S.

Output:

- The decapsulated symmetric key K if S satisfies P or an invalidity flag otherwise.

Algorithm:

1) Parse $params$, $(M_P, label_P)$, C_P, S and sk_S as above.

2) Compute $\mathsf{MSP\_Decode}(M_P, label_P, S)$ using the algorithm described in clause 4.2.1.5.3.

3) If MSP_Decode returned the invalidity flag then stop and return that flag.

4) Otherwise MSP_Decode returned a set of indices I and a list of nonzero integers d_I; then compute:

$$w = \prod_{i \in I} c_{i,1}^{d_i} \ , \quad K = \frac{e(x_1, z)}{e(w, x_2) \cdot \prod_{i \in I} e\left(sk_{pos(i)}, c_{i,2}^{d_i}\right)} \ .$$

where $pos(i)$ is the index of $label_P[i]$ in S.

5) Return K.

4.2.3 Specification of CP-FAME-KEM and KP-FAME-KEM

4.2.3.1 Hash functions

The hash functions $\mathcal{H}_{\ell,k}$ and $\mathcal{G}_{\ell,k}$ for $\ell = \{1,2,3\}$ and $k = \{1,2\}$ are used for hashing arbitrary bit-strings to a point on the elliptic curve group G_1. They are based on a generic hash function H and on a subroutine $map2point$ described in the clause 4.2.1.4. The description of G_1 includes a prime q and H shall be chosen such that it returns bit-strings of length at least $|q| + 80$.

Given an arbitrary bit-string m, $\mathcal{H}_{\ell,k}(m)$ shall be computed as follows:

1) Set $padding = \ell + 3k - 4$ and interpret $padding$ as an 8-bit string. Namely:

   - if $(\ell, k) = (1,1)$ then $padding = 0x00$,
   - if $(\ell, k) = (2,1)$ then $padding = 0x01$,
   - if $(\ell, k) = (3,1)$ then $padding = 0x02$,
   - if $(\ell, k) = (1,2)$ then $padding = 0x03$,
   - if $(\ell, k) = (2,2)$ then $padding = 0x04$,
   - if $(\ell, k) = (3,2)$ then $padding = 0x05$.

2) Return $\mathcal{H}_{\ell,k}(m) = map2point(H(padding \,||\, m) \bmod q)$.

Given an arbitrary bit-string m, $\mathcal{G}_{\ell,k}(m)$ shall be computed as follows:

1) Set $padding = \ell + 3k + 2$ and interpret $padding$ as an 8-bit string. Namely:

   - if $(\ell, k) = (1,1)$ then $padding = 0x06$,
   - if $(\ell, k) = (2,1)$ then $padding = 0x07$,
   - if $(\ell, k) = (3,1)$ then $padding = 0x08$,
   - if $(\ell, k) = (1,2)$ then $padding = 0x09$,
   - if $(\ell, k) = (2,2)$ then $padding = 0x0A$,
   - if $(\ell, k) = (3,2)$ then $padding = 0x0B$.

2) Return $\mathcal{G}_{\ell,k}(m) = map2point(H(padding \,||\, m) \bmod q)$.

The mechanisms for H shall be the SHA-2 family of functions and SHA-3, possibly composed with truncation as appropriate.

4.2.3.2 Setup for CP-FAME-KEM and KP-FAME-KEM

The setup operation creates public system parameters, a master public key and a master secret key. The master secret key shall be secured by the secret key issuer, as all secret decryption keys calculated within the system depend on it. Applications should establish a methodology for renewing the master secret key, the corresponding master public key and system parameters on a regular basis, and have a methodology for handling the disclosure of a master secret key.

Note that this setup operation is the same for both CP-FAME-KEM and KP-FAME-KEM.

Input:

- The security parameter κ.

Output:

- The system parameters $params$,
- The master public key mpk,
- The master secret key msk.

Algorithm:

The steps to setup the master public and secret keys and system parameters shall be as follows:

1) Given the security parameter κ, establish the set of base groups G_1, G_2, G_T and a pairing $e: G_1 \times G_2 \rightarrow G_T$. G_1 shall be of the form $E(GF(q))[p]$, where p and q are primes [i.28] contains recommendations for the generation of the groups, group generators and pairings. The base groups shall be established together with a generator g_1 of G_1 and a generator g_2 of G_2.

2) For ℓ in $\{1,2,3\}$ and k in $\{1,2\}$, compute $g_{\ell,k} = \mathcal{G}_{\ell,k}(1)$.

3) Set the system parameters to

$$params = \left(\kappa, p, G_1, G_2, G_T, e, g_1, g_2, g_{1,1}, g_{2,1}, g_{3,1}, g_{1,2}, g_{2,2}, g_{3,2}\right) .$$

4) Generate random variables $r, a_1, a_2, b_1, b_2, d_1, d_2, d_3$ in Z_p.

5) Calculate

$$g = g_1^r, \quad H_1 = g_2^{a_1}, \quad H_2 = g_2^{a_2}, \quad T_1 = e(g, g_2)^{d_1 a_1 + d_3}, \quad T_2 = e(g, g_2)^{d_2 a_2 + d_3} .$$

6) Set the master public key to:

$$mpk = (H_1, H_2, T_1, T_2) .$$

7) Set the master secret key to

$$msk = (g, a_1, a_2, b_1, b_2, d_1, d_2, d_3) .$$

8) Make the system parameters and the master public key available. Secure the master secret key.

4.2.3.3 CP-FAME-KEM

4.2.3.3.1 General

This mechanism realizes a ciphertext-policy attribute-based key encapsulation mechanism as defined in clause 4.1. Contrarily to CP-WATERS-KEM, this mechanism requires only 6 evaluations of a bilinear map during key decapsulation, which makes it particularly attractive when decapsulation is time-critical.

4.2.3.3.2 Secret-key generation

The secret-key extraction operation takes an arbitrary set of attributes S and generates a corresponding secret decryption key sk_S. The algorithm shall be as follows:

Input:

- The system parameters $params$.
- The master public key mpk.
- The master secret key msk.
- A set of attributes $S = (s_1, s_2, \dots, s_t)$.

Output:

- A secret decryption key sk_S.

Algorithm:

1) Parse $params$, mpk and msk as above.

2) Generate random variables r_1, r_2, σ in Z_p.

3) Compute

$$x_1 = g_2^{b_1 r_1}, \quad x_2 = g_2^{b_2 r_2}, \quad x_3 = g_2^{r_1 + r_2},$$

$$y_1 = g_{1,1}^{\frac{b_1 r_1}{a_1}} g_{2,1}^{\frac{b_2 r_2}{a_1}} g_{3,1}^{\frac{r_1 + r_2}{a_1}} g^{\frac{\sigma}{a_1} + d_1}, \quad y_2 = g_{1,2}^{\frac{b_1 r_1}{a_2}} g_{2,2}^{\frac{b_2 r_2}{a_2}} g_{3,2}^{\frac{r_1 + r_2}{a_2}} g^{\frac{\sigma}{a_2} + d_2}, \quad y_3 = g^{d_3 - \sigma}.$$

4) For i in $\{1, \dots, t\}$,

a) Generate a random variable σ_i in Z_p,

b) Compute:

$$sk_{i,1} = \mathcal{H}_{1,1}(s_i)^{\frac{b_1 r_1}{a_1}} \mathcal{H}_{2,1}(s_i)^{\frac{b_2 r_2}{a_1}} \mathcal{H}_{3,1}(s_i)^{\frac{r_1 + r_2}{a_1}} g^{\frac{\sigma_i}{a_1}},$$

$$sk_{i,2} = \mathcal{H}_{1,2}(s_i)^{\frac{b_1 r_1}{a_2}} \mathcal{H}_{2,2}(s_i)^{\frac{b_2 r_2}{a_2}} \mathcal{H}_{3,2}(s_i)^{\frac{r_1 + r_2}{a_2}} g^{\frac{\sigma_i}{a_2}}.$$

$$sk_{i,3} = g^{-\sigma_i}.$$

c) Set $sk_i = (sk_{i,1}, sk_{i,2}, sk_{i,3})$.

5) Return the secret key $sk_S = (x_1, x_2, x_3, y_1, y_2, y_3, sk_1, \dots, sk_t)$.

4.2.3.3.3 Symmetric-key encapsulation

The symmetric-key encapsulation operation takes as input an MSP encoding $(M_P, label_P)$ and generates a random symmetric key K as well as its encapsulation C_P. The algorithm shall be as follows.

Input:

- The system parameters $params$,
- The master public key mpk,
- An MSP encoding $(M_P, label_P)$,
- A random integer R serving as a seed for pseudo-random number generation.

Output:

- A pair (K, C_P) where K is a random symmetric key and C_P is its encapsulation.

Algorithm:

1) Parse $params$ and mpk as above.

2) Pseudo-randomly generate variables u_1, u_2 in Z_p.

3) Compute:

$$z_1 = H_1^{u_1}, \quad z_2 = H_2^{u_2}, \quad z_3 = g_2^{u_1 + u_2}, \quad K = T_1^{u_1} T_2^{u_2}.$$

4) Let n be the number of lines of M_P and m the number of its columns.

5) For i in $\{1, \dots, n\}$,

a) For ℓ in $\{1,2,3\}$, compute:

$$c_{i,\ell} = \left(\mathcal{H}_{\ell,1}(label[i])\right)^{u_1} \left(\mathcal{H}_{\ell,2}(label[i])\right)^{u_2} \prod_{j=1}^{m} \left(\mathcal{G}_{\ell,1}(j)^{u_1} \mathcal{G}_{\ell,2}(j)^{u_2}\right)^{M[i,j]}.$$

b) Set $c_i = (c_{i,1}, c_{i,2}, c_{i,3})$.

6) Set $C_P = (z_1, z_2, z_3, c_1, \dots, c_n)$.

7) Return (K, C_P).

4.2.3.3.4 Symmetric-key decapsulation

The symmetric-key decapsulation operation takes an encapsulation C_P and a secret key sk_S and returns either the symmetric key K if S satisfies P or an invalidity flag. The algorithm shall be as follows:

Input:

- The system parameters $params$,
- An MSP encoding $(M_P, label_P)$,
- An encapsulation C_P produced for $(M_P, label_P)$,
- A set of attributes S,
- A secret decryption key sk_S produced for S.

Output:

- The decapsulated symmetric key K if S satisfies P or an invalidity flag otherwise.

Algorithm:

1) Parse $params$, sk_S and C_P as above.
2) Compute $\mathsf{MSP\_Decode}(M_P, label_P, S)$ using the algorithm described in clause 4.2.1.5.3.
3) If MSP_Decode returned the invalidity flag then stop and return that flag.
4) Otherwise MSP_Decode returned a set of indices I and a list of nonzero integers d_I; then compute:

$$t_1 = y_1 \prod_{i \in I} sk_{label[i],1}{}^{d_i}\ , \quad t_2 = y_2 \prod_{i \in I} sk_{label[i],2}{}^{d_i}\ , \quad t_3 = y_3 \prod_{i \in I} sk_{label[i],3}{}^{d_i}\ ,$$

$$v_1 = \prod_{i \in I} c_{i,1}{}^{d_i}\ , \quad v_2 = \prod_{i \in I} c_{i,2}{}^{d_i}\ , \quad v_3 = \prod_{i \in I} c_{i,3}{}^{d_i}\ ,$$

$$K = \frac{e(t_1, z_1) \cdot e(t_2, z_2) \cdot e(t_3, z_3)}{e(v_1, x_1) \cdot e(v_2, x_2) \cdot e(v_3, x_3)}\ .$$

5) Return K.

NOTE: The decapsulation operation requires the computation of only 6 pairings independently of the complexity of the policy P and set of attributes S.

4.2.3.4 KP-FAME-KEM

4.2.3.4.1 General

This mechanism realizes a key-policy attribute-based key encapsulation mechanism as defined in clause 4.1.

4.2.3.4.2 Secret-key generation

The secret-key generation operation takes as input an MSP encoding $(M_P, label_P)$ and generates a corresponding secret key sk_P. The algorithm shall be as follows.

Input:

- The system parameters $params$,
- The master public key mpk,
- The master secret key msk,
- An MSP encoding $(M_P, label_P)$.

Output:

- A secret decryption key sk_P.

Algorithm:

1) Parse $params$, mpk and msk as above.

2) Generate random variables r_1, r_2 in Z_p.

3) Compute:

$$x_1 = g_2^{b_1 r_1}, \quad x_2 = g_2^{b_2 r_2}, \quad x_3 = g_2^{r_1 + r_2} \ .$$

4) Let n be the number of lines M_P and m the number of its columns.

5) Generate random variables $\rho_2, \ldots, \rho_m$ in Z_p.

6) For i in $\{1, \ldots, n\}$,

   a) Generate a random variable σ_i in Z_p,

   b) Compute:

$$A_{i,1} = \mathcal{H}_{1,1}(label[i])^{\frac{b_1 r_1}{a_1}} \mathcal{H}_{2,1}(label[i])^{\frac{b_2 r_2}{a_1}} \mathcal{H}_{3,1}(label[i])^{\frac{r_1 + r_2}{a_1}} g^{\frac{\sigma_i}{a_1} + d_1 M[i,1]} \ ,$$

$$B_{i,1} = \prod_{j=2}^{m} \left(\mathcal{G}_{1,1}(j)^{\frac{b_1 r_1}{a_1}} \mathcal{G}_{2,1}(j)^{\frac{b_2 r_2}{a_1}} \mathcal{G}_{3,1}(j)^{\frac{r_1 + r_2}{a_1}} g^{\frac{\rho_j}{a_1}} \right)^{M[i,j]} \ ,$$

$$sk_{i,1} = A_{i,1} B_{i,1} \ ,$$

$$A_{i,2} = \mathcal{H}_{1,2}(label[i])^{\frac{b_1 r_1}{a_2}} \mathcal{H}_{2,2}(label[i])^{\frac{b_2 r_2}{a_2}} \mathcal{H}_{3,2}(label[i])^{\frac{r_1 + r_2}{a_2}} g^{\frac{\sigma_i}{a_2} + d_2 M[i,1]} \ ,$$

$$B_{i,2} = \prod_{j=2}^{m} \left(\mathcal{G}_{1,2}(j)^{\frac{b_1 r_1}{a_2}} \mathcal{G}_{2,2}(j)^{\frac{b_2 r_2}{a_2}} \mathcal{G}_{3,2}(j)^{\frac{r_1 + r_2}{a_2}} g^{\frac{\rho_j}{a_2}} \right)^{M[i,j]} \ ,$$

$$sk_{i,2} = A_{i,2} B_{i,2} \ ,$$

$$sk_{i,3} = g^{-\sigma_i + d_3 M[i,1] - \sum_{j=2}^{m} \rho_j M[i,j]} \ .$$

   c) Set $sk_i = (sk_{i,1}, sk_{i,2}, sk_{i,3})$.

7) Return the secret decryption key $sk_P = (x_1, x_2, x_3, sk_1, \ldots, sk_n)$.

4.2.3.4.3 Symmetric-key encapsulation

The symmetric-key encapsulation operation takes set of attributes S and generates a random symmetric key K as well as its encapsulation C_S. The algorithm shall be as follows:

Input:

- The system parameters $params$,
- The master public key mpk,
- A set of attributes $S = (s_1, s_2, \dots, s_t)$,
- A random integer R serving as a seed for pseudo-random number generation.

Output:

- A pair (K, C_S) where K is a random symmetric key and C_S is its encapsulation.

Algorithm:

1) Parse $params$ and mpk as above.

2) Pseudo-randomly generate variables u_1, u_2 in Z_p.

3) Compute:

$$z_1 = H_1^{u_1}, \quad z_2 = H_2^{u_2}, \quad z_3 = g_2^{u_1+u_2}, \quad K = T_1^{u_1} T_2^{u_2} \ .$$

4) For i in $\{1, \dots, t\}$,

   a) Compute:

$$c_{i,1} = \mathcal{H}_{1,1}(s_i)^{u_1} \mathcal{H}_{1,2}(s_i)^{u_2} \ .$$

$$c_{i,2} = \mathcal{H}_{2,1}(s_i)^{u_1} \mathcal{H}_{2,2}(s_i)^{u_2} \ .$$

$$c_{i,3} = \mathcal{H}_{3,1}(s_i)^{u_1} \mathcal{H}_{3,2}(s_i)^{u_2} \ .$$

   b) Set $c_i = (c_{i,1}, c_{i,2}, c_{i,3})$.

5) Set $C_S = (z_1, z_2, z_3, c_1, \dots, c_t)$.

6) Return (K, C_S).

4.2.3.4.4 Symmetric-key decapsulation

The symmetric-key decapsulation operation takes an encapsulation C_S and a secret key sk_P and returns either the symmetric key K if S satisfies P or an invalidity flag. The algorithm shall be as follows:

Input:

- The system parameters $params$,
- A set of attributes S,
- An encapsulation C_S produced for S,
- An MSP encoding $(M_P, label_P)$,
- A secret decryption key sk_P produced for $(M_P, label_P)$.

Output:

- The decapsulated symmetric key K if S satisfies P or an invalidity flag otherwise.

Algorithm:

1) Parse $params$, sk_P and C_S as above.

2) Compute $\mathsf{MSP\_Decode}(M_P, label_P, S)$ using the algorithm described in clause 4.2.1.5.3.

3) If MSP_Decode returned the invalidity flag then stop and return that flag.

4) Otherwise MSP_Decode returned a set of indices I and a list of nonzero integers d_I; then compute:

$$t_1 = \prod_{i \in I} {sk_{i,1}}^{d_i} \ , \quad t_2 = \prod_{i \in I} {sk_{i,2}}^{d_i} \ , \quad t_3 = \prod_{i \in I} {sk_{i,3}}^{d_i} \ ,$$

$$v_1 = \prod_{i \in I} {c_{label[i],1}}^{d_i} \ , \quad v_2 = \prod_{i \in I} {c_{label[i],2}}^{d_i} \ , \quad v_3 = \prod_{i \in I} {c_{label[i],3}}^{d_i} \ ,$$

$$K = \frac{e(t_1, z_1) \cdot e(t_2, z_2) \cdot e(t_3, z_3)}{e(v_1, x_1) \cdot e(v_2, x_2) \cdot e(v_3, x_3)} \ .$$

5) Return K.

NOTE: Here again, the decapsulation operation requires the computation of only 6 pairings independently of the complexity of the policy P and set of attributes S.

4.2.4 Specification of KP-GSPW-KEM

4.2.4.1 General

This mechanism realizes a key-policy attribute-based encapsulation mechanism as defined in clause 4.1.3. A specific property of this mechanism is that all the possible attributes under which encapsulations may be produced shall be known at setup time.

4.2.4.2 Setup

The setup operation creates public system parameters, a master public key and a master secret key.

Input:

- The security parameter κ.

Output:

- The system parameters $params$,
- The master public key mpk,
- The master secret key msk.

Algorithm:

The steps to setup the master public and secret keys and system parameters shall be as follows:

1) Given the security parameter κ, establish the set of base groups G_1, G_2, G_T and a pairing $e: G_1 \times G_2 \to G_T$. The base groups shall be established together with a generator g_1 of G_1 and a generator g_2 of G_2.

2) Define a hash function H that hashes strings into integers modulo p.

3) Set the system parameters to:

$$params = (\kappa, p, G_1, G_2, G_T, e, g_1, g_2, H) \ .$$

4) Generate uniformly random variables a, b in Z_p and compute:

$$x = g_2^b \ , \quad y = e(g_1, g_2)^a \ .$$

5) For all possible attributes $\overline{s}_1, \dots, \overline{s}_u$, compute:

$$t_i = H(a, \overline{s}_i) \ , \quad T_i = g_1^{t_i} \ .$$

6) Set the master public key to $mpk = (y, T_1, \dots, T_u)$.

7) Set the master secret key to $msk = (x, a)$.

8) Make the system parameters and the master public key available. Secure the master secret key.

4.2.4.3 Secret-key generation

The generation of a secret decrytion key takes as input an MSP encoding $(M_P, label_P)$ and generates a corresponding secret key sk_P. The algorithm shall be as follows.

Input:

- The system parameters $params$,
- The master public key mpk,
- The master secret key msk,
- An MSP encoding $(M_P, label_P)$.

Output:

- A secret decryption key sk_P.

Algorithm:

1) Parse $params$, mpk and msk as above.

2) Let n be the number of lines M_P and m the number of its columns.

3) Generate $v_2, \dots, v_m$ uniformly at random in Z_p and compute:

$$(\mu_1, \dots, \mu_n) = M_P \cdot (a, v_2 \dots, v_m) \bmod p \ .$$

4) For $i = 1$ to n, compute:

$$t_i = H(a, label_P[i]) \ , \quad \sigma_i = \frac{\mu_i}{t_i} \bmod p \ , \quad sk_i = x^{\sigma_i} \ .$$

5) Return the secret key $sk_P = (sk_1, \dots, sk_n)$.

4.2.4.4 Symmetric-key encapsulation

The key encapsulation operation takes set of attributes S and generates a random symmetric key K as well as its encapsulation C_S. The algorithm shall be as follows:

Input:

- The system parameters $params$,
- The master public key mpk,
- A set of attributes $S = (s_1, s_2, \dots, s_t)$,
- A random integer R serving as a seed for pseudo-random number generation.

Output:

- A pair (K, C_S) where K is a random symmetric key and C_S is its encapsulation.

Algorithm:

1) Parse $params$ and mpk as above.
2) Pseudo-randomly generate u in Z_p and compute:

$$K = y^u \ .$$

3) For i in $\{1, \ldots, t\}$, find $j \in [1, u]$ such that $s_i = \overline{s}_j$ and compute $c_i = T_j^u$.
4) Set $C_S = (c_1, \ldots, c_t)$.
5) Return (K, C_S).

4.2.4.5 Symmetric-key decapsulation

The key decapsulation operation takes an encapsulation C_S and a secret key sk_P and returns either the symmetric key K if S satisfies P or an invalidity flag. The algorithm shall be as follows.

Input:

- The system parameters $params$,
- A set of attributes S,
- An encapsulation C_S produced for S,
- An MSP encoding $(M_P, label_P)$,
- A secret decryption key sk_P produced for $(M_P, label_P)$.

Output:

- The decapsulated symmetric key K if S satisfies P or an invalidity flag otherwise.

Algorithm:

1) Parse $params$, sk_P and C_S as above.
2) Compute MSP_Decode$(M_P, label_P, S)$ using the algorithm described in clause 4.2.1.5.3.
3) If MSP_Decode returned the invalidity flag then stop and return that flag.
4) Otherwise MSP_Decode returned a set of indices I and a list of nonzero integers d_I; then compute:

$$K = \prod_{i \in I} e\left(c_{pos(i)}, sk_i^{d_i}\right)$$

5) Where $pos_1(i)$ is the index of $label_P[i]$ in S.
6) Return K.

4.3 Ciphertext-policy and key-policy attribute-based encryption

4.3.1 Overview

Ciphertext-policy and key-policy attribute-based encryption (CP-ABE and KP-ABE) schemes for messages of arbitrary length achieving CPA-security are specified in clauses 4.3.2 and 4.3.3.

4.3.2 Ciphertext-policy ABE

The ciphertext-policy attribute-based encryption (CP-ABE) scheme shall associate secret keys with attributes from an attribute universe and ciphertexts with access policies. The messages to be encrypted can be of arbitrary integer length.

The following flow is specified:

- **System setup:**
  - The setup party shall specify a security parameter k and shall output system parameters $PARAMS_{CP\text{-}ABE}$, a master public key $MPK_{CP\text{-}ABE}$, and a master secret key $MSK_{CP\text{-}ABE}$. ($MSK_{CP\text{-}ABE}$ shall contain $MPK_{CP\text{-}ABE}$, $MPK_{CP\text{-}ABE}$ shall contain $PARAMS_{CP\text{-}ABE}$.)
- **Secret-key generation:**
  - The ABE authority shall take as input the master secret key $MSK_{CP\text{-}ABE}$, and an attribute set A from the attribute universe, and shall output a secret key SK_A. (SK_A shall contain $MPK_{CP\text{-}ABE}$.)
- **Message encryption:**
  - The encryptor shall take as input the master public key $MPK_{CP\text{-}ABE}$, a message M, an access policy AP, and a random tape R, and shall output a ciphertext C_{AP}. (AP shall be extractable from C_{AP}. R shall be a uniformly random bitstring from $\{0,1\}^k$.)
- **Ciphertext decryption:**
  - The decryptor shall take as input a secret key SK_A associated with an attribute set A from the attribute universe, and a ciphertext C_{AP} associated with an access policy AP, and shall output a message M if A satisfy AP, else output an error err that shall indicate that decryption has failed.

4.3.3 Key-policy ABE

The key-policy attribute-based encryption (KP-ABE) scheme shall associate ciphertexts with attributes from an attribute universe and secret keys with access policies. The messages to be encrypted can be of arbitrary integer length.

The following flow is specified:

- **Setup procedure:**
  - The setup party shall specify a security parameter k and shall output system parameters $PARAMS_{KP\text{-}ABE}$, a master public key $MPK_{KP\text{-}ABE}$, and a master secret key $MSK_{KP\text{-}ABE}$. ($MSK_{KP\text{-}ABE}$ shall contain $MPK_{KP\text{-}ABE}$, $MPK_{KP\text{-}ABE}$ shall contain $PARAMS_{KP\text{-}ABE}$.)
- **Secret-key generation:**
  - The ABE authority shall take as input the master secret key $MSK_{KP\text{-}ABE}$, and an access policy AP, and shall output a secret key SK_{AP}. (SK_{AP} shall contain $MPK_{KP\text{-}ABE}$.)
- **Message encryption:**
  - The encryptor shall take as input the master public key $MPK_{KP\text{-}ABE}$, a message M, an attribute set A from the attribute universe, and a random tape R, and shall output a ciphertext C_A. (A shall be extractable from C_A. R shall be a uniformly random bitstring from $\{0,1\}^k$.)
- **Ciphertext decryption:**
  - The decryptor shall take as input a secret key SK_{AP} associated with an access policy AP, and a ciphertext C_A associated with an attribute set A from the attribute universe, and shall output a message M if A satisfy AP, else output an error err that shall indicate that decryption has failed.

4.4 Specifications of CPA-secure ciphertext-policy and key-policy ABE

4.4.1 General

4.4.1.1 Introduction

The ABE security notion of indistinguishability under chosen-plaintext attacks (CPA) is the most basic security notion for ABE schemes. It captures that no efficient adversary can distinguish challenge ciphertexts under self-chosen messages. This even holds when the adversary has access to secret keys of its choice that are not authorized to decrypt the challenge ciphertexts.

Specifications of CPA-secure CP-ABE and KP-ABE schemes from CP-ABKEMs and KP-ABKEMs are given in clauses 4.4.2 and 4.4.3.

4.4.1.2 Pseudorandom generator

Let k be an integer. The pseudorandom-generator (PRG) operation P shall take as input a seed bitstring s in $\{0,1\}^m$ and the length integer parameter $l>m$, and shall return a pseudorandom bitstring r in $\{0,1\}^l$ or an invalidity flag otherwise.

4.4.2 CPA-secure CP-ABE

A CPA-secure CP-ABE scheme may be built from any CPA-secure CP-ABKEM and a pseudorandom generator. The attribute universe of the CP-ABE scheme shall be the attribute universe of the CP-ABKEM. The symmetric-keys shall be bitstrings of integer length m from $\{0,1\}^m$. The messages shall be bitstrings of integer length l from $\{0,1\}^l$. Let XOR be the bitwise-exclusive-or operator and let R be a uniformly random bitstring from $\{0,1\}^k$ for security parameter k.

The following flow is specified:

- **System setup:**
  - The setup party shall specify a security parameter k and a length parameter l, and shall run the system setup of the CP-ABKEM, and shall return the CP-ABKEM master public key $MPK_{CP\text{-}ABKEM}$ and master secret key $MSK_{CP\text{-}ABKEM}$. (Note that $PARAMS_{CP\text{-}ABKEM}$ is contained in $MPK_{CP\text{-}ABKEM}$.)
- **Secret-key generation:**
  - The ABE authority shall take as input the master secret key $MSK_{CP\text{-}ABKEM}$ and an attribute set A from the attribute universe, and shall output a secret key SK_A obtained from CP-ABKEM secret-key generation with $MSK_{CP\text{-}ABKEM}$ and A.
- **Message encryption:**
  - The encryptor shall take as input the master public key $MPK_{CP\text{-}ABKEM}$, a message M, an access policy AP, and random tape R, and shall run symmetric-key encapsulation of the CP-ABKEM with $MPK_{CP\text{-}ABKEM}$, AP, and R, to obtain a symmetric key $K_{CP\text{-}ABKEM}$ and C'_{AP}.
  - The encryptor shall run the pseudorandom generator with $K_{CP\text{-}ABKEM}$ and length parameter l to obtain a pseudorandom bitstring r in $\{0,1\}^l$.
  - The encryptor shall compute $C_D=r$ XOR M and shall output the ABE ciphertext $C_{AP}=(C'_{AP}, C_D, l)$.
- **Ciphertext decryption:**
  - The decryptor shall take as input the secret key SK_A and a ciphertext $C_{AP}=(C'_{AP}, C_D, l)$, and shall run the symmetric-key decapsulation of the CP-ABKEM scheme with SK_A and C'_{AP} to obtain $K_{CP\text{-}ABKEM}$.
  - If $K_{CP\text{-}ABKEM}$ is not equal to the error err, the decryptor shall run the pseudorandom generator with $K_{CP\text{-}ABKEM}$ and length parameter l to obtain a pseudorandom bitstring r from $\{0,1\}^l$, otherwise output error err.
  - The decryptor shall compute $M=r$ XOR C_D and shall output the message M.

4.4.3 CPA-secure KP-ABE scheme

A CPA-secure KP-ABE scheme may be built from any CPA-secure KP-ABKEM and a pseudorandom generator. The attribute universe of the KP-ABE scheme shall be the attribute universe of the KP-ABKEM. The symmetric-keys shall be bitstrings of integer length m from $\{0,1\}^m$. The messages shall be bitstrings of integer length l from $\{0,1\}^l$. Let XOR be the bitwise-exclusive-or operator and let R be a uniformly random bitstring from $\{0,1\}^k$ for security parameter k.

The following flow is specified:

- **System setup:**
  - The setup party shall specify a security parameter k and shall run the system setup of the KP-ABKEM, and shall return the KP-ABKEM master public key $MPK_{KP\text{-}ABKEM}$ and master secret key $MSK_{KP\text{-}ABKEM}$. (Note that $PARAMS_{KP\text{-}ABKEM}$ is contained in $MPK_{KP\text{-}ABKEM}$.)
- **Secret-key generation:**
  - The ABE authority shall take as input the master secret key $MSK_{KP\text{-}ABKEM}$ and an access policy AP, and shall output a secret key SK_{AP} obtained from KP-ABKEM secret-key generation with $MSK_{KP\text{-}ABKEM}$ and AP.
- **Message encryption:**
  - The encryptor shall take as input the master public key $MPK_{KP\text{-}ABKEM}$, a message M, an attribute set A from the attribute universe, and random tape R, and shall run the symmetric-key encapsulation of the KP-ABKEM scheme with $MPK_{KP\text{-}ABKEM}$, A, and R, to obtain a symmetric key $K_{KP\text{-}ABKEM}$ and C'_A.
  - The encryptor shall run the pseudorandom generator with $K_{KP\text{-}ABKEM}$ and length parameter l to obtain a pseudorandom bitstring r in $\{0,1\}^l$.
  - The encryptor shall compute $C_D=r$ XOR M and shall output the ABE ciphertext $C_A=(C'_A, C_D, l)$.
- **Ciphertext decryption:**
  - The decryptor shall take as input the secret key SK_{AP} and a ciphertext $C_A=(C'_A, C_D, l)$, and shall run symmetric-key decapsulation of the KP-ABKEM scheme with SK_{AP} and C_A to obtain $K_{KP\text{-}ABKEM}$.
  - If $K_{KP\text{-}ABKEM}$ is not equal to the error err, the decryptor shall run the pseudorandom generator with $K_{KP\text{-}ABKEM}$ and length parameter l to obtain a pseudorandom bitstring r in $\{0,1\}^l$, otherwise output error err.
  - The decryptor shall compute $M=r$ XOR C_D and shall output the message M.

4.5 Specifications of CCA-secure CP-ABKEMs and KP-ABKEMs, CP-ABE schemes, and KP-ABE schemes

4.5.1 General

4.5.1.1 Introduction

The ABKEM and ABE security notions of indistinguishability under chosen-ciphertext attacks (CCA) are the standard security notions for ABKEMs and ABE schemes. It captures that no efficient adversary can distinguish symmetric keys even in case that the encapsulating ciphertext is present (ABKEM) or no efficient adversary can distinguish challenge encryptions of self-chosen messages (ABE). This also holds when the adversary has access to secret keys of its choice that are not authorized to decapsulate the encapsulating ciphertext (ABKEM) or decrypt the challenge ciphertexts (ABE).

Specifications of CCA-secure CP-ABKEMs and KP-ABKEMs from CPA-secure CP-ABE and KP-ABE schemes are given in clauses 4.5.2 and 4.5.3.

Specifications of CCA-secure CP-ABE and KP-ABE schemes from CCA-secure CP-ABKEMs and KP-ABKEMs are given in clauses 4.5.4 and 4.5.5.

The concatenation of bitstrings is denoted by the symbol ||.

4.5.1.2 Collusion-resistant hash function

A collusion-resistant hash function H shall take as input a bitstring of arbitrary integer length from $\{0,1\}^*$ and shall output a uniformly random bitstring of size $2k$ from $\{0,1\}^{2k}$ for security parameter k.

4.5.1.3 Authenticated encryption

Authenticated encryption (AE) is a symmetric-key encryption mechanism.

The following flow is specified:

- **System setup:**
  - The setup party shall specify a security parameter k and output a symmetric key K_{AE}.
- **Message encryption:**
  - The encryptor shall take as input a message M of arbitrarily length, a symmetric key K_{AE}, and additional authenticated data AAD, and shall output the ciphertext C_{AE}.
- **Ciphertext decryption:**
  - The decryptor shall take as input a ciphertext C_{AE}, a key K_{AE}, and additional authenticated data AAD, and shall output a message M or output an error *err* that shall indicate that decryption has failed.

4.5.2 CCA-secure CP-ABKEM

A CCA-secure CP-ABKEM may be built from any CPA-secure CP-ABE scheme. The attribute universe of the CP-ABKEM shall be the attribute universe of the CP-ABE scheme. The messages size of the underlying CP-ABE scheme is set to $2k$ where k is the security parameter. The symmetric-key size shall be k.

The following flow is specified:

- **System setup:**
  - The setup party shall specify a security parameter k and shall run the system setup of the CP-ABE scheme, and shall return the CP-ABE master public key $MPK_{CP\text{-}ABE}$ and master secret key $MSK_{CP\text{-}ABE}$. (Note that $PARAMS_{CP\text{-}ABE}$ is contained in $MPK_{CP\text{-}ABE}$.)
- **Secret-key generation:**
  - The ABE authority shall take as input the master secret key $MSK_{CP\text{-}ABE}$ and an attribute set A from the attribute universe, and shall output a secret key SK_A obtained from the CP-ABE secret-key generation with $MSK_{CP\text{-}ABE}$ and A.
- **Symmetric-key encapsulation:**
  - The encryptor shall take as input the master public key $MPK_{CP\text{-}ABE}$ and an access policy AP, and shall choose a random symmetric key K from $\{0,1\}^k$ and a random bitstring r from $\{0,1\}^k$.
  - The encryptor shall run the message encryption of the CP-ABE scheme with $MPK_{CP\text{-}ABE}$, message $M=K||r$, access policy AP, and random tape $R=H(r||K||AP)$ to obtain a ciphertext C_{AP}. (Note that H is a collision resistant hash function as defined in 4.5.1.1.)
  - The encryptor shall output the symmetric key K and ciphertext C_{AP}.
- **Symmetric-key decapsulation:**
  - The decryptor shall take as input the secret key SK_A and a ciphertext C_{AP}, and shall run ciphertext decryption of the CP-ABE scheme with SK_A and C_{AP} to obtain $M=K||r$.

- If $K||r$ is not equal to the error *err*, the decryptor shall run message encryption of the CP-ABE scheme with $MPK_{CP\text{-}ABE}$, message $K||r$, access policy AP, and random tape $R=H(r||K||AP)$ to obtain a ciphertext C'_{AP}.
- The decryptor shall output the symmetric key K if $C'_{AP}=C_{AP}$, otherwise output error *err*.

4.5.3 CCA-secure KP-ABKEM

A CCA-secure KP-ABKEM may be built from any CPA-secure KP-ABE scheme. The attribute universe of the KP-ABKEM shall be the attribute universe of the KP-ABE scheme. The messages size of the underlying KP-ABE scheme is set to $2k$ where k is the security parameter. The symmetric-key size shall be k.

The following flow is specified:

- **System setup:**
  - The setup party shall specify a security parameter k and shall run the system setup of the KP-ABE scheme, and shall return the KP-ABE master public key $MPK_{KP\text{-}ABE}$ and master secret key $MSK_{KP\text{-}ABE}$. (Note that $PARAMS_{KP\text{-}ABE}$ is contained in $MPK_{KP\text{-}ABE}$.)
- **Secret-key generation:**
  - The ABE authority shall take as input the master secret key $MSK_{KP\text{-}ABE}$ and an access policy AP, and shall output a secret key SK_{AP} obtained from KP-ABE secret-key generation with $MSK_{KP\text{-}ABE}$ and AP.
- **Symmetric-key encapsulation:**
  - The encryptor shall take as input the master public key $MPK_{KP\text{-}ABE}$ and an attribute set A from the attribute universe, and shall choose a random symmetric key K from $\{0,1\}^k$ and a random bitstring r from $\{0,1\}^k$.
  - The encryptor shall run the message encryption of the KP-ABE scheme with $MPK_{KP\text{-}ABE}$, message $K||r$, attribute set A, and random tape $R=H(r||K||A)$ to obtain a ciphertext C_A.
  - The encryptor shall output the symmetric key K and ciphertext C_A.
- **Symmetric-key decapsulation:**
  - The decryptor shall take as input the secret key SK_{AP} and a ciphertext C_A, and shall run ciphertext decryption of the KP-ABE scheme with SK_{AP} and C_A to obtain $K||r$.
  - If $K||r$ is not equal to the error *err*, the decryptor shall run message encryption of the KP-ABE scheme with $MPK_{KP\text{-}ABE}$, message $K||r$, attribute set A, and random tape $R=H(r||K||A)$ to obtain a ciphertext C'_A.
  - The decryptor shall output the symmetric key K if $C'_{AP}=C_{AP}$, otherwise output error *err*.

4.5.4 CCA-secure CP-ABE

A CCA-secure CP-ABE scheme with arbitrary message size may be built from any CCA-secure CP-ABKEM scheme and an AE scheme. The attribute universe of the CP-ABE scheme shall be the attribute universe of the CP-ABKEM. Let R be a uniformly random bitstring of from $\{0,1\}^k$.

The following flow is specified:

- **System setup:**
  - The setup party shall specify a security parameter k and shall run the system setup of the CP-ABKEM, and shall return the CP-ABKEM master public key $MPK_{CP\text{-}ABKEM}$ and master secret key $MSK_{CP\text{-}ABKEM}$. (Note that $PARAMS_{CP\text{-}ABKEM}$ is contained in $MPK_{CP\text{-}ABKEM}$.)

- **Secret-key generation:**
  - The ABE authority shall take as input the master secret key $MSK_{CP\text{-}ABKEM}$ and an attribute set A from the attribute universe, and shall output a secret key SK_A obtained from the CP-ABKEM secret-key generation with $MSK_{CP\text{-}ABKEM}$ and A.
- **Message encryption:**
  - The encryptor shall take as input the master public key $MPK_{CP\text{-}ABKEM}$, a message M and an access policy AP, and a random tape R, and shall run symmetric-key encapsulation of the CP-ABKEM scheme with $MPK_{CP\text{-}ABKEM}$, AP, and R, to obtain a symmetric key K_{AE} and ciphertext C'_{AP}.
  - The encryptor shall run AE encryption with key K_{AE}, message M, and $AAD=C_{AP}$, to obtain the AE ciphertext $C_{AE.}$
  - The encryptor shall output the CP-ABE ciphertext $C_{AP}=(C'_{AP},C_{AE})$.
- **Ciphertext decryption:**
  - The decryptor shall take as input the secret key SK_A and a ciphertext $C_{AP}=(C'_{AP},C_{AE})$, and shall run symmetric-key decapsulation of the CP-ABKEM with SK_A and C'_{AP} to obtain K_{AE}.
  - If K_{AE} is not equal to the error *err*, the decryptor shall run AE decryption with key K_{AE}, ciphertext C_{AE}, and $AAD=C'_{AP}$, to obtain the message $M_.$
  - If M is not equal to the error *err*, the decryptor shall output the message M.

4.5.5 CCA-secure KP-ABE

A CCA-secure KP-ABE scheme with arbitrary message size may be built from any CCA-secure KP-ABKEM scheme and an AE scheme. The attribute universe of the KP-ABE scheme shall be the attribute universe of the KP-ABKEM. Let R be a uniformly random bitstring from $\{0,1\}^k$.

The following flow is specified:

- **System setup:**
  - The setup party shall specify a security parameter k and shall run the system setup of the KP-ABKEM, and shall return the KP-ABKEM master public key $MPK_{KP\text{-}ABKEM}$ and master secret key $MSK_{KP\text{-}ABKEM}$. (Note that $PARAMS_{KP\text{-}ABKEM}$ is contained in $MPK_{KP\text{-}ABKEM}$.)
- **Secret-key generation:**
  - The ABE authority shall take as input the master secret key $MSK_{KP\text{-}ABKEM}$ and an access policy AP, and shall output a secret key SK_{AP} obtained from the KP-ABKEM secret-key generation with $MSK_{KP\text{-}ABKEM}$ and AP.
- **Message encryption:**
  - The encryptor shall take as input the master public key $MPK_{KP\text{-}ABKEM}$, a message M, an attribute set A from the attribute universe, and random tape R, and shall run symmetric-key encapsulation of the KP-ABKEM scheme with $MPK_{KP\text{-}ABKEM}$, A, and R, to obtain a symmetric key K_{AE} and ciphertext C'_A.
  - The encryptor shall run AE encryption with key K_{AE}, message M, and $AAD=C'_A$, to obtain the AE ciphertext $C_{AE.}$
  - The encryptor shall output the KP-ABE ciphertext $C_A=(C'_A,C_{AE})$.
- **Ciphertext decryption:**
  - The decryptor shall take as input the secret key SK_{AP} and a ciphertext $C_A=(C'_A,C_{AE})$, and shall run symmetric-key decapsulation of the KP-ABKEM scheme with SK_{AP} and C'_A to obtain K.
  - If K_{AE} is not equal to the error *err*, the decryptor shall run AE decryption with key K_{AE}, ciphertext C_{AE}, and $AAD=C'_A$, to obtain the message $M_.$

- If M is not equal to the error err, the decryptor shall output the message M.

4.6 Requirements for compliant ABKEMs

4.6.1 General

The clause 4.2 specifies:

- CP-WATERS-KEM as a CP-ABKEM,
- CP-FAME-KEM as a CP-ABKEM,
- KP-FAME-KEM as a KP-ABKEM,
- KP-GPSW-KEM as a KP-ABKEM.

However, other CP-ABKEMs and KP-ABKEMs may be used if they comply with the following requirements.

4.6.2 Requirement 1: correctness and indistinguishability under chosen-plaintext attacks for ABKEMs

4.6.2.1 Correctness

The considered CP-ABKEM mechanism shall be correct in the sense that for any security parameter, any system parameters, any master secret and public keys output by the system setup, any secret keys output by the secret-key generation for any attributes from the attribute universe, any access policy, any symmetric key $K_{CP\text{-}ABKEM}$ and ciphertext output by the symmetric-key generation, the symmetric-key decapsulation outputs the same symmetric key $K_{CP\text{-}ABKEM}$.

The considered KP-ABKEM mechanism shall be correct in the sense that for any security parameter, any system parameters, any master secret and public keys output by the system setup, any secret keys output by the secret-key generation for any policy, any attribute set from the attribute universe, any symmetric key $K_{KP\text{-}ABKEM}$ and ciphertext output by the symmetric-key generation, the symmetric-key decapsulation outputs the same symmetric key $K_{KP\text{-}ABKEM}$.

4.6.2.2 Indistinguishability under chosen-plaintext attacks

The considered ABKEM mechanism shall admit a reductionist security proof of its indistinguishability under chosen-plaintext attacks (CPA) as formulated below. The CPA security shall be proven under a complexity assumption that is known to provide 128, 192, 256, 384 and 512-bit security (see Requirement 2 below). Security proofs that hold in the Standard Model (SM), the Random Oracle Model (ROM), the Generic Group Model (GGM) or a combination of those are approved.

Indistinguishability under chosen-plaintext attacks is defined as the resistance against all efficient attackers $\mathcal{A}$ (that is, probabilistic algorithms running in polynomial time depending on the security parameter) that engage, together with a challenger algorithm $\mathcal{C}$, into the following random experiment.

1) $\mathcal{A}$ selects
    - **CP-ABKEM:** an access policy P',
    - **KP-ABKEM:** a set of attributes S',
    - and provides those to $\mathcal{C}$.
2) The challenging algorithm $\mathcal{C}$ runs the system setup to generate $(params, mpk, msk)$ and provides mpk to $\mathcal{A}$.

3) $\mathcal{A}$ selects:

   - **CP-ABKEM:** a set of attributes S,
   - **KP-ABKEM:** an access policy P,

   - and provides those to $\mathcal{C}$.

4) $\mathcal{C}$ runs the secret-key extraction algorithm on msk as well as on:

   - **CP-ABKEM:** the set of attributes S,
   - **KP-ABKEM:** the access policy P,

   - and provides the resulting secret key to $\mathcal{A}$. Steps 3. and 4. may be repeated as often as $\mathcal{A}$ desires. The constraint shall be that S shall not satisfy P' (**CP-ABKEM**) and S' shall not satisfy P (**KP-ABKEM**).

5) $\mathcal{C}$ runs symmetric-key encapsulation on mpk and:

   - **CP-ABKEM:** the access policy P',
   - **KP-ABKEM:** the set of attributes S',

   - to generate a pair (K_0, C) and sends C to $\mathcal{A}$.

6) Steps 3. and 4. are repeated.

7) $\mathcal{C}$ chooses $K_1$with $|K_0| = |K_1|$ and a bit b from $\{0,1\}$ at random.

8) $\mathcal{C}$ sends K_b to $\mathcal{A}$.

9) $\mathcal{A}$ returns a bit $\hat{b}$.

10) $\mathcal{C}$ returns 1 if $\hat{b} = b$ or 0 otherwise.

The success probability of $\mathcal{A}$ is the probability that $\mathcal{C}$ returns 1. Indistinguishability under chosen-plaintext attacks is realized when the success probability of any efficient attacker $\mathcal{A}$ is proven to be negligible in the security parameter κ.

A CP-ABKEM or a KP-ABKEM is CPA-secure if indistinguishability under chosen-plaintext attacks is realized in the sense of the above.

4.6.3 Requirement 2: Sufficient security levels

The considered mechanism shall admit values of the security parameter κ that provide 128, 192, 256, 384 and 512-bit security.

4.7 Revocation

4.7.1 Attribute revocation

Attribute may be revoked:

- by providing an attribute-revocation list:

  - Once attributes are revoked, the corresponding attributes shall be included in the latest version of the attribute-revocation list.
  - The encapsulating party shall use the latest available version of the attribute-revocation list and shall not consider attributes included in the attribute-revocation list.
  - The key-generation party shall use the latest available version of the attribute-revocation list and shall not consider attributes included in the attribute-revocation list.

- by universe regeneration, i.e. by providing an entire new ABKEM with redistribution of a new master public key, a new master-secret key, and new secret keys (see clause 5.2.4 for ABE Public Key revocation).

4.7.2 Secret-key revocation

Secret keys may be revoked:

- by introducing time intervals:
  - The encryptor shall:
    - Define an expiration timestamp as a constant (EXP_TIME).
    - Include a condition on EXP_TIME in any access policy and attribute used to encapsulate a symmetric key.
  - The decryptor shall:
    - Receive attributes and access policy asserting the current date/time (CUR_TIME).
    - Use the secret key associated to attributes and access policy to match the expiration condition (CUR_TIME<EXP_TIME) in step 2 above.
- By universe regeneration, i.e. providing an entire new ABKEM:
  - The setup party shall redistribute a new master public key, a new master secret key, and new user secret keys (see clause 5.2.4 for ABKEM Public Key revocation).

4.8 Recommendations

4.8.1 Overview

In clause 4.8.2, recommendations are given concerning the efficiency of the CP-ABE and KP-ABE schemes. In clause 4.8.3, recommendations are given concerning the security of CP-ABE and KP-ABE schemes.

4.8.2 Efficiency considerations

If repeating attributes per policy are required and fast ciphertext decryption is not required, the CP-ABE scheme with CP-WATERS-KEM (clauses 4.5.4 and 4.2.2) and the KP-ABE scheme with KP-GPSW-KEM (clauses 4.5.5 and 4.2.4) should be used. If fast ciphertext decryption is required and repeating attributes per policy is not required, the CP-ABE scheme with CP-FAME-KEM (clauses 4.5.4 and 4.2.3.3) and the KP-ABE scheme with KP-FAME-KEM (clauses 4.5.5 and 4.2.3.4) should be used. Table 4.1 summarize the efficiency considerations for CP-ABE and KP-ABE schemes.

Table 4.1: Efficiency considerations for CP-ABE and KP-ABE schemes

Properties required	CP-ABE with CP-WATERS-KEM	CP-ABE with CP-FAME-KEM	KP-ABE with KP-FAME-KEM	KP-ABE with KP-GPSW-KEM
Repeating attributes per policy	Yes	No	No	Yes
Fast ciphertext decryption	No	Yes	Yes	No

4.8.3 Security considerations

CCA-secure CP-ABE and KP-ABE (4.5.4 and 4.5.5) schemes should be used. In scenarios where the CPA-security notion is sufficient, CPA-secure CP-ABE and KP-ABE schemes (4.4.2 and 4.4.3) may be chosen.

5 Trust models

5.1 Overview

Notwithstanding the possibility of applying the specifications provided by the present document to different use cases, this clause specifies three general trust models under which several use cases may be framed.

These trust models differ for the involved actors, their roles and the specification of functions they shall execute.

The three trust models hereunder defined are:

- Long term storage
- Offline access control
- Platform Provider

The different trust models apply to use cases identified in ETSI TS 103 458 [i.3].

5.2 Roles

5.2.1 Data Consumer

A Data Consumer is a natural or legal person, public authority, agency or any other body accessing data for a given purpose.

5.2.2 Data Controller

A Data Controller is a natural or legal person, public authority, agency or any other body which alone or jointly with other data controllers determines the purposes and means of the processing of personal data.

5.2.3 Data Processor

A Data Processor is a natural or legal person, public authority, agency or any other body which processes personal data on behalf of the Data Controller.

5.2.4 Data Subject

A Data Subject is an identifiable person, i.e. a person who can be identified, directly or indirectly, in particular by reference to an identification number or to one or more factors specific to his physical, physiological, mental, economic, cultural or social identity.

5.2.5 Device Manager

A device manager is a natural or legal person, public authority, agency or any other body which manages and configures a device.

5.2.6 Platform Provider

A Platform Provider is a natural or legal person, public authority, agency or any other body which provides services necessary to run and support a platform. These services shall include identity management for the Platform Users (Pu) and any interfaces (e.g. API, components, etc.) allowing third parties to use and possibly extend the platform services. The Platform Provider acts as a Data Processor and may act as a Data Controller with regard to its users' data. See clause 6.1 for more details.

5.2.7 Third Party Service Provider

- A Third Party Service Provider (3SP) is a natural or legal person, public authority, agency or any other body which provides support to the provisioning of services by Platform Provider, or to the consumption of service by a Platform User. The 3SP may be a specialization of Data Processor and of Data Controller. The 3SP may be in business relationship with the Data Subject or other stakeholders. See clause 6.1 for more details.

5.2.8 Platform User

- A Platform user (Pu) is a Data Subject in business relation with a Platform Provider consuming one or more services of the Platform. The Platform User is typically assigned an identifier (PuID) by the Platform Provider. See clause 6.1 for more details.

5.3 Models

5.3.1 Long term storage

The aim of this model is to preserve the confidentiality of (static) data, when data is located in an untrusted storage. A user can process confidential data inside a secure environment of a device. Afterwards, the user may need to move the data to an untrusted storage, either physically owned (i.e. a local storage) or not (i.e. a remote storage like a public Cloud storage service). In order to preserve the confidentiality of such data and to avoid unwanted disclosure to unauthorized parties (possibly including the storage owner if different from the user), the data is encrypted and key material is kept separate from the storage. Using an ABKEM cryptosystem the user can define access control policies for the ciphertext similar to those originally defined for the data before encryption (plaintext), and automatically enforce them when a data consumer requires the ciphertext.

In this simple model, two roles are identified: the user and the data consumer. The data consumer may be a data processor. For the provisioning of key material, the user shall trust an ABE Authority which may be under control of the user or of an external actor.

The following flow is specified:

- The ABE Authority shall execute the setup process.
- The user shall be provisioned with the master public key generated by the ABE Authority.
- The user shall define the applicable policy.
- Through an ABKEM algorithm running in a secure execution environment and using the ABE public key, a symmetric key is generated and encapsulated. The symmetric key shall be used to encrypt the plaintext.
- The user shall move the ciphertext to the target untrusted storage.
- The user shall be responsible for revealing the location of the ciphertext to data consumers.
- Upon a data download request from a data consumer, the untrusted storage shall respond with the ciphertext.
- The ABE Authority shall execute the key generation process.
- Either the ABE Authority or the user on its behalf or any other intermediary provisioned with the master secret key by the ABE Authority shall be responsible to distribute a secret key to data consumers.
- To access the data, the data consumer shall decapsulate the symmetric key and use it to decrypt the ciphertext.

To support change of policies, attributes revocation and secret key revocation, the user shall replace the old ciphertext with a new one encrypted using a new symmetric key encapsulation generated from the new policy, attributes, expiration timestamps. The data consumer shall be provisioned with a new secret key.

The aforementioned procedure shall be executed also in case of regeneration of the ABE universe (revocation of the master public key) after the user has been provisioned with the new master public key generated by the ABE Authority.

5.3.2 Offline access control

The aim of the present model is to protect the confidentiality of data on a device which has no connectivity with other devices. Due to the specific capabilities of some devices (e.g. industrial devices), operational constraints or costs of the communication links, the unavailability of a central authority supporting access control and policy management may make difficult the definition and maintenance of data access control policies, especially in a multi-stakeholder environment.

The data may be generated by the device (data flow) or may simply be stored on the device (static data).

Three roles are defined:

- the user - may be a data subject (if the data in question refers to the user) or not.
- the device manager - may coincide with the user or be a different stakeholder.
- the data consumer - may be a data processor or not.

For the provisioning of key material, the device manager shall trust an ABE Authority which may be under the control of the device manager or of an external entity.

The following flow is specified:

- The ABE Authority shall execute the setup process.
- The device manager shall define the applicable policy.
- The device manager shall preload the target device with the master public key and the defined policy.
- The ABE Authority shall execute the key generation process.
- Either the ABE Authority or the device manager on its behalf or any other intermediary provisioned with the master secret key by the ABE Authority shall be responsible to distribute a secret key to data consumers.
- Upon a data download request from a data consumer, the device shall execute a new key encapsulation algorithm and produce a new symmetric key and its encapsulation. The device shall use the symmetric key to encrypt the plaintext. The device shall return the ABE ciphertext to the data consumer.
- For improved confidentiality and performance, the device may cache encrypted data.
- To access the data, the data consumer shall decapsulate the symmetric key and use it to decrypt the ciphertext.

To support change of policies, revocation of keys/attributes and expiration, the device manager shall replace any old policy preloaded on the device with the corresponding new one. The device shall execute a new symmetric key generation and encapsulation algorithm. The device shall proceed with re-encryption of any cached ciphertext encrypted with an old symmetric key with a new one encrypted with the actual symmetric key. The data consumer shall be provisioned with the new secret key.

The aforementioned procedure shall be executed also in case of regeneration of the ABE universe (revocation of the master public key) after the device manager has been provisioned with the new master public key generated by the ABE Authority.

5.3.3 Platform Provider

The "Platform Provider" model specified in this clause considers three roles:

- The Platform user (Pu) is a data subject and shall subscribe to services offered by the Platform Provider. The Platform user trusts the Platform Provider, who typically manages and has full access to her data (including identifiers, PII, personal data).

- The Platform Provider (PP), is a specialization of the Identity Provider model described in ETSI TR 187 010 [i.7]. In particular, the Platform Provider shall be an Identity Provider and shall offer additional functions other than identity management (e.g. API, marketplace services, etc.) to Third Party Service Providers. These functions may insist on different kinds of data related to the data subject (including identifiers, PII, personal data). The Platform Provider is a data processor (and can be data controller as well, processing user's data for its own purposes).
- The Third Party Service Provider(s) (3SP) is a Relying Party as defined by ETSI TR 187 010 [i.7], may provide the Pu with additional services and may play the role of data processor, obtaining Pu's data from the PP (3SP can be data controller as well, processing user's data for its own purposes).

NOTE: The "Platform Provider" model described in the present document is a generalization of the "Device Platform Provider" (DPP) model described in ISO/IEC 19944 [i.9], which is based on the Cloud Service Provider model described ISO/IEC 17789 [i.8]. Compared to the DPP model, the present model:

- ▪ does not necessarily assume the presence of Cloud services;
- ▪ does not necessarily assume the presence of a physical, hardware device extending the platform functionalities.

In the present clause, ABE is used to protect the confidentiality of Pu's data and to govern its access by 3SPs using policies defined by both the Pu and the PP.

- The ABE Authority shall execute the setup process.
- The ABE Authority may be under the control of the PP.
- The PP and the Pu shall define the applicable set of policies over the Pu's data.
- Upon a Pu's data request from a 3SP, the PP shall encrypt the requested data through a symmetric key generated by an encapsulation algorithm using the applicable policies. The PP shall return the ciphertext to the 3SP.
- The PP may cache the ciphertext to improve performances.
- The 3SP may subscribe to Pu's data change via PP subscription service if available. In this case, upon a subscribed data change, the PP shall encrypt the changed data through a symmetric key generated by an encapsulation algorithm using the applicable policies. The PP shall distribute the resulting ciphertext to 3SPs who have subscribed to this data change.
- The ABE Authority shall execute the key generation process.
- The PP shall be responsible for distributing the secret key to legitimate 3SP.
- To access the data, the 3SP shall decapsulate the symmetric key and use it to decrypt the ciphertext.

To support change of policies, revocation of keys/attributes and expiration, the PP shall replace any old policy with the corresponding new one. The PP shall execute a new symmetric key generation and encapsulation algorithm and - encrypt any cached ciphertext encrypted with outdated symmetric keys (stale ciphertext). Any 3SP shall be provisioned with a new secret key.

The aforementioned procedure shall be executed also in case of regeneration of the ABE universe (revocation of the master public key) after the PP has been provisioned with the new master public key generated by the ABE Authority.

5.4 Functions

5.4.1 Authority function

Each trust model shall implement the set up and maintenance of the ABKEM universe, including the maintenance of an attribute revocation list. The execution of these functions shall happen on a trusted execution environment. In the Platform Provider trust model, the trusted execution environment may be part of a Platform.

5.4.2 Assertion function

5.4.2.1 General

For the purpose of the present document, assertions are statements being made by an authority about a property of a target (a subject or an object) for access control decisions. Two different types of assertions are specified: assertions relevant for data access (data access assertions), and assertions relevant to data capture (data capture assertions). Each trust model may implement assertion functions. In such a case, source of data access assertions and data capture assertions shall be identified as in table 5.1.

5.4.2.2 Data access assertion

Data access assertions may be used in the evaluation of access control policies when the policy relies on properties of the subject. Data access assertions are assertions concerned with properties (attributes) of the subject as well as the environment. Data access assertions shall be produced when a subject attempts to gain access to an object, close in time to the access control decision.

NOTE: This kind of properties includes, for example, location and time of access, identity and authentication of the subject, attributes of the subject such as type, role, and clearance.

5.4.2.3 Data capture assertion

Data capture assertions may be used during the evaluation of an access control policy when the policy relies on properties of the data. Data capture assertions are assertions concerned with the trustworthiness of properties of the captured data. Data capture assertions shall be produced at the time the data is captured.

NOTE: This kind of properties includes, for example, location and time of the capture, type of the capturing device and level of assurance that can be given to data accuracy, type and format of the data itself, information about intermediate processing nodes.

5.4.3 Encryption function

Each trust model shall implement the encryption function. The source of data may coincide with the encryptor. However, the source of data may delegate encryption to a trusted entity.

Static data is assumed not to vary for a relatively long period, therefore the ciphertext shall be considered outdated only in case of attribute revocation or secret key expiration. In such cases, the encryptor shall produce a new ciphertext, taking care of revoked attributes.

In case of data flow, the encryptor shall additionally replace the ciphertext with a new one generated from the most recent version of the information available in the data flow once the data varies.

The encryption shall be performed on a trusted execution environment which may be part of a device (for the Offline access control trust model) or of a platform (for the Platform Provide trust model).

5.4.4 Policy Management function

Each trust model shall implement the policy management function. For each trust model, the actor responsible for generating and maintaining the set of policies shall be identified as specified in table 5.1.

5.4.5 Key distribution function

Each trust model shall implement the key distribution function. The role responsible for key material distribution shall be identified as specified in table 5.1. For the Platform Provide trust model, the Platform Provider may use a PKIX infrastructure as specified in clause 6.2. The infrastructure shall take care of issuing and distributing certificates and maintaining Certificate Revocation Lists (CRLs).

5.4.6 Decryption function

Each trust model shall implement the decryption function. For each trust model, the ciphertext transmission flow shall happen as specified in table 5.1. The decryptor shall take the role specified in table 5.1.

Table 5.1: Summary of trust models

Function	Long Term Storage	Offline Access Control	Platform Provider
Data source	Any	Device	Platform
Data type	Static data	Static data Data flow	Data flow
Setup party and key generation party	on a Trusted Execution Environment	on a Trusted Execution Environment	on Platform
Choice and maintenance of policies	User	Device manager	Platform User, Platform Provider
Key material distribution by	User	Device manager	Platform Provider
Source of assertions	Trusted Execution Environment	Trusted Execution Environment	Platform
Encryption on	Trusted Execution Environment	Device	Platform
Ciphertext transmission	- User to Long Term Storage - Long Term Storage to Data consumer	Device to Data consumer	Platform Provider to Third Party Service Provider
Decryption	Data consumer	Data consumer	Third Party Service Provider

6 Procedures for distributing attributes and keys

6.1 Introduction

The present clause specifies procedures for securely distributing attributes and secret keys when the Platform Provider trust model is adopted.

6.2 Platform Provider extended with Public Key Infrastructure X.509

6.2.1 Overview

This clause specifies the use of RSA cryptography along with ABE cryptography to increase protection against some attacks that may threat the Platform Provider trust model.

In a first attack scenario, a malicious user which has received the master public key is enabled to perform encryption operations. Thus, a malicious user could impersonate a PP entity and provide a fake secret key to a 3SP. The 3SP can use the secret key to get access to fake information. Instead, by using RSA-based signatures and X.509 based certificates, the 3SP can verify the PP's signature and thus authenticate the PP when receiving the key material, as described in clause 6.2.3.4.

In a second scenario similar to the one above described, a malicious user could try to impersonate the ABEA. In this case the fake ABEA could provide to a target 3SP an illegitimate secret key, which would not allow the decryption of ciphertext coming from the legitimate PP, and would potentially allow the decryption of ciphertext coming from a fake PP. This scenario may be avoided by using RSA-based signatures and X.509 certificates as described in clause 6.2.3.4.

Finally, in a third potential threat, a malicious user could eavesdrop the secret key material transferred from the ABEA to the 3SP. This would allow the eavesdropper to decrypt encrypted ciphertext meant for the 3SP. Relying on Key Transport as described in clause 6.2.3.4 provides increased security against this threat.

6.2.2 Entities

6.2.2.1 Introduction

IETF RFC 5280 [i.5] describes a traditional PKI infrastructure. The present document introduces three additional architectural elements that shall be present in the system:

- the ABE Authority (ABEA);
- the Third Party Service Provider (3SP);
- the Platform Provider (PP).

Figure 6.1 summarizes the ABE-extended PKI architecture.

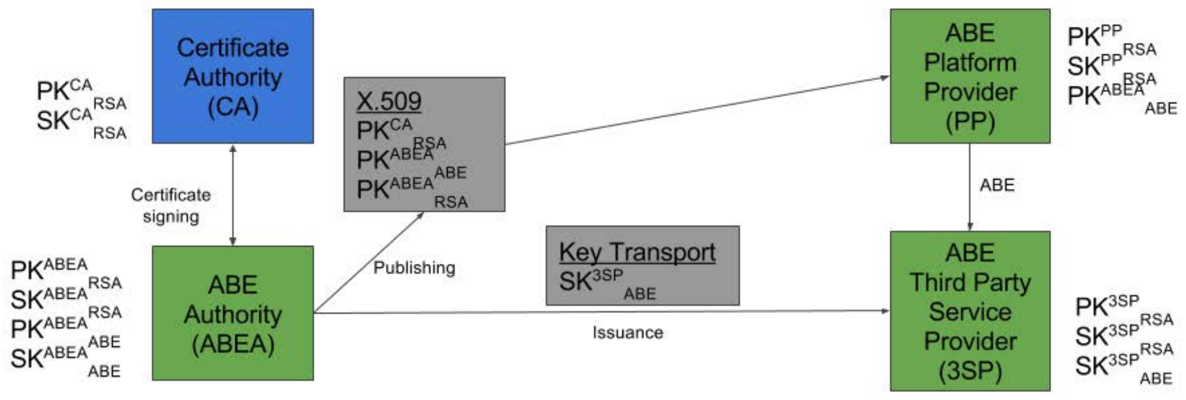

Figure 6.1: ABE-Extended X.509 PKI Summary

6.2.2.2 ABE Authority (ABEA)

In PKI X.509, Certification Authorities (CAs) are in charge of the following tasks:

- issuing X.509 certificates to users (EE)
- maintaining Certificate Revocation Lists (CRLs)

In the present document, an ABE Authority (ABEA) is in charge of the following tasks:

- setting up the ABKEM system
- creating and distributing the secret keys to Third Party Service Providers (3SP);

NOTE 1: An ABE Authority does not replace a Certification Authority as it provides different functions.

NOTE 2: The ABE Authority can be controlled by the same administrative entity that controls the PP.

The ABE Authority is associated to the following keys/parameters:

- PK^{ABEA}_{RSA}, SK^{ABEA}_{RSA}: RSA public and secret key pair of the ABEA;
- PK^{ABEA}_{ABE}, SK^{ABEA}_{ABE}: ABE master public key and master secret key of the ABEA.

The ABEA shall never disclose at any time the secret keys (SK^{ABEA}_{RSA} and SK^{ABEA}_{ABE}).

6.2.2.3 Keys associated to the Third Party Service Provider (3SP)

The 3SP shall be associated to the following keys/parameters:

- PK^{3SP}_{RSA}, SK^{3SP}_{RSA}: the RSA public and secret key pair of the 3SP;
- SK^{3SP}_{ABE}: the ABE secret key of the 3SP. 3SP uses these key for decapsulation.

The 3SP shall never disclose at any time the secret keys (SK^{3SP}_{RSA} and SK^{3SP}_{ABE}).

6.2.2.4 Keys associated to the Platform Provider (PP)

The PP is associated to the following keys/parameters:

- PK^{PP}_{RSA}, SK^{PP}_{RSA}: the RSA public and secret key pair of the PP.

The PP shall keep secret the secret key (SK^{PP}_{RSA}).

6.2.3 ABE Key Distribution

6.2.3.1 General

The distribution of the ABE keys and policies shall be performed as described in the following clauses.

6.2.3.2 Setup

The ABEA shall generate the following keys:

- PK^{ABEA}_{RSA}, SK^{ABEA}_{RSA}: RSA public and secret key pair of the ABEA;
- PK^{ABEA}_{ABE}, SK^{ABEA}_{ABE}: ABE master public key and master secret key of the ABEA.

The PP shall generate the following keys:

- PK^{PP}_{RSA}, SK^{PP}_{RSA}: the RSA public and secret key pair of the PP.

The 3SP shall generate the following keys:

- PK^{3SP}_{RSA}, SK^{3SP}_{RSA}: the RSA public and secret key pair of the 3SP.

6.2.3.3 ABE Public Key distribution

The CA shall issue a CA-signed (extended) X.509 certificate that shall contain the ABEA public keys PK^{ABEA}_{RSA} and PK^{ABEA}_{ABE}. The certificate shall contain also information on supported attributes along with global cryptographic parameters, as described in clause 6.2.3.5. The ABEA shall publish this certificate.

The PP shall receive the key PK^{ABEA}_{ABE} embedded in this certificate, along with the global cryptographic parameters, and shall use it for encryption.

6.2.3.4 ABE secret key material distribution

Both the PP and the 3SP shall submit a certificate signing request to the CA using X.509 standard protocols and shall obtain corresponding certificates containing PK^{PP}_{RSA} and PK^{3SP}_{RSA} respectively. The PP and the 3SP shall obtain offline the root certificate of the CA beforehand.

The PP and the 3SP shall publish the aforementioned certificates.

The PP shall authenticate to 3SPs by means of a mutual RSA authentication, through the X.509 certificates containing PK^{PP}_{RSA} and PK^{3SP}_{RSA}.

The certificate containing PK^{3SP}_{RSA} shall be used by the 3SP to authenticate to the ABEA, in order to obtain the ABE secret key SK^{3SP}_{ABE}.

The secret key SK^{3SP}_{ABE} shall be transferred from the ABEA to the 3SP by means of KTS-OAEP Key Transport as specified in NIST SP 800-56B [2].

The 3SP shall use the SK^{3SP}_{ABE} to decapsulate the symmetric key and use it to decrypt the ciphertext generated by the PP. The 3SP shall never disclose the secret key SK^{3SP}_{ABE}.

6.2.3.5 Attributes distribution

The ABE authority may be willing to limit the use of attributes for encapsulation, by limiting the issuing of secret keys to only those related to valid attributes. In this case, the 3SP shall discard secret keys related to not valid attributes.

The 3SP may receive a list of valid attributes embedded in a certificate. To this end, the CA shall issue a CA-signed (extended) X.509 certificate that shall contain the ABEA valid attributes along with cryptographic global parameters. This certificate is the same as defined in clause 6.2.3.3 and published by the ABEA.

6.2.4 ABE Public Key revocation

To revoke an ABEA PK^{ABEA}_{ABE} key the ABEA shall instruct the CA to revoke the corresponding X.509 certificate by adding this certificate to its CRL.

The ABEA shall regenerate the ABE universe issuing a new master public key and a new master secret key (PK^{ABEA}_{ABE}, SK^{ABEA}_{ABE}) and repeat the distribution procedures described in clauses 6.2.3.3, 6.2.3.4 and 6.2.3.5.

6.3 Assertions

6.3.1 Introduction

Assertions may be widely used to convey semantics between parties involved in an ABE system. The present clause specifies several types of assertions and their binding.

6.3.2 Types of assertions

The present document defines the following types of assertions:

A (data access assertions) - In large universe CP-ABEKEM, the key generation party may use data access assertions to check attribute values of the decapsulating party before generating a related secret key and delivering it to the decapsulating party.

B (data capture assertions) - In large universe KP-ABEKEM, the decapsulating party may use data capture assertions to check attribute values of the encapsulating party.

NOTE: An instance of this case is, for example, a data consumer that checks whether a given device presents valid properties (GPS position, temperature, etc.) at the time the device encrypted the data.

C (data use statements) - The decapsulating party may combine data capture assertions and data access assertions to claim data use statements to the encapsulating party. Data use statements are described in ISO/IEC 19944 [i.9].

D (attribute assertions) - The ABE authority may be willing to limit the use of valid attributes for encapsulation, by limiting the issuing of secret keys containing policies satisfied by those attributes only (in large universe KP-ABEKEM) or containing only sets of valid attributes (in large universe CP-ABEKEM). The decapsulating party may use assertions to check if one attribute or more attributes belongs to the list of valid attributes. The decapsulating party shall discard secret keys related to not valid attributes. The asserting party is the ABE authority.

E (public key assertions) - The setup party (or any other entitled system entity) may use assertions to transmit the Master Public Key to the encapsulating party.

F (secret key assertions) - The setup party (or any other entitled system entity) may use assertions to transmit secret keys to the decapsulating party.

6.3.3 Mapping to SAML

6.3.3.1 SAML Attributes

SAML attributes shall be described by a local "name" and a "nameformat" as specified in SAML specifications [5]. The "nameformat" shall contain an URI reference that shall be used to interpret the local "name".

To identify ABE attributes (assertion types A, B, C and D) the following rules shall apply:

- Different ABE universes shall be associated to different nameformats.
- Different attributes shall have different local names.
- Different versions of the same attribute should have different local names.

Attributes refers to a SAML subject. The subject shall be identified according to SAML system entity identification.

To identify ABE keys (assertion types E and F) the following rules shall apply:

- For type E (public key assertions), the local "name" shall contain the string "MSK".
- For type F (secret key assertions), the local "name" shall be formed by either the comma separated set of related attributes (CP-ABKEM) or the related monotone access policy (KP-ABEM) expressed as specified in clause 7.2.3.1 ("General definition of policies and syntax").

6.3.3.2 SAML Attribute Statements

6.3.3.2.1 Unencrypted format

Attribute statements shall be transported in SAML <AttributeStatement> element. Inside the <AttributeStatement> element:

The <Subject> element shall indicate the subject to which the attributes refer to.

The <Attribute> element shall contain the "name" and the "nameformat" XML attributes. In case the attribute statement is used in a response to an attribute query (clause 6.3.3.3), the "name" and the "nameformat" XML attributes shall match the corresponding fields in the query. Inside the <Attribute> element, the <AttributeValue> subelement shall supply the actual value of the attribute.

6.3.3.2.2 Encrypted format

<AttributeStatement> may contain attributes in an encrypted form. In this case, the <EncryptedAttribute> element shall appear on behalf of the <Attribute> element specified in clause 6.3.3.2.1. In particular, the content of the whole <Attribute> element shall be encrypted according to Syntax and Processing specification [12] and shall appear inside the <EncryptedData> sub-element of the <EncryptedAttribute> element. The <EncryptedKey> sub-element of the <EncryptedAttribute> shall contain a wrapped decryption keys, whose format shall be as defined in W3C XML [12].

6.3.3.3 SAML Attribute Queries

To carry out a query about one or more attributes, SAML <AttributeQuery> element shall be used. Inside the <AttributeQuery> element:

The <Subject> subelement shall indicate the subject to which the statement refers to.

The <Attribute> subelement shall indicate the attribute(s) to be returned. If omitted, it indicates all attributes allowed by policy. Attributes shall be identified as specified in clause 6.3.3.1.

Once a SAML attribute query is received, a SAML authority shall return assertions by attribute statements.

6.3.3.4 Key assertions

For type E (public key) assertions, the setup party (or any other entitled system entity) shall transmit the Master Public Key to the encapsulating party inside an <AttributeStatement> element, whose <Subject> subelement shall refer to the setup party and whose <Attribute> subelement shall refer to the Master Public Key.

For type F (secret key) assertions, the setup party (or any other entitled system entity) shall transmit secret keys to the decapsulating party inside an <AttributeStatement> element, whose <Subject> subelement shall refer to the decapsulating party and whose <Attribute> subelement shall refer to the secret key.

The <AttributeValue> subelement shall contain the secret key.

6.3.3.5 Security considerations

Depending on the particular use case, the assertion party may need to be authenticated, and integrity protection may be required. Mechanisms for authentication and message integrity defined for SAML [6] may apply.

Assertion may be signed according to SAML specification [5].

SAML confidentiality protection mechanism (through the <EncriptedAssertion> element) may apply when assertions need to pass through an untrusted intermediary.

6.3.4 SAML binding for CoAP

6.3.4.1 Message encapsulation

When CoAP [15] is used as a transport protocol for assertions described in clause 6.3.3, SAML Attribute Queries, SAML Attribute Statements, and key assertions shall be transported as payload in CoAP messages. The Content-Format option shall be set to "application/xml".

SAML messages shall not be compressed or otherwise binary-encoded prior to being sent over CoAP. When a SAML message is expected to cause fragmentation, it shall be transported using CoAP Block-Wise Transfer [16].

CoAP clients shall send SAML Attribute Queries as Confirmable messages and use the POST method. The response may be piggybacked or sent separately.

6.3.4.2 Addressing and intermediaries

The URI to access the SAML endpoint shall conform to IETF RFC 7252 [15], clause 6. CoAP proxies may be present on the path to the SAML endpoint, in which case the Proxy-URI option shall be set to URI of the SAML endpoint. If the SAML endpoint is accessible by HTTP, proxies shall act according to IETF RFC 7252 [15], clause 10.

SAML protocol messages transported over CoAP shall not be cached by CoAP proxies and endpoints. In order to enforce this, the CoAP origin server shall include a Max-Age Option set to 0 in the response message.

6.3.4.3 Security

SAML messages supported by this binding shall be digitally signed according to SAML [5] using ECDSA with SHA-256, as identified by "http://www.w3.org/2001/04/xmldsig-more#ecdsa-sha256". SAML Attribute Statements and key assertions shall be encrypted according to SAML [5], where the encryption algorithm shall be AES as defined in FIPS 197 [3].

7 Attribute Based Access Control layer

7.1 Overview

In the present clause access control using ABKEM is defined in a series of layers that allow bridging the capabilities of large universe ABKEM schemes with various access control methods, in particular standardized access control systems.

Each layer can be used in a standalone manner. Layer 1, defined in clause 7.2, is the closest to the capabilities of ABKEM schemes and provides binding rules meant for implementations directly manipulating ABKEM keys and ciphertexts. Layer 2 extends the capabilities of Layer 1 and is meant for higher-level implementations, such as pre-processors and compilers, that wish to extend ABKEM access control beyond the native capabilities of a schemes. Layer 2 offers the capabilities required for adaption to XACML.

7.2 Base ABKEM access control capabilities ("Layer 1")

7.2.1 Introduction

The present clause defines capabilities to be supported in ABKEM annotations and access structures for large universe schemes which may have the restriction that attributes are to be used only once.

7.2.2 Attributes

7.2.2.1 Syntax for attribute declaration

Attributes shall be declared with type, name, and maximum number of occurrence. Attribute declaration at Layer 1 shall comply with the following ABNF [13] specification:

```
attribute-decl = "define" SP attribute-def "." max-occurence [SP source-datatype] CRLF
attribute-def = attribute-type "." attribute-name
max-occurence = %x31-39 *DIGIT
source-datatype = 1*(VCHAR)
```

Valid values for the attribute-type element are specified in clause 7.2.2.2. Attribute names shall comply with the following ABNF [13] definition:

```
attribute-name = component *(sep-namespace component) *1(sep-extension component)
component = 1*(ALPHA / DIGIT)
sep-namespace = ":"
sep-extension = "-"
```

According to the above definition, allowed characters for attribute names shall be a subset of the ASCII character set as defined in ANSI INCITS 4-1986 [10].

The component separator `sep-namespace` shall be reserved for implementations providing namespaces for attributes names. This shall not be used to express external data types to which the attribute maps to. For this purpose, the `source-datatype` component shall be used.

The component separator `sep-extension` shall be reserved for implementations operating at Layer 2 as specified in clause 7.3.

Attribute names shall be case-sensitive. Attribute names (`attribute-name` element) shall be unique within the ABKEM universe in which they are declared. Implementations shall reject universe declarations in which more than one attribute bear the same name (as defined by the `attribute-name` element), even when said attributes are of different types.

7.2.2.2 Attribute types

The following attribute types shall be supported: unsigned integer, boolean, and string. Attribute type shall be declared in uppercase (e.g. BOOL) and may be appended parameters (e.g. UINT(2)).

The unsigned integer attribute type shall be defined as UINT(k), where k shall be a strictly positive integer. An attribute declared using the type UINT(k) shall have its value encoded over k bits, the base shall be base 2. The unsigned integer attribute type shall conform to the following ABNF [13] specification, where the value of k shall be that of the first `parameter` element:

```
attribute-type = attribute-type-uint
attribute-type-uint = "UINT(" parameter *(comma parameter) ")"
parameter = 1*DIGIT
comma = ","
```

The boolean attribute type shall conform to the following ABNF [13] specification:

```
attribute-type =/ attribute-type-bool
attribute-type-bool = "BOOL"
```

The string attribute type shall conform to the following ABNF [13] specification:

```
attribute-type =/ attribute-type-string
attribute-type-string = "STRING"
```

7.2.2.3 Syntax for ABKEM universe declaration

An ABKEM universe shall be declared conforming to the following ABFN [13] specification:

```
universe = version SP uni-type SP uni-id SP crypto-params CRLF 1*(attribute-decl)
version = 1*(ALPHA / DIGIT)
uni-type = "CP-ABKEM" / "KP-ABKEM"
uni-id = uni-name "." uni-version
uni-name = 1*(ALPHA / DIGIT)
uni-version = 1*(ALPHA / DIGIT / ":")
crypto-params = 1*(VCHAR)
```

The `version` component indicates the version of the specification to which the universe declaration is compliant. For universe declarations compliant to the present document, it shall be set to the string "1.1.1". Implementations shall reject universe declarations for values of the `version` component that they do not support.

Methods to ensure uniqueness of the ABKEM universe (e.g. via an official registry) are out of scope of the present document.

7.2.2.4 Syntax for value assignment in annotations

An annotation operation shall provide a reference to the ABKEM universe according to the following ABNF [13] specification:

```
universe-ref = "universe:" SP uni-id CRLF
```

An assignment operation for an attribute of type boolean shall conform to the following ABNF [13] specification:

```
assignment = "set:" SP attribute-type-bool "." attribute-name SP bool-value
bool-value = ("0" / "1") ; true or false
```

An assignment operation for an attribute of type unsigned integer shall conform to the following ABNF [13] specification:

```
assignment =/ "set:" SP attribute-type-uint "." attribute-name SP uint-value
uint-value = 1*(DIGIT)
```

The present document specifies two methods for assigning values to string attributes: plain string and encoded string. The plain string method provides a limited, human readable set of characters, while the encoded string method shall use base64 [4] encoding and allows full leverage of the source character set. For attributes of type string expressed with the plain string method, the assignment shall conform to the following ABNF [13] specification:

```
assignement =/ "set:" SP attribute-type-string "." attribute-name SP string-value-plain
string-value-plain = string-plain
string-plain = "string:plain:" string-plain-value
string-plain-value = 1*(SP / VCHAR)
```

For attributes of type string expressed with the encoded string method, the assignment shall conform to the following ABNF [13] specification:

```
assignement =/ "set:" SP attribute-type-string "." attribute-name SP string-value-base64
string-value-base64 = string-base64 [13]
string-base64 =/ "string:encoded:base64:" charset ":" string-base64-value
charset = 1*(ALPHA / DIGIT / "-")
string-base64-value = (1*(4base64-char) [base64-pad]) / base64-pad
base64-pad = (2base64-char "==") / (3base64-char "=")
base64-char = ALPHA / DIGIT / "+" / "/"
```

NOTE: The above specification is not meant to specify the base64 encoding.

7.2.3 Policies

7.2.3.1 General definition of a policy and syntax

A policy is a logical statement. A logical statement can itself be made of:

- a relational statement, or
- a logical operation between two logical statements, or
- a logical threshold gate between two or more logical statements.

A policy shall comply with the following ABNF [13] definition:

```
policy = policy-id SP policy-version SP logical-statement
policy-id = 1*(ALPHA / DIGIT) *(("-" / ".") 1*(ALPHA / DIGIT))
policy-version = 1*(DIGIT) *(("-" / ".") 1*(DIGIT))
logical-statement = relational-statement / "(" logical-statement log-op logical-statement ")"
logical-statement =/ threshold-gate
```

7.2.3.2 Relational statements

7.2.3.2.1 Introduction

A relational statement is a comparison between an attribute and a constant by means of a relational operator, in accordance with the type of the attribute.

7.2.3.2.2 Relational operators for the unsigned integer attribute type

The following relational operators shall be supported, where A shall be of type UINT(k), k a strictly positive integer, and b shall be a constant of the same type:

Table 7.1: relational operators for unsigned integers

Operator	Relational statement	Matches if
<	A < b	A is strictly inferior to b
<=	A <= b	A is inferior or equal to b
>	A > b	A is strictly superior to b
>=	A >= b	A is superior or equal to b
==	A == b	A is equal to b
!=	A != b	A is different from b

A relational statement for an unsigned integer attribute shall comply with the following ABNF [13] specification:

```
relational-statement = "(" attribute-name SP rel-op-int SP uint-constant ")"
rel-op-int = "<" / "<=" / ">" / ">=" / "==" / "!="
uint-constant = 1*DIGIT
```

7.2.3.2.3 Relational operators for the boolean attribute type

The following operators shall be supported, where A shall be of type BOOL:

Table 7.2: relational operators for booleans

Operator	Relational statement	Matches if
is_true	A is_true	A is set to the value 1
is_false	A is_false	A is set to the value 0

A relational statement for a boolean attribute shall comply with the following ABNF [13] specification:

```
relational-statement =/ "(" attribute-name SP rel-op-bool ")"
rel-op-bool = "is_true" / "is_false"
```

7.2.3.2.4 Relational operators for the string attribute type

The equality operator shall be supported, where A shall be of type STRING and b shall be a string constant:

Table 7.3: relational operators for strings

Operator	Relational statement	Matches if
eq	A eq b	The string represented by A is the same, character-by-character, as the string constant b

A relational statement for a string attribute shall comply with the following ABNF [13] specification:

```
relational-statement =/ "(" attribute-name SP rel-op-string SP string-constant ")"
rel-op-string = "eq"
string-constant = string-plain / string-base64
```

7.2.3.3 Logical operators

The following logical operators shall be supported, where X and Y shall be two logical statements:

Table 7.4: logical operators for logical statements

Operator	Policy statement	Matches if
AND	X AND Y	Both X and Y match
OR	X OR Y	Either X matches, Y matches, or both

A logical operator shall comply with the following ABNF specification:

```
log-op = "AND" / "OR"
```

7.2.3.4 Threshold gates

A threshold gate is a logical operator that operates over a set of N logical statements, where N is a strictly positive integer, and matches if at least K of these logical statement match, where $0 < K <= N$.

A threshold gate shall comply with the following ABNF [13] specification:

```
threshold-gate = threshold "_OF(" logical-statement 1*(comma logical-statement) ")"
threshold = %x31-39 *DIGIT
```

Implementations shall reject threshold gates where the value of the element `threshold` is strictly superior to the number of enclosed logical statements in the related operator declaration.

7.2.3.5 Top-level statements

A relational statement, logical statement, or threshold gate shall be referred to as a top-level statement when it is evaluated for all branches of the policy, i.e. when it is always evaluated.

7.2.4 ABKEM bindings

7.2.4.1 Introduction

This clause specifies the translation rules between Layer 1 access control and the underlying ABKEM implementation.

7.2.4.2 Binding rules for value assignment to attributes in annotation

7.2.4.2.1 Common translation rules

An ABKEM attribute is an anonymous string. Any attribute defined at Layer 1 shall be translated into an ABKEM attribute set of cardinality `max-occurence` as defined in the related Layer 1 declaration (refer to clause 7.2.2.1). ABKEM attributes related to the same Layer 1 attribute shall have their identifier defined according to the following ABNF [13] specification:

```
attribute-id = attribute-name "." id
id = %x31-39 *DIGIT
```

When the ABKEM in use does not support attribute repetition in a policy:

1) ABKEM attributes shall not be instantiated more than once in a policy. KP-ABKEM implementations shall reject generation of a secret key bearing more than one instance of any attribute. CP-ABKEM implementations shall reject generation of ciphertexts bearing more than one instance of any attribute.

2) The mechanism defining `attribute-id` with values of `id` from 1 to `max-occurrence` shall be used.

When the ABKEM in use supports attribute repetition in a policy:

- The mechanism defining `attribute-id` with values of `id` from 1 to `max-occurrence` shall be used, but `max-occurrence` shall always be set to 0 (ignoring the value provided in the Layer 1 declaration for the attribute).

7.2.4.2.2 Unsigned integer

In addition to being subject of the rules in clause 7.2.4.2.1, unsigned integer attributes are decomposed into a set of ABKEM attributes with each ABKEM attribute representing a bit in the base 2 representation of the Layer 1 attribute value. Unsigned integer attributes shall be translated according to the algorithm below.

Implementations shall reject unsigned integer attributes where the `attribute-type-uint` element declares more than one `parameter` element.

NOTE: Unsigned integer attributes where the `attribute-type-uint` element declares more than one `parameter` elements are allowed at Layer 2 and are first translated to Layer 1 attributes.

Convert the value of the `uint-value` element defined at Layer 1 for the attribute (refer to clause 7.2.2.4) into its base 2 representation, where bit position 0 shall be that of the least significant bit, and bit position m shall be that of the most significant bit.

Fetch the value of the `max-occurence` element that corresponds to the declaration of the attribute.

Fetch the value of the parameter k that corresponds to the declared length of the attribute in bits. Raise a compilation error if k is strictly inferior to m.

For each value of the element `id` between 1 and `max-occurence`.

For each position `bit-position` between 0 and m, annotate with an attribute according to the following ABNF [13] specification, where the `bit-value` element shall be set to the bit value of the corresponding `bit-position` position in the base 2 representation of the attribute value at Layer 1:

```
abkem-encoding-uint = attribute-type-uint "." attribute-id "." bit-position "." bit-value
bit-position = 1*DIGIT
bit-value = "0" / "1"
```

If k is strictly superior to m, for each position `bit-position` between k and m +1, annotate with an attribute according to the above ABNF [13] specification, where the `bit-value` element shall be set to the value 0.

7.2.4.2.3 Boolean

Boolean attributes shall be translated according to the algorithm below.

Fetch the value of the `max-occurence` element that corresponds to the declaration of the attribute.

For each value of the element `id` between 1 and `max-occurence`, annotate with an attribute according to the following ABNF [13] specification, where the value shall be assigned according to the semantic of the `bool-value` element:

```
abkem-encoding-bool = attribute-type-bool "." attribute-id "." bool-value
```

7.2.4.2.4 String

String attributes with values assigned using the plain string method shall be translated according to the algorithm below.

Fetch the value of the `max-occurence` element that corresponds to the declaration of the attribute.

For each value of the element `id` between 1 and `max-occurence`, annotate with an attribute according to the following ABNF [13] specification, where the value shall be assigned according to the semantic of the `string-plain-value` element:

```
abkem-encoding-string-plain = attribute-type-string "." attribute-id "." string-plain
```

String attributes with values assigned using the encoded string method shall be translated according to the algorithm below.

Fetch the value of the `max-occurence` element that corresponds to the declaration of the attribute.

For each value of the element `id` between 1 and `max-occurence`, annotate with an attribute according to the following ABNF [13] specification, where the value shall be assigned according to the semantic of the `string-base64-value` element:

```
abkem-encoding-string-base64 = attribute-type-string "." attribute-id "." string-base64
```

7.2.4.3 Binding rules for policy translation

7.2.4.3.1 Common translation rules

When the ABKEM in use does not support attribute repetition in a policy, implementations translating Layer 1 policies into ABKEM policies shall maintain an occurrence counter for each Layer 1 attribute in the context of a given policy. At the first occurrence of a Layer 1 attribute in said Layer 1 policy, the related occurrence counter shall be set to 1, and shall be incremented on each subsequent occurrence of the attribute in the same policy. These occurrence counters are used in the subsequent clauses.

When the ABKEM in use supports attribute repetition in a policy, the occurrence counter shall always be set to 1.

7.2.4.3.2 Integer

7.2.4.3.2.1 "<" operator

A relational statement conformant to the following ABNF [13] specification:

```
"(" attribute-name SP "<" SP uint-constant ")"
```

where the value of the `uint-constant` element is strictly positive, shall first be translated into a statement in the form:

```
"(" attribute-name SP "<=" SP uint-constant ")"
```

where the value of the `uint-constant` element shall be decreased by one. Translation shall continue according to clause 7.2.4.3.2.5.

7.2.4.3.2.2 ">" operator

A relational statement conformant to the following ABNF specification:

```
"(" attribute-name SP ">" SP uint-constant ")"
```

where the value of the `uint-constant` element is strictly inferior to the maximum integer value that can be expressed over k bits (refer to clause 7.2.2.2), shall first be translated into a statement in the form

```
"(" attribute-name SP ">=" SP uint-constant ")"
```

where the value of the `uint-constant` element shall be increased by one. Translation shall continue according to clause 7.2.4.3.2.6.

7.2.4.3.2.3 "==" operator

A Layer 1 relational statement conformant to the following ABNF specification:

```
"(" attribute-name SP "==" SP uint-constant ")"
```

shall be translated according to the algorithm below.

Let *count* be the current occurrence counter value of the Layer 1 attribute of type unsigned integer.

Convert the value of the `uint-constant` element into its base 2 representation, where bit position 0 shall be that of the least significant bit, and bit position m shall be that of the most significant bit. Verify that the length of the `uint-constant` element in bits is at most k (that is, m is inferior or equal to k), raise a compilation error otherwise.

Initialize the ABKEM relational statement with the string `"("`.

If m is strictly inferior to k, for each position `bit-position` from k to m + 1, append the ABKEM relational statement with a string conformant to the following ABNF [13] specification:

```
attribute-type-uint "." attribute-id "." bit-position "." bit-value SP "AND" SP
```

where the `bit-value` element shall be set to 0, and the value of *count* shall be used for the `id` element of `attribute-id`.

For position `bit-position` m, append the ABKEM relational statement with a string conformant to the following ABNF [13] specification:

```
attribute-type-uint "." attribute-id "." bit-position "." bit-value
```

where the `bit-value` element shall be set to the bit value of position m in the base 2 representation of the `uint-constant` element, and the value of *count* shall be used for the `id` element of `attribute-id`.

For each position `bit-position` from m - 1 to 0, append the ABKEM relational statement with a string conformant to the following ABNF specification:

```
SP "AND" SP attribute-type-uint "." attribute-id "." bit-position "." bit-value
```

where the `bit-value` element shall be set to the bit value of the corresponding `bit-position` position in the base 2 representation of the `uint-constant` element, and the value of *count* shall be used for the `id` element of `attribute-id`.

End the ABKEM relational statement with the string `")"`.

7.2.4.3.2.4 "!=" operator

A Layer 1 relational statement conformant to the following ABNF specification:

```
"(" attribute-name SP "!=" SP uint-constant ")"
```

shall be translated according to the algorithm below.

Let *count* be the current occurrence counter value of the Layer 1 attribute of type unsigned integer.

Convert the value of the `uint-constant` element into its base 2 representation, where bit position 0 shall be that of the least significant bit, and bit position k shall be that of the most significant bit. Verify that the length of the `uint-constant` element in bits is at most k (that is, m is inferior or equal to k), raise a compilation error otherwise.

Initialize the ABKEM relational statement with the string `"("`.

If m is strictly inferior to k, for each position `bit-position` from k to m + 1, append the ABKEM relational statement with a string conformant to the following ABNF [13] specification:

```
attribute-type-uint "." attribute-id "." bit-position "." bit-value SP "OR" SP
```

where the `bit-value` element shall be set to 1, and the value of *count* shall be used for the `id` element of `attribute-id`.

For position `bit-position` m, append the ABKEM relational statement with a string conformant to the following ABNF [13] specification:

```
attribute-type-uint "." attribute-id "." bit-position "." bit-value
```

where the `bit-value` element shall be set to the complement of the bit value of position m in the base 2 representation of the `uint-constant` element, and the value of *count* shall be used for the `id` element of `attribute-id`.

For each position `bit-position` from m - 1 to 0, append the ABKEM relational statement with a string conformant to the following ABNF [13] specification:

```
SP "OR" SP attribute-type-uint "." attribute-id "." bit-position "." bit-value
```

where the `bit-value` element shall be set to the complement of the bit value of the corresponding `bit-position` position in the base 2 representation of the `uint-constant` element, and the value of *count* shall be used for the `id` element of `attribute-id`.

End the ABKEM relational statement with the string `")"`.

7.2.4.3.2.5 "<=" operator

A Layer 1 relational statement conformant to the following ABNF specification:

```
"(" attribute-name SP "<=" SP uint-constant ")"
```

shall be translated according to the algorithm below.

Let *count* be the current occurrence counter value of the Layer 1 attribute of type unsigned integer.

Convert the value of the `uint-constant` element into its base 2 representation, where bit position 0 shall be that of the least significant bit, and bit position m shall be that of the most significant bit. Verify that the length of the `uint-constant` element in bits is at most k (that is, m is inferior or equal to k), raise a compilation error otherwise.

Initialize the ABKEM relational statement with the string `"("`.

If m is strictly inferior to k, for each position `bit-position` from k to m + 1, append the ABKEM relational statement with a string conformant to the following ABNF [13] specification:

```
attribute-type-uint "." attribute-id "." bit-position "." bit-value SP "AND" SP
```

where the `bit-value` element shall be set to 0, and the value of *count* shall be used for the `id` element of `attribute-id`.

For each position `bit-position` from m to 1, if the bit value of the corresponding `bit-position` position in the base 2 representation of the `uint-constant` element is 0, append the ABKEM relational statement with a string conformant to the following ABNF [13] specification:

```
attribute-type-uint "." attribute-id "." bit-position "." bit-value SP "AND ("
```

where the `bit-value` element shall be set to 0, and the value of *count* shall be used for the `id` element of `attribute-id`.

else (if the bit value of the corresponding `bit-position` position in the base 2 representation of the `uint-constant` element is 1), append the ABKEM relational statement with a string conformant to the following ABNF [13] specification:

```
attribute-type-uint "." attribute-id "." bit-position "." bit-value SP "OR ("
```

where the `bit-value` element shall be set to 0, and the value of *count* shall be used for the `id` element of `attribute-id`.

For position `bit-position` 0, if the corresponding bit value in the base 2 representation of the `uint-constant` element is 0, append the ABKEM relational statement with a string conformant to the following ABNF [13] specification:

```
attribute-type-uint "." attribute-id "." bit-position "." bit-value
```

where the `bit-value` element shall be set to 0, and the value of *count* shall be used for the `id` element of `attribute-id`.

else (if the corresponding bit value in the base 2 representation of the `uint-constant` element is 1), append the ABKEM relational statement with a string conformant to the following ABNF specification:

```
attribute-type-uint "." attribute-id "." bit-position "." %x30 SP "OR" SP attribute-type-uint "."
attribute-id "." bit-position "." %x31
```

where the value of *count* shall be used for the `id` element of `attribute-id`.

For each position `bit-position` from m to 1, append the ABKEM relational statement with a `")"` string.

End the ABKEM relational statement with the string `")"`.

7.2.4.3.2.6 ">=" operator

A Layer 1 relational statement conformant to the following ABNF specification:

```
"(" attribute-name SP ">=" SP uint-constant ")"
```

shall be translated according to the algorithm below.

Let *count* be the current occurrence counter value of the Layer 1 attribute of type unsigned integer.

Convert the value of the `uint-constant` element into its base 2 representation, where bit position 0 shall be that of the least significant bit, and bit position m shall be that of the most significant bit. Verify that the length of the `uint-constant` element in bits is at most k (that is, m is inferior or equal to k), raise a compilation error otherwise.

Initialize the ABKEM relational statement with the string `"("`.

If m is strictly inferior to k, for each position `bit-position` from k to m + 1, append the ABKEM relational statement with a string conformant to the following ABNF [13] specification:

```
attribute-type-uint "." attribute-id "." bit-position "." bit-value SP "OR" SP
```

where the `bit-value` element shall be set to 1, and the value of *count* shall be used for the `id` element of `attribute-id`.

For each position `bit-position` from m to 1, if the bit value of the corresponding `bit-position` position in the base 2 representation of the `uint-constant` element is 0, append the ABKEM relational statement with a string conformant to the following ABNF [13] specification:

```
attribute-type-uint "." attribute-id "." bit-position "." bit-value SP "OR ("
```

where the `bit-value` element shall be set to 1, and the value of *count* shall be used for the `id` element of `attribute-id`.

else (if the bit value of the corresponding `bit-position` position in the base 2 representation of the `uint-constant` element is 1), append the ABKEM relational statement with a string conformant to the following ABNF [13] specification:

```
attribute-type-uint "." attribute-id "." bit-position "." bit-value SP "AND ("
```

where the `bit-value` element shall be set to 1, and the value of *count* shall be used for the `id` element of `attribute-id`.

For position `bit-position` 0, if the corresponding bit value in the base 2 representation of the `uint-constant` element is 0, append the ABKEM relational statement with a string conformant to the following ABNF [13] specification:

```
attribute-type-uint "." attribute-id "." bit-position "." %x30 SP "OR" SP attribute-type-uint "."
attribute-id "." bit-position "." %x31
```

where the value of *count* shall be used for the `id` element of `attribute-id`.

else (if the corresponding bit value in the the base 2 representation of the `uint-constant` element is 1), append the ABKEM relational statement with a string conformant to the following ABNF [13] specification:

```
attribute-type-uint "." attribute-id "." bit-position "." %x31
```

where the value of *count* shall be used for the `id` element of `attribute-id`.

For each position `bit-position` from m to 1, append the ABKEM relational statement with a `")"` string.

End the ABKEM relational statement with the string `")"`.

7.2.4.3.3 Boolean

Let *count* be the current occurrence counter value of the Layer 1 attribute of type boolean. A Layer 1 relational statement conformant to the following ABNF specification:

```
layerone-rl = "(" attribute-name SP "is_true" ")"
```

shall be translated into the ABKEM relational statement:

```
abkem-rl = "(" attribute-type-bool "." attribute-id "." %x31 ")"
```

using the value of *count* for the `id` element of `attribute-id`.

A Layer 1 relational statement conformant to the following ABNF specification:

```
layerone-rl = "(" attribute-name SP "is_false" ")"
```

shall be translated into the ABKEM relational statement:

```
abkem-rl = "(" attribute-type-bool "." attribute-id "." %x30 ")"
```

using the value of *count* for the `id` element of `attribute-id`.

7.2.4.3.4 String

Let *count* be the current occurrence counter value of the Layer 1 attribute of type string. A Layer 1 relational statement conformant to the following ABNF specification:

```
layerone-rl = "(" attribute-name SP "eq" SP string-constant ")"
```

shall be translated into the ABKEM relational statement:

```
abkem-rl = "(" attribute-type-string "." attribute-id "." string-constant ")"
```

using the value of *count* for the `id` element of `attribute-id`.

7.3 Intermediate access control layer ("Layer 2")

7.3.1 Introduction (informative)

The present clause describes translation rules allowing to build advanced attributes and policies on top of the access control capabilities supported by a compliant ABKEM scheme, that are specified in clause 7.2.

7.3.2 Additional attribute types

7.3.2.1 Double

7.3.2.1.1 Definition

The attribute type double shall be used to handle the float and double types as defined in ANSI/IEEE 754™-1985 [14]. It shall be defined as DOUBLE(k,l), where k and l shall both be strictly positive integers. The declaration shall follow the following ABNF specification:

```
double-decl = "define" SP double-type "." (attribute-name) "." max-occurence [SP source-datatype]
CRLF
double-type = "DOUBLE(" uint-value "," uint-value ")"
```

Attributes of type double shall be represented using the decimal representation. Using the definition DOUBLE(k,l), k shall be the number of bits assigned to the integer part, l shall be the number of bits assigned to the fractional part.

An attribute of type double shall be declared at Layer 1 using two attributes of type unsigned integer as defined in clause 7.2.2.1. Let an attribute of type double, defined by DOUBLE(k,l) be named *`attr`*. The integer part of *`attr`* at Layer 1 shall be named *`attr-ipart`* and shall be of Layer 1 type UINT(k), the fractional part of *`attr`* shall be named *`attr-fpart`* and shall be of Layer 1 type UINT(l). The names *`attr-ipart`* and *`attr-fpart`* shall be used as values for the `attribute-name` element in the Layer 1 attribute declaration.

7.3.2.1.2 Relational operators for doubles

Let *`attr`* be of type double and `const` be a constant of the same type.

When the operator `op` is <, >, <=, >=, or `!=`, the relational statement `(attr op const)` at Layer 2 shall be translated into a Layer 1 relational statement as follow:

```
(
(attr-ipart op const-ipart) OR
(
(attr-ipart == const-ipart) AND
(attr-fpart op const-fpart)
)
)
```

When the operator op is ==, the relational statement `(attr op const)` at Layer 2 shall be translated into a Layer 1 relational statement as follow:

```
((attr-ipart op const-ipart) AND (attr-fpart op const-fpart))
```

7.3.2.2 Time measurement

7.3.2.2.1 Timestamp

7.3.2.2.1.1 Definition

The attribute type timestamp allows the representation of elapsed time from an origin. It shall be defined as TIMESTAMP(k, U, C), where k shall be a strictly positive integer, U and C shall be optional string labels. If C is specified, then U shall be specified. Thus, the following constructs shall be interpreted as valid:

- TIMESTAMP(k),
- TIMESTAMP(k,U), and
- TIMESTAMP(k,U,C).

The declaration shall follow the following ABNF [13] specification:

```
timestamp-decl = "define" SP timestamp-type "." (attribute-name) "." max-occurence [SP source-
datatype] CRLF
timestamp-type = "TIMESTAMP(" uint-value ")"
timestamp-type =/ "TIMESTAMP(" uint-value "," string-plain ")"
timestamp-type =/ "TIMESTAMP(" uint-value "," string-plain "," string-plain ")"
```

Parameter k shall represent the number of bits over which the timestamp is encoded. Label U shall indicate the time unit in which the timestamp is expressed. The default value shall be "second". Valid values for label U in the present document shall be as detailed in table 7.5.

Table 7.5: supported units for label U of TIMESTAMP(k,U,C)

Value	Semantic
"millisecond"	The timestamp is expressed in milliseconds
No label, or "second" (default value)	The timestamp is expressed in seconds
"minute"	The timestamp is expressed in minutes
"hour"	The timestamp is expressed in hours
"day"	The timestamp is expressed in days
"week"	The timestamp is expressed in weeks
"month"	The timestamp is expressed in months
"year"	The timestamp is expressed in years

NOTE: When usage permits, coarser units allow to reduce the value of parameter k.

Label C shall indicate the context (reference system) within which the timestamp is expressed. The default value shall be "Epoch". Valid values for label C shall be as specified in table 7.6.

Table 7.6: supported reference points for label R of TIMESTAMP(k,U,C)

Value	Semantic
No label, or "Epoch" (default value)	The timestamp is expressed as a value since the Epoch.

When label C is not specified, or specified to be "Epoch", an attribute of type timestamp shall hold values expressed in Coordinated Universal Time (UTC), as a number of seconds since the Epoch (defined as 00:00:00, Thursday, January 1 1970) and shall account for leap seconds.

An attribute of type timestamp shall be declared at Layer 1 using an attribute of type unsigned integer as defined in clause 7.2.2.1. Let an attribute of type timestamp, defined by TIMESTAMP(k), be named *attr*. It shall be instantiated at Layer 1 by an attribute *attr* of type UINT(k), where *attr* shall be the value of the `attribute-name` element.

7.3.2.2.1.2 Relational operators

Relational operators for timestamps shall be as defined in clause 7.2.3.2.2.

7.3.2.2.2 Duration

7.3.2.2.2.1 Definition

The attribute type duration in the present clause shall be used to handle the type duration defined in ISO/IEC 8601 [11], over a single measurement unit. It shall be defined as DURATION(k,U), where k shall be a strictly positive integer and U shall be an optional string label. The following constructs shall be interpreted as valid:

- DURATION(k), and
- DURATION(k,U).

The declaration shall follow the following ABNF [13] specification:

```
duration-decl = "define" SP duration-type "." (attribute-name) "." max-occurence [SP source-
datatype] CRLF
duration-type = "DURATION(" uint-value ")"
duration-type =/ "DURATION(" uint-value "," string-plain ")"
```

Parameter k shall represent the number of bits over which the duration is encoded. Label U shall indicate the time unit in which the duration is expressed. The default values shall be "second". Valid values for label U in the present document shall be as detailed in table 7.7.

Table 7.7: supported units for label U of DURATION(k,U)

Value	Semantic
"millisecond"	The duration is expressed in milliseconds
No label, or "second" (default value)	The duration is expressed in seconds
"minute"	The duration is expressed in minutes
"hour"	The duration is expressed in hours
"day"	The duration is expressed in days
"week"	The duration is expressed in weeks
"month"	The duration is expressed in months
"year"	The duration is expressed in years

An attribute of type duration shall be instantiated at Layer 1 using an attribute of type unsigned integer as defined in clause 7.2.2.1. Let an attribute of type duration, defined by DURATION(k,U), be named *`attr`*. It shall be instantiated at Layer 1 by an attribute *`attr`* of type UINT(k), where *attr* shall be the value of the `attribute-name` element.t

Due to the complexity and performance overhead of representing a duration of time in more than one dimension (as is possible in ISO/IEC 8601 [11]), the present document limits support to one dimension (i.e. a single time unit). A time duration expressed in more than one dimension should be converted to a value expressed in the dimension of highest granularity present in the original value, taking into account the caveats indicated in xmlschema-2 [].

7.3.2.2.2.2 Relational operators

Relational operators for durations shall be as defined in clause 7.2.3.2.2.

7.3.2.2.3 Cycles

7.3.2.2.3.1 Definition

The attribute type cycles shall be defined as CYCLES(k, C), where k shall be a strictly positive integer and C shall be a string label.

The declaration shall follow the following ABNF [13] specification:

```
cycles-decl = "define" SP cycles-type "." (attribute-name) "." max-occurence [SP source-datatype]
CRLF
cycles-type =/ "CYCLES(" uint-value "," string-plain ")"
```

An attributes of type cycles shall hold values representing the number of elapsed cycles starting from a reference point.

Parameter k shall represent the number of bits over which the number of elapsed cycles is encoded. Label C references a context, which is the selection of a reference point and of the method used to issue cycles. How implementations configure and agree on the semantics of C is left out of scope of the present document.

An attribute of type cycles shall be instantiated at Layer 1 using an attribute of type integer as defined in clause 7.2.2.1. Let an attribute of type cycles, defined by CYCLES(k,C), be named *`attr`*. It shall be instantiated at Layer 1 by an attribute *`attr`* of type UINT(k), where *`attr`* shall be the value of the `attribute-name` element.

7.3.2.2.3.2 Relational operators

Relational operators for cycles shall be as defined in clause 7.2.3.2.2.

7.3.2.3 Location

7.3.2.3.1 Zone

7.3.2.3.1.1 Definition

The attribute type zone allows the representation of a location with a pre-configured vocabulary (e.g, countries, regions, districts, building names, floor numbers, room numbers, area numbers, and so forth). It shall be defined as ZONE(n), where n shall be a strictly positive integer.

An attribute of type zone shall be defined at Layer 2 using a list of allowed zone names (pre-configured strings), where the list shall be at most of size n. The declaration shall follow the following ABNF specification:

```
zone-decl = "define" SP zone-type "." (attribute-name) "." max-occurence SP zone-allowed-values [SP
source-datatype] CRLF
cycles-type =/ "ZONE(" uint-value ")"
zone-allowed-values = "allowed values (" string-allowed-value-component *("," string-allowed-value-
component) ")"

string-allowed-value-component = string-value-plain / string-value-base64
```

An attribute of type zone shall be instantiated at Layer 1 using an attribute of type unsigned integer as defined in clause 7.2.2.1. Let an attribute of type zone, defined by ZONE(n), be named *`attr`*. It shall be instantiated at Layer 1 by an attribute *`attr`* of type UINT(k), where k shall be the smallest number of bits allowing encoding of n values, and where *`attr`* shall be the value of the `attribute-name` element. References to zones defined at Layer 2 shall start with value 0 at Layer 1 and be incremented for each new string in the list. When the list size is less than the maximum value of *`attr`* at Layer 1, remaining values shall not be used.

7.3.2.3.1.2 Relational operators

Relational operators for zones shall be restricted to the equality and inequality operators as defined in clause 7.2.3.2.2.

7.3.2.3.2 Grid

7.3.2.3.2.1 Definition

The attribute type grid allows the representation of a location as an area made of the intersection of a column and a row. It shall be defined as GRID(n, m, C), where n and m shall be strictly positive integers, and C shall be a string label.

The declaration shall follow the following ABNF [13] specification:

```
grid-decl = "define" SP grid-type "." (attribute-name) "." max-occurence [SP source-datatype] CRLF
grid-type =/ "GRID(" uint-value "," uint-value "," string-plain ")"
```

Parameter n shall be interpreted as the maximum value of the abscissa, and parameter m shall by interpreted as the maximum value of the ordinate. The origin area shall be defined by the tuple (0,0).

Label C references a context, which allows implementations to determine how a location is to be expressed within the grid. How implementations configure and agree on the semantics of C is left out of scope of the present document.

An attribute of type grid shall be instantiated at Layer 1 using two attributes of type unsigned integer, as defined in clause 7.2.2.1. Let an attribute of type grid, defined by GRID(n,m) be named *attr*. The column (abscissa) part of attr at Layer 1 shall be named *attr-col* and shall be of type UINT(k), where k shall be the smallest number of bits allowing encoding of n values, and *attr-col* shall be the value of the `attribute-name` element. The row (ordinate) part of *attr* at Layer 1 shall be named *attr-row* and shall be of type UINT(l), where l shall be the smallest number of bits allowing encoding of m values, and *attr-row* shall be the value of the `attribute-name` element.

7.3.2.3.2.2 Relational operators

Relational operators for grids shall be restricted to the equality and inequality operators.

Let `attr` be of type grid and `const` be a constant of the same type representing an intersection in the referenced grid.

When the operator op is `==`, the relational statement `(attr op const)` at Layer 2 shall be translated into a Layer 1 relational statement as follow:

```
((attr-col op const-col) AND (attr-row op const-row)
```

When the operator op is `!=`, the relational statement `(attr op const)` at Layer 2 shall be translated into a Layer 1 relational statement as follow:

```
((attr-col op const-col) OR (attr-row op const-row)
```

7.3.2.3.3 1d point

7.3.2.3.3.1 Definition

The attribute type 1d point allows the representation of a location on a line as a distance from an origin (e.g. altitude, depth). It shall be defined as 1D-POINT(k, U, C), where k shall be a strictly positive integer, U and C shall be mandatory string labels.

The declaration shall follow the following ABNF [13] specification:

```
d1-point-decl = "define" SP d1-point-type "." (attribute-name) "." max-occurence [SP source-
datatype] CRLF
d1-point-type =/ "1D-POINT(" uint-value "," string-plain "," string-plain ")"
```

Parameter k shall represent the number of bits over which the distance to origin is encoded. The origin shall be at distance 0. Label U shall indicate the unit in which the distance is expressed. Valid values for label U in the present document shall be as detailed in table 7.8.

Table 7.8: supported units for label U of 1D-POINT(k,U,C)

Value	Semantic
unit	a unit defined by the context label C
centimetre	centimetre in the SI [1]
decimetre	decimetre in the SI [1]
metre	metre in the SI [1]
kilometre	kilometre in the SI [1]

NOTE: When usage permits, coarser units allow to reduce the value of parameter k.

Label C shall indicate the context (reference system) allowing implementations to learn about the origin and calculate distances. Definition of a context is out of scope of the present document.

An attribute of type 1d point shall be instantiated at Layer 1 using an attribute of type unsigned integer as defined in clause 7.2.2.1. Let an attribute of type 1d point, defined by 1D-POINT(k,U,C), be named `attr`. It shall be instantiated at Layer 1 by an attribute `attr` of type UINT(k), where `attr` shall be the value of the `attribute-name` element.

7.3.2.3.3.2 Relational operators

Relational operators for 1d points shall be as defined in clause 7.2.3.2.2.

7.3.2.3.4 2d point

7.3.2.3.4.1 Definition

The attribute type 2d point allows the representation of a location in a two-dimensional cartesian system. It shall be defined as 2D-POINT(k,l,U,C), where k and l shall be strictly positive integers, U and C shall be mandatory string labels.

The declaration shall follow the following ABNF [13] specification:

```
d2-point-decl = "define" SP d2-point-type "." (attribute-name) "." max-occurence [SP source-
datatype] CRLF
d2-point-type =/ "2D-POINT(" uint-value "," uint-value "," string-plain "," string-plain ")"
```

Parameter k shall represent the number of bits over which the distance to origin on the x-axis (abscissa) is encoded. Parameter l shall represent the number of bits over which the distance to origin on the y-axis (ordinate) is encoded. The origin shall be defined by the tuple (0,0). Label U shall indicate the unit in which the distance is expressed. Valid values for label U in the present document shall be as detailed in table 7.9.

Table 7.9: supported units for label U of 2D-POINT(k,l,U,C)

Value	Semantic
unit	a unit defined by the context label C
centimetre	centimetre in the SI [1]
decimetre	decimetre in the SI [1]
metre	metre in the SI [1]
kilometre	kilometre in the SI [1]

NOTE: When usage permits, coarser units allow to reduce the value of parameters k and l.

Label C shall indicate the context (reference system) allowing implementations to learn about the origin and calculate distances on the axes. Definition of a context is out of scope of the present document.

An attribute of type 2d point shall be instantiated at Layer 1 using two attributes of type unsigned integer as defined in clause 7.2.2.1. Let an attribute of type 2d point, defined by 2D-POINT(k,l,U,C), be named *`attr`*. It shall be instantiated at Layer 1 by an attribute *`attr-x`* of type UINT(k), and an attribute *`attr-y`* of type UINT(l), where *`attr-x`* and *`attr-y`* shall be the respective values of the `attribute-name` element.

7.3.2.3.4.2 Relational operators

Relational operators for 2d points shall be defined as `inside` and `outside`. The inside operator matches if the value of a target attribute is within a square zone defined by two constant points. The outside operator matches if the value of the target attribute is outside a square zone defined by two constant points.

Let `attr` be of type 2d point and `A`, `B`, be two constants of the same type with the conditions that `A-x` < `B-x` and `A-y` < `B-y`.

The relational statement `(attr inside A B)` at Layer 2 shall be translated into a Layer 1 relational statement as follow:

```
(
 (attr-x >= A-x) AND (attr-y >= A-y) AND
 (attr-x <= B-x) AND (attr-y <= B-y)
)
```

The relational statement `(attr outside A B)` at Layer 2 shall be translated into a Layer 1 relational statement as follow:

```
((attr-x <= A-x) OR (attr-y <= A-y) OR (attr-x >= B-x) OR (attr-y >= B-y))
```

7.3.2.3.5 3d point

7.3.2.3.5.1 Definition

The attribute type 3d point allows the representation of a location in a three-dimensional cartesian system. It shall be defined as 3D-POINT(k,l,m,U,C), where k, l, and m shall be strictly positive integers, U and C shall be mandatory string labels.

The declaration shall follow the following ABNF [13] specification:

```
3d-point-decl = "define" SP 3d-point-type "." (attribute-name) "." max-occurence [SP source-
datatype] CRLF
3d-point-type =/ "3D-POINT(" uint-value "," uint-value "," uint-value "," string-plain "," string-
plain ")"
```

Parameter k shall represent the number of bits over which the distance to origin on the x-axis (abscissa) is encoded. Parameter l shall represent the number of bits over which the distance to origin on the y-axis (ordinate) is encoded. Parameter m shall represent the number of bits over which the distance to origin on the z-axis is encoded. The origin area shall be defined by the tuple (0,0,0). Label U shall indicate the unit in which the distance is expressed. Valid values for label U in the present document shall be as detailed in table 7.10.

Table 7.10: supported units for label U of 3D-POINT(k,l,m,U,C)

Value	Semantic
unit	a unit defined by the context label C
centimetre	centimetre in the SI [1]
decimetre	decimetre in the SI [1]
metre	metre in the SI [1]
kilometre	kilometre in the SI [1]

NOTE: When usage permits, coarser units allow to reduce the value of parameters k, l, and m.

Label C shall indicate the context (reference system) allowing implementations to learn about the origin and calculate distances on the axes. Definition of a context is out of scope of the present document.

An attribute of type 3d point shall be instantiated at Layer 1 using three attributes of type unsigned integer, as defined in clause 7.2.2.1. Let an attribute of type 3d point, defined by 2D-POINT(k,l,m,U,C), be named *attr*. It shall be instantiated at Layer 1 by an attribute *attr-x* of type UINT(k), an attribute *attr-y* of type UINT(l), and an attribute *attr-z* of type UINT(m), where *attr-x*, *attr-y* and *attr-z* shall be the respective values of the attribute-name element.

7.3.2.3.5.2 Relational operators

Relational operators for 3d points shall be defined as inside and outside. The inside operator matches if the value of a target attribute is within a cubic zone defined by two constant points. The outside operator matches if the value of the target attribute is outside a cubic zone defined by two constant points.

Let attr be of type 2d point and A, B, be two constants of the same type with the conditions that A and B represent two extremes of the cube diagonal and A is the closest point of the cube to the origin (0,0,0). By construction, A-x < B-x, A-y < B-y, and A-z < B-z.

The relational statement (attr inside A B) at Layer 2 shall be translated into a Layer 1 relational statement as follow:

```
(
 (attr-x >= A-x) AND (attr-y >= A-y) AND (attr-z >= A-z) AND
 (attr-x <= B-x) AND (attr-y <= B-y) AND (attr-z <= B-z)
)
```

The relational statement `(attr outside A B)` at Layer 2 shall be translated into a Layer 1 relational statement as follow:

```
(
 (attr-x <= A-x) OR (attr-y <= A-y) OR (attr-z <= A-z) OR
 (attr-x >= B-x) OR (attr-y >= B-y) OR (attr-z >= B-z)
)
```

7.3.2.3.6 Circle perimeter

7.3.2.3.6.1 Definition

The attribute type circle perimeter allows the representation of a location as a distance from a single point of reference within a plane (the circle centre). It shall be defined as CIRCLE(k, U, C), where k shall be a strictly positive integer, U and C shall be mandatory string labels.

The declaration shall follow the following ABNF [13] specification:

```
circle-decl = "define" SP circle-type "." (attribute-name) "." max-occurence [SP source-datatype]
CRLF
circle-type =/ "CIRCLE(" uint-value "," string-plain "," string-plain ")"
```

Parameter k shall represent the number of bits over which the distance is encoded. Label U shall indicate the unit in which the distance is expressed. Valid values for label U in the present document shall be as detailed in table 7.11.

Table 7.11: supported units for label U of CIRCLE(k,U,C)

Value	Semantic
unit	a unit defined by the context label C
centimetre	centimetre in the SI [1]
decimetre	decimetre in the SI [1]
metre	metre in the SI [1]
kilometre	kilometre in the SI [1]

NOTE: When usage permits, coarser units allow to reduce the value of parameter k.

Label C shall indicate the context (reference system) allowing implementations to learn about the circle centre and calculate distances. Definition of a context is out of scope of the present document.

An attribute of type circle perimeter shall be instantiated at Layer 1 using an attribute of type unsigned integer, as defined in clause 7.2.2.1. Let an attribute of type circle perimeter, defined by CIRCLE(k,U,C), be named *`attr`*. It shall be instantiated at Layer 1 by an attribute *`attr`* of type UINT(k), where *`attr`* shall be the value of the `attribute-name` element.

7.3.2.3.6.2 Relational operators

Relational operators for circle perimeters shall be as defined in clause 7.2.3.2.2.

7.3.2.3.7 Sphere surface

7.3.2.3.7.1 Definition

The attribute type sphere surface allows the representation of a location as a distance from a single point of reference within a volume (the sphere centre). It shall be defined as SPHERE(k, U, C), where k shall be a strictly positive integer, U and C shall be mandatory string labels.

The declaration shall follow the following ABNF [13] specification:

```
sphere-decl = "define" SP sphere-type "." (attribute-name) "." max-occurence [SP source-datatype]
CRLF
sphere-type =/ "SPHERE(" uint-value "," string-plain "," string-plain ")"
```

Parameter k shall represent the number of bits over which the distance is encoded. Label U shall indicate the unit in which the distance is expressed. Valid values for label U in the present document shall be as detailed in table 7.12.

Table 7.12: supported units for label U of SPHERE(k,U,C)

Value	Semantic
unit	a unit defined by the context label C
centimetre	centimetre in the SI [1]
decimetre	decimetre in the SI [1]
metre	metre in the SI [1]
kilometre	kilometre in the SI [1]

NOTE: When usage permits, coarser units allow to reduce the value of parameter k.

Label C shall indicate the context (reference system) allowing implementations to learn about the sphere centre and calculate distances. Definition of a context is out of scope of the present document.

An attribute of type sphere surface shall be instantiated at Layer 1 using an attribute of type unsigned integer as defined in clause 7.2.2.1. Let an attribute of type sphere surface, defined by SPHERE(k,U,C), named *`attr`*. It shall be instantiated at Layer 1 by an attribute *`attr`* of type UINT(k), where *`attr`* shall be the value of the `attribute-name` element.

7.3.2.3.7.2 Relational operators

Relational operators for sphere surfaces shall be as defined in clause 7.2.3.2.2.

7.3.2.4 Abstract string types

7.3.2.4.1 Free string

7.3.2.4.1.1 Definition

The attribute type free string allows the representation of a foreign data type, when such foreign data type can be translated into the base STRING type at Layer 1 (refer to clause 7.2). Implementations can keep track of the foreign data type thanks to the `source-datatype` component. It shall be defined at Layer 2 as FREESTRING. The declaration shall follow the following ABNF [13] specification:

```
freestring-decl = "define" SP freestring-type "." attribute-name "." max-occurence SP source-
datatype CRLF
```

Attributes of type free string shall be instantiated at Layer 1 using a string attribute type as defined in clause 7.2.2.1. Let an attribute of type freestring be name *`attr`*. It shall be instantiated at Layer 1 by an attribute *`attr`* of type STRING, where *`attr`* shall be the value of the `attribute-name` element. The `source-datatype` component shall be set to the value of the foreign datatype identifier (refer to clause 7.3.3 for examples of such use).

Implementations should use the `sep-namespace` separator of the `attribute-name` element as defined in clause 7.2.2.2 in order to group attributes of type free string into namespaces. Implementations shall support groupings independently of the value of the `source-datatype` component.

7.3.2.4.1.2 Relational operators

Relational operators for the free string attribute type shall be restricted to the equality operator as defined in clause 7.2.3.2.4.

7.3.2.4.2 Clearance

7.3.2.4.2.1 Definition

The attribute type clearance allows the representation of a clearance level. It shall be defined at Layer 2 as CLEARANCE and shall take no parameter, but may be assigned a list of allowed clearance levels (pre-configured strings). The declaration shall follow the following ABNF [13] specification:

```
clearance-decl = "define" SP clearance-type "." (attribute-name) "." max-occurence SP clearance-
allowed-values [SP source-datatype] CRLF
clearance-type = "CLEARANCE"
clearance-allowed-values = "allowed values (" string-allowed-value-component *("," string-allowed-
value-component) ")"
```

An attribute of type clearance shall be instantiated at Layer 1 using a string attribute type as defined in clause 7.2.2.1. Let an attribute of type clearance be named `attr`. It shall be instantiated at Layer 1 by an attribute `attr` of type STRING, where `attr` shall be the value of the `attribute-name` element.

Implementations should use the `sep-namespace` separator of the `attribute-name` element as defined in clause 7.2.2.2 in order to group attributes of type clearance into namespaces.

7.3.2.4.2.2 Relational operators

Relational operators for the clearance attribute type shall be restricted to the equality operator as defined in clause 7.2.3.2.4.

7.3.2.4.3 Role

7.3.2.4.3.1 Definition

The attribute type role allows the representation of a role. It shall be defined at Layer 2 as ROLE and shall take no parameter, but may be assigned a list of allowed role names (pre-configured strings). The declaration shall follow the following ABNF [13] specification:

```
role-decl = "define" SP role-type "." (attribute-name) "." max-occurence SP role-allowed-values [SP
source-datatype] CRLF
role-type = "ROLE"
role-allowed-values = "allowed values (" string-allowed-value-component *("," string-allowed-value-
component) ")"
```

An attribute of type role shall be instantiated at Layer 1 using a string attribute type as defined in clause 7.2.2.1. Let an attribute of type role be named `attr`. It shall be instantiated at Layer 1 by an attribute `attr` of type STRING, where `attr` shall be the value of the `attribute-name` element.

Implementations should use the `sep-namespace` separator of the `attribute-name` element as defined in clause 7.2.2.2 in order to group attributes of type role into namespaces.

7.3.2.4.3.2 Relational operators

Relational operators for the role attribute type shall be restricted to the equality operator as defined in clause 7.2.3.2.4.

7.3.2.4.4 User

7.3.2.4.4.1 Definition

The attribute type user allows the representation of a user. It shall be defined at Layer 2 as USER and shall take no parameter. The declaration shall follow the following ABNF [13] specification:

```
user-decl = "define" SP user-type "." (attribute-name) "." max-occurence [SP source-datatype] CRLF
user-type = "USER"
```

An attribute of type user shall be instantiated at Layer 1 using a string attribute type as defined in clause 7.2.2.1. Let an attribute of type user be named `attr`. It shall be instantiated at Layer 1 by an attribute `attr` of type STRING, where `attr` shall be the value of the `attribute-name` element.

Implementations should use the `sep-namespace` separator of the `attribute-name` element as defined in clause 7.2.2.2 in order to group attributes of type user into namespaces.

7.3.2.4.4.2 Relational operators

Relational operators for the user attribute type shall be restricted to the equality operator as defined in clause 7.2.3.2.4.

7.3.2.4.5 Device

7.3.2.4.5.1 Definition

The attribute type device allows the representation of a device. It shall be defined at Layer 2 as DEVICE and shall take no parameter. The declaration shall follow the following ABNF [13] specification:

```
device-decl = "define" SP device-type "." (attribute-name) "." max-occurence [SP source-datatype]
CRLF
device-type = "DEVICE"
```

An attribute of type device shall be instantiated at Layer 1 using a string attribute type as defined in clause 7.2.2.1. Let an attribute of type device be named *`attr`*. It shall be instantiated at Layer 1 by an attribute *`attr`* of type STRING, where *`attr`* shall be the value of the `attribute-name` element.

Implementations should use the `sep-namespace` separator of the `attribute-name` element as defined in clause 7.2.2.2 in order to group attributes of type device into namespaces.

7.3.2.4.5.2 Relational operators

Relational operators for the device attribute type shall be restricted to the equality operator as defined in clause 7.2.3.2.4.

7.3.2.4.6 Function

7.3.2.4.6.1 Definition

The attribute type function allows the representation of an executable function. It shall be defined at Layer 2 as FUNCTION and shall take no parameter. The declaration shall follow the following ABNF [13] specification:

```
function-decl = "define" SP function-type "." (attribute-name) "." max-occurence [SP source-
datatype] CRLF
function-type = "FUNCTION"
```

An attribute of type device shall be instantiated at Layer 1 using a string attribute type as defined in clause 7.2.2.1. Let an attribute of type device be named *`attr`*. It shall be instantiated at Layer 1 by an attribute *`attr`* of type STRING, where *`attr`* shall be the value of the `attribute-name` element.

Implementations should use the `sep-namespace` separator of the `attribute-name` element as defined in clause 7.2.2.2 in order to group attributes of type functions into namespaces.

7.3.2.4.6.2 Relational operators

Relational operators for the device attribute type shall be restricted to the equality operator as defined in clause 7.2.3.2.4.

7.3.2.4.7 Datatype

7.3.2.4.7.1 Definition

The attribute type datatype allows the representation of a data type. It shall be defined at Layer 2 as DATATYPE and shall take no parameter. The declaration shall follow the following ABNF [13] specification:

```
datatype-decl = "define" SP datatype-type "." (attribute-name) "." max-occurence [SP source-
datatype] CRLF
datatype-type = "DATATYPE"
```

An attribute of type datatype shall be instantiated at Layer 1 using a string attribute type as defined in clause 7.2.2.1. Let an attribute of type device be named *`attr`*. It shall be instantiated at Layer 1 by an attribute *`attr`* of type STRING, where *`attr`* shall be the value of the `attribute-name` element.

Implementations should use the `sep-namespace` separator defined in clause 7.2.2.2 in order to group attributes of type datatype into namespaces.

7.3.2.4.7.2 Relational operators

Relational operators for the datatype attribute type shall be restricted to the equality operator as defined in clause 7.2.3.2.4.

7.3.2.4.8 Origin

7.3.2.4.8.1 Definition

The attribute type origin allows the representation of the origin (source) of data. It shall be defined at Layer 2 as ORIGIN and shall take no parameter. The declaration shall follow the following ABNF [13] specification:

```
origin-decl = "define" SP origin-type "." (attribute-name) "." max-occurence [SP source-datatype]
CRLF
origin-type = "ORIGIN"
```

An attribute of type origin shall be instantiated at Layer 1 using a string attribute type as defined in clause 7.2.2.1. Let an attribute of type device be named *`attr`*. It shall be instantiated at Layer 1 by an attribute *`attr`* of type STRING, where *`attr`* shall be the value of the `attribute-name` element.

Implementations should use the `sep-namespace` separator defined in clause 7.2.2.2 in order to group attributes of type origin into namespaces.

7.3.2.4.8.2 Relational operators

Relational operators for the origin attribute type shall be restricted to the equality operator as defined in clause 7.2.3.2.4.

7.3.3 Support for foreign data types

7.3.3.1 Introduction

The present clause provides rules for supporting foreign data types using those defined at Layer 2 (clause 7.2) and Layer 3 (clause 7.3).

7.3.3.2 Datatypes identified in annex C

7.3.3.2.1 Primitive data types from XML Schema

The string data type from XML Schema [8] shall be translated into a Layer 1 STRING data type, the `source-datatype` component shall be set to "http://www.w3.org/2001/XMLSchema#string".

The boolean data type from XML Schema [8] shall be translated as a Layer 1 BOOL data type, the `source-datatype` component shall be set to "http://www.w3.org/2001/XMLSchema#boolean".

The integer data type from XML Schema [8] shall be translated as a Layer 1 UINT data type, the `source-datatype` component shall be set to "http://www.w3.org/2001/XMLSchema#integer". Only positive value of the integer data type instance shall be supported.

The double data type from XML Schema [8] shall be translated as a Layer 2 DOUBLE data type, the `source-datatype` component shall be set to "http://www.w3.org/2001/XMLSchema#double". Only positive value of the double data type instance shall be supported.

The hexBinary data type from XML Schema [8] shall be translated as a Layer 2 FREESTRING data type, the `source-datatype` component shall be set to "http://www.w3.org/2001/XMLSchema#hexBinary".

The base64Binary data type from XML Schema [8] shall be translated as a Layer 2 FREESTRING data type, the `source-datatype` component shall be set to "http://www.w3.org/2001/XMLSchema#base64Binary".

7.3.3.2.2 Time data types from XML Schema

The time data type from XML Schema [8] shall be translated into a Layer 2 TIMESTAMP data type, the `source-datatype` component shall be set to "http://www.w3.org/2001/XMLSchema#time".

The date data type from XML Schema [8] shall be translated into a Layer 2 TIMESTAMP data type, the `source-datatype` component shall be set to "http://www.w3.org/2001/XMLSchema#date".

The dateTime data type from XML Schema [8] shall be translated into a Layer 2 TIMESTAMP data type, the `source-datatype` component shall be set to "http://www.w3.org/2001/XMLSchema#dateTime".

The dayTimeDuration data type from XML Schema [8] shall be translated into a Layer 2 DURATION data type, the `source-datatype` component shall be set to "http://www.w3.org/2001/XMLSchema#dayTimeDuration".

The yearMonthDuration data type from XML Schema [8] shall be translated into a Layer 2 DURATION data type, the `source-datatype` component shall be set to "http://www.w3.org/2001/XMLSchema#yearMonthDuration".

7.3.3.2.3 Resource identifiers

The X.500 directory name data type shall be translated into a Layer 2 FREESTRING data type, the `source-datatype` component shall be set to "urn:oasis:names:tc:xacml:1.0:data-type:x500Name".

The IETF RFC 822 [i.29] name data type shall be translated into a Layer 2 FREESTRING data type, the `source-datatype` component shall be set to "urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name".

The IP address data type shall be translated into a Layer 2 FREESTRING data type, the `source-datatype` component shall be set to "urn:oasis:names:tc:xacml:2.0:data-type:ipAddress".

The DNS name data type shall be translated into a Layer 2 FREESTRING data type, the `source-datatype` component shall be set to "urn:oasis:names:tc:xacml:2.0:data-type:dnsName".

The XPath expression data type shall be translated into a Layer 2 FREESTRING data type, the `source-datatype` component shall be set to "urn:oasis:names:tc:xacml:3.0:data-type:xpathExpression".

The Any URI data type shall be translated into a Layer 2 FREESTRING data type, the `source-datatype` component shall be set to "http://www.w3.org/2001/XMLSchema#anyURI".

7.4 ABKEM operations

7.4.1 General

The present clause specifies methods to implement common access control use cases using Layer 1 (clause 7.2) and Layer 2 (clause 7.3) capabilities. Attribute and constant names are informative and given for consistency of the methods. Implementations shall select appropriate attribute and constant names in order to guarantee uniqueness.

7.4.2 Time-based implicit secret key revocation

7.4.2.1 Implementation in KP-ABKEM

Time-based implicit revocation relies on the ciphertext being annotated with a timestamp attribute with a value of relevance (e.g. the time at which the protected data was captured), and the secret keys being valid for a limited range of value of said attribute by means of a policy evaluating the attribute against boundary constants.

Let an attribute *`attr`* be of Layer 2 type timestamp as defined in clause 7.3.2.2.1. Let constants `validity_max` and `validity_min` be of the same type. `validity_min` shall represent a point in the past compared to `validity_max.`

The ABKEM ciphertext shall be annotated with an ABKEM attribute translated from attribute *`attr`* and holding a value representing a point in time, that shall be set by the encapsulating party.

In order to expire the ABKEM secret key starting from a time point `validity_max`, the ABKEM secret key shall contain a top-level relational statement translated from `"(attr < validity_max)"` in its access structure.

In order to bound the validity of an ABKEM secret key between two points in time `validity_min` and `validity_max`, the ABKEM secret key shall contain a top-level logical statement translated from `"((attr > validity_min) AND (attr < validity_max))"` in its access structure.

7.4.2.2 Implementation in CP-ABKEM

Time-based implicit revocation relies on the secret key being annotated with one or two timestamp attributes set to boundary constants, and the ciphertext containing a policy based on a comparison against a generated timestamp of relevance (e.g. the time at which the protected data was captured).

Let two attributes *`attr1`* and *`attr2`* be of Layer 2 type timestamp as defined in clause 7.3.2.2.1. Let constant `validity_max` , `validity_min`, and `time` be of the same type. `validity_min` shall represent a point in the past compared to `validity_max.`

In order to expire an ABKEM secret key starting from a point in time `validity_max`:

- The ABKEM secret key shall be annotated with an ABKEM attribute translated from the attribute *`attr1`* set to the value `validity_max`, and
- The ABKEM ciphertext shall contain a top-level relational statement translated from `"(attr1 > time)"` in its access structure, where the value `time` shall be set by the encapsulating party.

In order to bound the validity of an ABKEM secret key between two points in time `validity_min` and `validity_max`:

- The ABKEM secret key shall be annotated with an ABKEM attribute translated from the attribute *`attr1`* set to the value `validity_max`, and with an ABKEM attribute translated from the attribute *`attr2`* set to the value `validity_min`, and
- The ABKEM ciphertext shall contain a top-level logical statement translated from `"((attr1 > time) AND (attr2 < time))"` in its access structure, where the value `time` shall be set by the encapsulating party.

7.4.3 Counter-based implicit secret key revocation

7.4.3.1 Implementation in KP-ABKEM

The approach is similar to that described in clause 7.4.2.1, with the timestamp being replaced by an unsigned integer representing a strictly increasing counter.

Let an attribute *`attr`* be of Layer 1 type UINT(k) with k a strictly positive integer as defined in clause 7.2.2.2. Let constants `validity_max` and `validity_min` be of Layer 1 type UINT(k). `validity_min` shall hold a value less than `validity_max.`

The ABKEM ciphertext shall be annotated with an ABKEM attribute translated from the attribute *`attr`* holding a value that shall be set by the encapsulating party. The value of *`attr`* may remain constant across more than one ciphertext, and shall not decrease when a new ciphertext is annotated.

In order to expire an ABKEM secret key starting from a counter value `validity_max`, the ABKEM secret key shall contain a top-level relational statement translated from `"(attr < validity_max)"` in its access structure.

In order to bound the validity of an ABKEM secret key between counter values `validity_min` and `validity_max`, the ABKEM secret key shall contain a top-level logical statement in its access structure derived from the logical statement `"((attr > validity_min) AND (attr < validity_max))"`.

7.4.3.2 Implementation in CP-ABKEM

The approach is similar to that described in clause 7.4.2.2, with the timestamps being replaced by unsigned integers representing strictly increasing counters.

Let two attributes *`attr1`* and *`attr2`* be of Layer 1 type UINT(k) with k a strictly positive integer as defined in clause 7.2.2.2. Let constant `validity_max` , `validity_min`, and `count` be of Layer 1 type UINT(k). `validity_min` shall hold a value less than `validity_max.`

In order to expire an ABKEM secret key starting from counter value `validity_max`:

- The ABKEM secret key shall be annotated with an ABKEM attribute translated from the attribute *`attr1`* set to the value `validity_max`, and
- The ABKEM ciphertext shall contain a top-level relational statement translated from `"(attr1 > count)"` in its access structure, where the value `count` shall be set by the encapsulating party.

In order to bound the validity of an ABKEM secret key between counter values `validity_min` and `validity_max`:

- The ABKEM secret key shall be annotated with ABKEM attributes translated from the attribute *`attr1`* set to the value `validity_max`, and from the attribute *`attr2`* set to the value `validity_min`, and
- The ABKEM ciphertext shall contain a top-level logical statement translated from `"((attr1 > count) AND (attr2 < count))"` in its access structure where the value `time` shall be set by the encapsulating party.

7.4.4 Simple Mandatory access control

7.4.4.1 Implementation in KP-ABKEM

Mandatory access control relies on the ciphertext being annotated with an ABKEM string attribute representing a target clearance level, and the secret key being valid for said value by means of a policy that is always comparing the attribute against a string constant.

Let an attribute *`attr`* be of Layer 2 type clearance as defined in clause 7.3.2.4.1. Let constant `clearance` be of the same type.

The ABKEM ciphertext shall be annotated with an ABKEM attribute translated from attribute *`attr`* and holding a value representing the clearance level applicable to the plaintext, that shall be set by the encapsulating party.

The ABKEM secret key shall contain a top-level relational statement translated from `"(attr eq clearance)"` in its access structure, where the value `clearance` shall represent a clearance level assigned to the owner of the secret key.

7.4.4.2 Implementation in CP-ABKEM

Mandatory access control relies on the secret key being annotated with a string attribute representing an assigned clearance level, and the ciphertext being valid for said value by means of a policy that is always comparing the attribute against a string constant.

Let an attribute *`attr`* be of Layer 2 type clearance as defined in clause 7.3.2.4.1. Let constant `clearance` be of the same type.

The ABKEM secret key shall be annotated with an ABKEM attribute translated from attribute *`attr`* set to a value which shall represent a clearance level assigned to the owner of the secret key.

The ABKEM ciphertext shall contain a top-level relational statement translated from `"(attr eq clearance)"` in its access structure, where the constant `clearance` shall be set by the encapsulating party.

7.4.5 Role-based access control

7.4.5.1 Implementation in KP-ABKEM

Role-based access control relies on the ciphertext being annotated with an ABKEM string attribute representing a target role set by the encapsulating party, and the secret key being valid for said value by means of a relational statement that compares the attribute against a string constant. If the role is applicable to all branches of the policy in the secret key, then this shall be a top-level statement.

Let an attribute *`attr`* be of Layer 2 type role as defined in clause 7.3.2.4.2. Let constant `role` be of the same type.

The ABKEM ciphertext shall be annotated with an ABKEM attribute translated from attribute *`attr`* and holding a value representing the role applicable to the plaintext, that shall be set by the encapsulating party.

The ABKEM secret key shall contain a relational statement translated from `"(attr eq role)"` in its access structure, where `role` shall represent a role assigned to the owner of the secret key.

7.4.5.2 Implementation in CP-ABKEM

Role-based access control relies on the secret key being annotated with an ABKEM string attribute representing an assigned role, and the ciphertext being valid for said value by means of a relational statement that compares the attribute against a string constant. If the role is applicable to all branches of the policy in the ciphertext, then this shall be a top-level statement.

Let an attribute *`attr`* be of Layer 2 type role as defined in clause 7.3.2.4.1. Let constant `role` be of the same type.

The ABKEM secret key shall be annotated with an ABKEM attribute translated from attribute *`attr`* set to the value `role`, which shall represent a role assigned to the owner of the secret key.

The ABKEM ciphertext shall contain a relational statement translated from `"(attr eq role)"` in its access structure, where the constant `role` shall be set by the encapsulating party.

7.4.6 Location-based access control (informative)

Location-based access control can be implemented using the attribute types and related operators defined in clause 7.3.2.3 using methods similar to those of other ABKEM operations defined in clause 4.

Table 7.13 and table 7.14 provide implementation examples for KP-ABKEM and CP-ABKEM. A location restriction set as a top-level statement is applicable to an overall policy.

Table 7.13: implementations of location-based access control in KP-ABKEM

Operator used	Ciphertext annotations	Relational or logical statements in secret key
Zone (clause 7.3.2.3.1)	Attribute of type zone with value set by the encapsulating party.	Equality or inequality of the attribute to a constant set in the secret key.
Grid (clause 7.3.2.3.2)	Attribute of type grid with row and column values set by the encapsulating party.	Equality or inequality of the attribute to a constant set in the secret key.
1d point (clause 7.3.2.3.3)	Position encoded as a 1d point attribute and set by the encapsulating party. Unit and reference system negotiated out of band.	Comparison of the attribute to a distance constant set in the secret key.
2d point (clause 7.3.2.3.4)	Position encoded as a 2d point attribute and set by the encapsulating party. Unit and reference system negotiated out of band.	Verification that the attribute value is inside or outside a square zone defined by two constant points.
3d point (clause 7.3.2.3.5)	Position encoded as a 3d point attribute and set by the encapsulating party. Unit and reference system negotiated out of band.	Verification that the attribute value is inside or outside a cubic zone defined by two constant points.
Circle perimeter (clause 7.3.2.3.6)	Similar to 1d point attribute type.	
Sphere surface (clause 7.3.2.3.7)	Similar to 1d point attribute type.	

Table 7.14: implementations of location-based access control in CP-ABKEM

Operator used	Secret key annotations	Relational or logical statements in ciphertext
Zone (clause 7.3.2.3.1)	Attribute of type zone with value depending on access rights.	Equality or inequality of the attribute to a constant set by the encapsulating party.
Grid (clause 7.3.2.3.2)	Attribute of type grid with row and column values depending on access rights.	Equality or inequality of the attribute to a constant set by the encapsulating party.
1d point (clause 7.3.2.3.3)	Position range encoded as two 1d point attribute depending on access rigths. Unit and reference system negotiated out of band.	Comparison of the attributes to a distance constant set by the encapsulating party.
2d point (clause 7.3.2.3.4)	Position range encoded as two 2d point attribute depending on access rights. Unit and reference system negotiated out of band.	Verification that a position constant set by the encapsulating party is inside or outside a square zone defined by the attributes.
3d point (clause 7.3.2.3.5)	Position range encoded as two 3d point attribute depending on access rights. Unit and reference system negotiated out of band.	Verification that a position constant set by the encapsulating party is is inside or outside a cubic zone defined by the two attributes.
Circle perimeter (clause 7.3.2.3.6)	Similar to 1d point attribute type.	
Sphere surface (clause 7.3.2.3.7)	Similar to 1d point attribute type.	

7.4.7 Reduced access control based on the emergency level

7.4.7.1 Implementation in KP-ABKEM

Reduction of access control criteria in case of emergency relies on the ciphertext being annotated with an ABKEM string attribute representing an emergency level set by the encapsulating party, and the secret key holding one or more logical statements allowing access with reduced or no other criteria, when said attribute is set to a given constant. When a given emergency level and the associated reduced set of access control criteria are applicable to all branches of the policy in the secret key, then the logical statement holding the comparison to the emergency level shall be a top-level statement.

The emergency level may be detected by the device or registered to the device by external means.

Let an attribute *`emergency`* be of type string as defined in clause 7.2.2.2. Let constant `emergency_level` be of the same type.

The ABKEM ciphertext shall be annotated with an ABKEM attribute translated from attribute *`emergency`* and holding a value representing the emergency level applicable at the time of key encapsulation, that shall be set by the encapsulating party.

The ABKEM secret key shall contain a relational statement translated from `"(emergency eq emergency_level)"` in its access structure, where `emergency_level` shall represent the emergency level for which the secret key holder is authorized to bypass normal access control policies for the data protected in the ciphertext, when said emergency level is applicable.

7.4.7.2 Implementation in CP-ABKEM

Reduction of access control criteria in case of emergency relies on the secret key being annotated with an ABKEM string attribute representing the emergency level for which the secret key holder is allowed to bypass normal access control policies for the data protected in the ciphertext (when said emergency level is applicable) on the one hand, and on the ciphertext holding a policy that is being extended with an additional logical statement when the emergency level is applicable, on the other hand.

The emergency level may be detected by the device or registered to the device by external means.

Let an attribute *`emergency`* be of type string as defined in clause 7.2.2.2. Let constant `emergency_level` be of the same type.

The ABKEM secret key shall be annotated with an ABKEM attribute translated from attribute *`emergency`* and set to the value of the constant `emergency_level`, which shall represent an emergency level for which the owner of the secret key is allowed to bypass normal access control policies for the protected data.

When the emergency level represented by `emergency_level` is applicable, the policy that is contained in the ABKEM secret key access structure under normal conditions shall be extended with a logical statement translated from `"(emergency eq emergency_level)"`, where the constant `emergency_level` shall be set by the encapsulating party. When all branches of the policy can be bypassed under the same emergency condition, the logical statement shall be a top-level statement.

7.4.8 Access control based on service tier

7.4.8.1 Implementation in KP-ABKEM

Access control based on the service tier relies on the ciphertext being annotated with an ABKEM string attribute representing an applicable service tier set by the encapsulating party, and the secret key being valid for said value by means of a relational statement that compares the attribute against a string constant. If the service tier is applicable to all branches of the policy in the secret key, then this shall be a top-level statement.

Let an attribute *`attr`* be of Layer 1 type string as defined in clause 7.2.2. Let constant `tier` be of the same type.

The ABKEM ciphertext shall be annotated with an ABKEM attribute translated from attribute *`attr`* and holding a value representing the service tier applicable to the plaintext, that shall be set by the encapsulating party.

The ABKEM secret key shall contain a relational statement translated from `"(attr eq tier)"` in its access structure, where `tier` shall represent a service tier that is accessible to the owner of the secret key.

7.4.8.2 Implementation in CP-ABKEM

Access control based on the service tier relies on the secret key being annotated with an ABKEM string attribute representing a service tier available to the key owner, and the ciphertext being valid for said value by means of a relational statement that compares the attribute against a string constant. If the service tier is applicable to all branches of the policy in the ciphertext, then this shall be a top-level statement.

Let an attribute *`attr`* be of Layer 1 type string as defined in clause 7.2.2. Let constant `tier` be of the same type.

The ABKEM secret key shall be annotated with an ABKEM attribute translated from attribute *`attr`* set to the value `tier`, which shall represent a service tier accessible to the owner of the secret key.

The ABKEM ciphertext shall contain a relational statement translated from `"(attr eq tier)"` in its access structure, where the constant `tier` shall be set by the encapsulating party.

7.5 Translation rules for XACML

7.5.1 Introduction (informative)

The rules provided in the present clause are applicable to access control systems implementing the XACML Role Based Access Control Profile [7]. Two types of <`PolicySet`> elements in said profile are of interest for the present clause:

- the Role `<PolicySet>` element which associates a role with a Permission <`PolicySet`>, and
- the Permission `<PolicySet>` which contains permissions

7.5.2 General requirements

The requirements in [7] shall be supported. In addition, the following additional requirements shall be supported.

In order to allow full compatibility between the XACML-based access control system and the ABE access control system, the expressiveness of XACML elements shall be limited to the attribute types defined in clauses 7.2 and 7.3.

A <Rule> element shall always be instantiated with an Effect attribute set to the value "Permit".

7.5.3 Implementation in KP-ABKEM

7.5.3.1 KP-ABKEM specific requirements

A ciphertext shall be annotated with attributes representing the characteristics of the data, and may be annotated with contextual attributes. When issuing an access control policy, ABE authorities shall use attributes representing the characteristics of the data for the translation of the <Rule>'s <Target>, and shall use contextual attributes for the translation of the <Rule>'s <Condition>.

7.5.3.2 Issuance of secret keys

When issuing ABE secret keys to an access control subject, the ABE Authority shall, in this order:

1) identify the roles that apply to the access control subject;

2) for each role identified in the previous step, retrieve the Role <PolicySet> for which the <Target> element makes the Role <PolicySet> applicable to said role, if it exists, otherwise proceed to the next identified role;

3) from the Role <PolicySet> retrieved in the previous step, use the <PolicySetIdReference> element to identify the related Permission <PolicySet>;

4) from the identified Permission <PolicySet>, using the method defined in clause 7.5.3.3, obtain a set of logical statements conformant to clause 7.2.3, using attribute types defined at Layer 1 (clause 7.2) and Layer 2 (clause 7.3);

5) if an ABE secret key is to be issued, that applies to more than one logical statement obtained in step 4, build a policy where each selected logical statement shall be a top-level logical statement, and where the top-level logical statements shall be separated by the logical operator "OR";

6) translate the resulting policy (or each resulting policy) into an ABKEM policy according to the rules defined in clause 7.2.4.3 as well as rules defined for operators specific to Layer 2 attribute in clause 7.3.2

7) generate the access structure of the ABE secret key from the ABKEM policy obtained in the previous step.

NOTE: Step 4 is implementation specific as it requires that the XACML-based access control system and the ABE Authority are synchronized with regards to the attributes declared in the ABE universe.

7.5.3.3 Processing of Permission <PolicySet> element

When an ABE authority has identified a Permission <PolicySet>, it shall:

1) retrieve the tree of <PolicySet>, <Policy> and <Rule> elements by recursively following the <PolicySet>, <Policy> and <PolicySetIdReference> elements, starting from the Permission <PolicySet>;

2) from the tree constructed in step 1, identify all possible <Target> elements of <Rule> elements;

3) for each unique <Target> element, construct a view of the tree in step 2, that contains only <Rule> elements for which the <Target> applies;

4) for each view constructed in step 3, build a logical statement as defined in clause 7.2.3 according to the policy-combining algorithms of each <PolicySet> element in the view, the rule-combining algorithms of each <Policy> element in the view, and the FunctionId attribute of each <Condition> element in the view.

Allowed combining algorithms and functions are detailed in clause 7.5.5.

7.5.4 Implementation in CP-ABKEM

7.5.4.1 CP-ABKEM specific requirements

The ABE authority shall convert polices expressed in XACML into Layer 2 policies when configuring the ABE cryptosystem, such that devices can be provisioned with Layer 2 policies prior to ciphertext being generated.

Prior to converting policies expressed in XACML into Layer 2 policies, the ABE authority shall identify the characteristics of the data sets that will be encrypted by devices. These characteristics will be used to identify matching <Target> elements in <Rule> elements.

7.5.4.2 Preparation of policies for ciphertexts

When preparing Layer 2 policies for ABE ciphertext, the ABE Authority shall, in this order:

1) identify all the roles that are defined in the XACML-based access control system;
2) for each role identified in the previous step, retrieve the Role <PolicySet> for which the <Target> element makes the Role <PolicySet> applicable to said role, if it exists, otherwise proceed to the next identified role;
3) from the Role <PolicySet> retrieved in the previous step, use the <PolicySetIdReference> element to identify the related Permission <PolicySet>;
4) from the identified Permission <PolicySet> and the related role, using the method defined in clause 7.5.4.3 produce a database of logical statements conformant to clause 7.2.3 using attribute types defined at Layer 1 (clause 7.2) and Layer 2 (clause 7.3);
5) provision the database to devices.

NOTE: Step 4 is implementation specific as it requires that the XACML-based access control system and the ABE Authority are synchronized with regards to the attributes declared in the ABE universe.

7.5.4.3 Processing of Permission <PolicySet> element

When an ABE authority has identified a Permission <PolicySet>, it shall:

1) from the Permission <PolicySet>, retrieve the tree of <PolicySet>, <Policy> and <Rule> elements by recursively following the <PolicySet>, <Policy> and <PolicySetIdReference> elements, starting from the Permission <PolicySet>;
2) for each encrypting device (or device type) in the ABE cryptosystem, identify the set of <Rule>'s <Target> elements that will match the characteristics of the various data sets to be encrypted by said device (or device type);
3) for each <Target> element identified in step 2, construct a view of the tree built in step 1, that contains only <Rule> elements for which the <Target> applies;
4) for each view constructed in step 3, build a logical statement as defined in clause 7.2.3 according to the policy-combining algorithms of each <PolicySet> element in the view, the rule-combining algorithms of each <Policy> element in the view, the FunctionId attribute of each <Condition> element in the view, and the role related to the Permission <PolicySet>;
5) store each logical statement produced in step 4 into a database associating each logical statement with the dataset characteristics (e.g. using the related <Target> element instance) and the related role.

Allowed combining algorithms and functions are detailed in clause 7.5.5.

7.5.4.4 Encapsulation into ciphertext

Upon encapsulating a data set into an ABE ciphertext, the encapsulating party shall identify the characteristics of the data set and select the corresponding logical statement from the database generated according to clause 7.5.4.2. The logical statement shall be used to produce the ciphertext access structure.

In case the encapsulating party cannot identify a logical statement to use, it shall abort encryption and register an error to the ABE Authority. How this is done is out of scope of the present document.

7.5.5 Combining algorithms and functions

The policy-combining algorithms given in table 7.15 shall be supported, and for step 4 of clauses 7.5.3.3 and 7.5.4.3 the logical operator to be used between logical statements resulting from each <Policy> element shall be as indicated by the last column. Other policy-combining algorithms shall not be used.

Table 7.15: translation of policy-combining algorithms

Policy-combining algorithm	Policy-combing algorithm identifier in XACML [9]	Logical operator
Deny-override	urn:oasis:names:tc:xacml:3.0:policy-combining-algorithm:deny-overrides	AND
Ordered-deny-overrides	urn:oasis:names:tc:xacml:3.0:policy-combining-algorithm:ordered-deny-overrides	AND
Permit-overrides	urn:oasis:names:tc:xacml:3.0:policy-combining-algorithm:permit-overrides	OR
Ordered-permit-overrides	urn:oasis:names:tc:xacml:3.0:policy-combining-algorithm:ordered-permit-overrides	OR
Deny-unless-permit	urn:oasis:names:tc:xacml:3.0:policy-combining-algorithm:deny-unless-permit	OR
Permit-unless-deny	urn:oasis:names:tc:xacml:3.0:policy-combining-algorithm:permit-unless-deny	AND

The rule-combining algorithms given in table 7.16 shall be supported, and for step 4 of clauses 7.5.3.3 and 7.5.4.3 the logical operator to be used between logical statements resulting from each <Rule> element shall be as indicated by the last column. Other rule-combining algorithms shall not be used.

Table 7.16: translation of rule-combining algorithms

Rule-combining algorithm	rule-combing algorithm identifier in XACML [9]	Logical operator
Deny-override	urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:deny-overrides	AND
Ordered-deny-overrides	urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:ordered-deny-overrides	AND
Permit-overrides	urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:permit-overrides	OR
Ordered-permit-overrides	urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:ordered-permit-overrides	OR
Deny-unless-permit	urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:deny-unless-permit	OR
Permit-unless-deny	urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:permit-unless-deny	AND

The FunctionID attribute for the <Condition> element given in table 7.17 shall be supported, and for step 4 of clauses 7.5.3.3 and 7.5.4.3 the logical operator to be used between logical or relational statements expressed by the <Condition> element shall be as indicated in the last column.

Table 7.17: translation of logical functions

Function identifier in XACML [9]	Logical operator
urn:oasis:names:tc:xacml:1.0:function:or	OR
urn:oasis:names:tc:xacml:1.0:function:and	AND
urn:oasis:names:tc:xacml:1.0:function:n-of	Threshold gate

The FunctionID attribute for the <Condition> element given in table 7.18 shall be supported, and for step 4 of clauses 7.5.3.3 and 7.5.4.3 the relational operator and related attribute type to be used for relational statements expressed by the <Condition> element shall be as indicated in the last two column. Other values of the FunctionID attribute shall not be used.

Table 7.18: translation of relational functions

Function identifier in XACML [9]	relational operator	Base type to use in Layer 2 policies
urn:oasis:names:tc:xacml:1.0:function:string-equal	eq	STRING
urn:oasis:names:tc:xacml:1.0:function:boolean-equal	is_true, is_false	BOOL
urn:oasis:names:tc:xacml:1.0:function:integer-equal	==	UINT
urn:oasis:names:tc:xacml:1.0:function:double-equal	==	DOUBLE
urn:oasis:names:tc:xacml:1.0:function:date-equal	==	TIMESTAMP
urn:oasis:names:tc:xacml:1.0:function:time-equal	==	TIMESTAMP
urn:oasis:names:tc:xacml:1.0:function:dateTime-equal	==	TIMESTAMP
urn:oasis:names:tc:xacml:3.0:function:dayTimeDuration-equal	==	DURATION
urn:oasis:names:tc:xacml:3.0:function:yearMonthDuration-equal	==	DURATION
urn:oasis:names:tc:xacml:1.0:function:anyURI-equal	eq	STRING
urn:oasis:names:tc:xacml:1.0:function:x500Name-equal	eq	STRING
urn:oasis:names:tc:xacml:1.0:function:rfc822Name-equal	eq	STRING
urn:oasis:names:tc:xacml:1.0:function:hexBinary-equal	eq	STRING
urn:oasis:names:tc:xacml:1.0:function:base64Binary-equal	eq	STRING
urn:oasis:names:tc:xacml:1.0:function:integer-greater-than	>	UINT
urn:oasis:names:tc:xacml:1.0:function:integer-greater-than-or-equal	>=	UINT
urn:oasis:names:tc:xacml:1.0:function:integer-less-than	<	UINT
urn:oasis:names:tc:xacml:1.0:function:integer-less-than-or-equal	<=	UINT
urn:oasis:names:tc:xacml:1.0:function:double-greater-than	>	DOUBLE
urn:oasis:names:tc:xacml:1.0:function:double-greater-than-or-equal	>=	DOUBLE
urn:oasis:names:tc:xacml:1.0:function:double-less-than	<	DOUBLE
urn:oasis:names:tc:xacml:1.0:function:double-less-than-or-equal	<=	DOUBLE
urn:oasis:names:tc:xacml:1.0:function:time-greater-than	>	TIMESTAMP
urn:oasis:names:tc:xacml:1.0:function:time-greater-than-or-equal	>=	TIMESTAMP
urn:oasis:names:tc:xacml:1.0:function:time-less-than	<	TIMESTAMP
urn:oasis:names:tc:xacml:1.0:function:time-less-than-or-equal	<=	TIMESTAMP
urn:oasis:names:tc:xacml:2.0:function:time-in-range	<= and >=	TIMESTAMP
urn:oasis:names:tc:xacml:1.0:function:dateTime-greater-than	>	TIMESTAMP
urn:oasis:names:tc:xacml:1.0:function:dateTime-greater-than-or-equal	>=	TIMESTAMP
urn:oasis:names:tc:xacml:1.0:function:dateTime-less-than	<	TIMESTAMP
urn:oasis:names:tc:xacml:1.0:function:dateTime-less-than-or-equal	<=	TIMESTAMP
urn:oasis:names:tc:xacml:1.0:function:date-greater-than	>	TIMESTAMP
urn:oasis:names:tc:xacml:1.0:function:date-greater-than-or-equal	>=	TIMESTAMP
urn:oasis:names:tc:xacml:1.0:function:date-less-than	<	TIMESTAMP
urn:oasis:names:tc:xacml:1.0:function:date-less-than-or-equal	<=	TIMESTAMP

7.6 Authentication using ABKEM

7.6.1 Introduction

A simple challenge-response authentication mechanism based on ABE is hereby defined, where a claimant can prove their identity to a verifier. In order do so, the claimant possesses an ABE secret key such that the claimant can resolve the challenge encrypted using ABKEM to the criteria of the verifier.

The present clause focuses on the challenge-response mechanism itself. Mechanisms necessary to provide other properties needed for secure authentication, and bindings to application protocols are not in the scope of the present document.

7.6.2 Principles

The device may require authentication using ABKEM of the access control subject, in case of direct access.

The flow diagram below describes the overall resource request and challenge-response mechanisms.

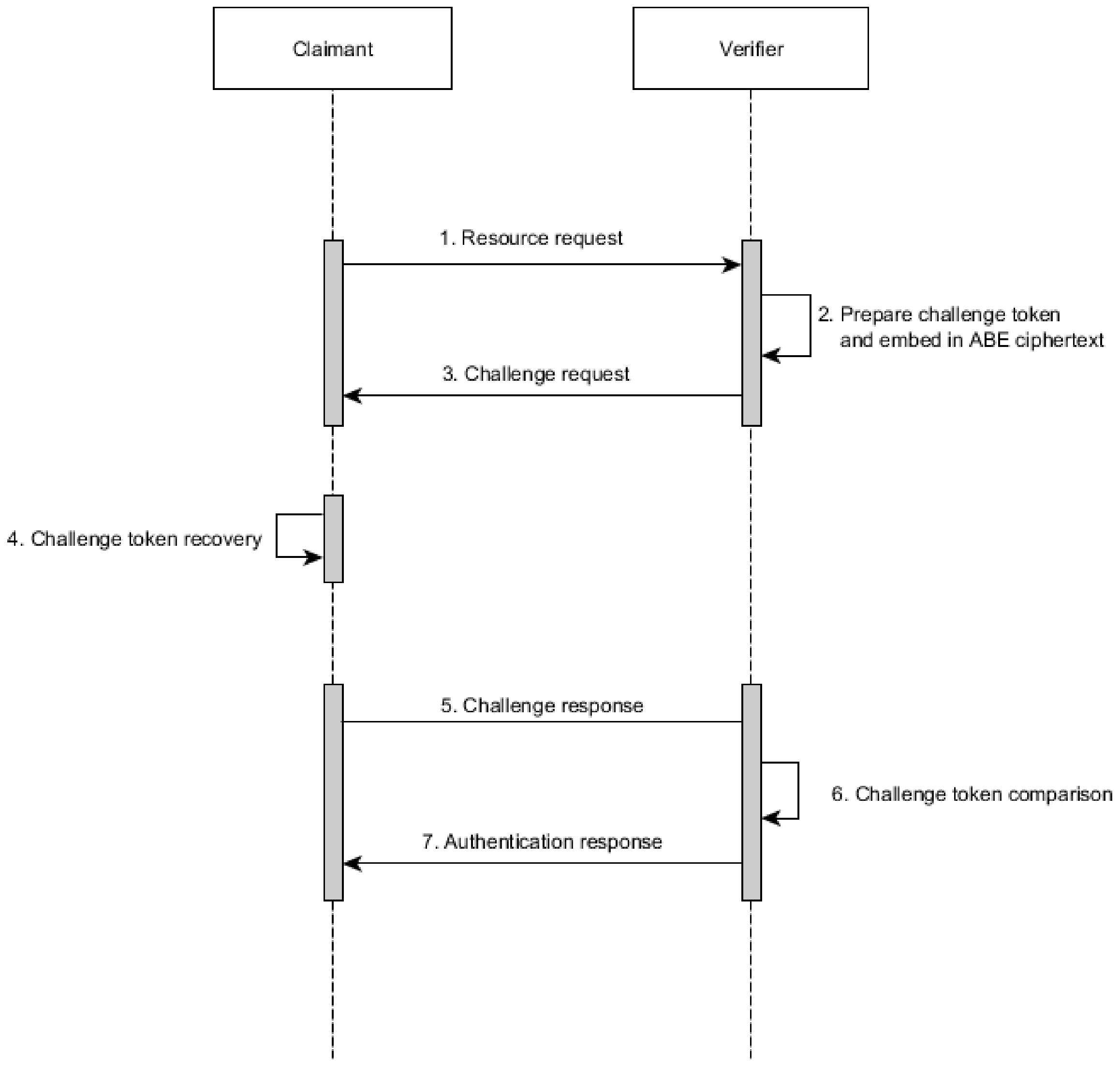
Claimant
Verifier
1. Resource request
2. Prepare challenge token and embed in ABE ciphertext
3. Challenge request
4. Challenge token recovery
5. Challenge response
6. Challenge token comparison
7. Authentication response

Figure 7.1: overall principle of the challenge-response mechanism for authentication using ABKEM

The steps are as follow:

1) The claimant issues a resource request including the resource identifier to be accessed and, optionally, the claimant's identity.

2) The verifier prepares a challenge token.

3) The verifier responds with a message indicating that an ABE challenge-response protocol will be initiated, the reference to the ABKEM universe that will be used, and the policy (or set of attributes) that the claimant will fulfil by means of an ABE secret key. The response further includes the challenge token encrypted in an ABE ciphertext. The response can include a reference (such as a URL) to the entity managing the ABKEM universe. The response can include a request to indicate the claimant identity.

4) Using the appropriate ABE secret key, the claimant recovers the challenge token from the ABE ciphertext.

5) The client responds with the challenge token. If the claimant identity was requested, the identity is appended to the response.

6) The verifier compares the received challenge token with the original one.

7) Authentication is successful when the two tokens match, in which case a response indicating authentication success is send by the verifier.

a) If authentication is not successful, the verifier sends an authentication failure response to the claimant.

8) The protocol for accessing the resource can continue.

7.6.3 Common messages

7.6.3.1 Resource identification

The resource request shall identify the resource by its `attribute-name` element. The request shall include the universe identifier `uni-id` element. The request shall not include any attribute (identified by the `attribute-name` element) that has not been declared within the ABKEM universe identified by `uni-id`.

7.6.3.2 Claimant identification

When the claimant identifies itself as a user, the user attribute type as defined in clause 7.3.2.4.4 shall be used.

When the claimant identifies itself as a device, the device attribute type as defined in clause 7.3.2.4.5 shall be used.

When the claimant identifies itself as a function, the function attributes type as defined in clause 7.3.2.4.6 shall be used.

The claimant may use several attributes to define its identity. In such case, the identity shall be interpreted as a logical conjunction of all attributes.

7.6.4 Implementation in KP-ABKEM

Using the identity of the claimant and the resource identifier, the verifier shall select an applicable access control policy and derive a set of attributes.

The verifier shall generate a challenge token. It shall execute a new key encapsulation algorithm using the derived set of attributes, produce a new symmetric key with its encapsulation, and use the symmetric key to encrypt the challenge token. The verifier shall return the ABE ciphertext to the claimant.

The claimant shall select an ABE secret key with a policy matching the attributes selected by the verifier, decapsulate the symmetric key from the ABE ciphertext, and use it to decrypt the challenge token.

7.6.5 Implementation in CP-ABKEM

Using the identity of the claimant and the resource identifier, the verifier shall select an applicable access control policy.

The verifier shall generate a challenge token. It shall execute a new key encapsulation algorithm using the selected access control policy, produce a new symmetric key with its encapsulation, and use the symmetric key to encrypt the challenge token. The verifier shall return the ABE ciphertext to the claimant.

The claimant shall select an ABE secret key with attributes matching the policy selected by the verifier, decapsulate the symmetric key from the ABE ciphertext, and use it to decrypt the challenge token.

Annex A (informative): ABE schemes from the cryptographic literature

The present annex lists CP-ABE and KP-ABE schemes from the cryptographic literature according to type, expressiveness, key and ciphertext sizes, security, large-universe, and key-revocation properties. The list is reported in table A.1.

Table A.1: ABE schemes available in the cryptographic literature

Scheme	Type	Expressive-ness	MPK	MSK	SK	Ciphertext	Security	Large Universe	Key revocation
[i.6] **Bethencourt, Sahai, & Waters, 2007** Ciphertext-Policy Attribute-Based Encryption	CP	Monotonic	Constant	Constant	Linear in number of attributes (that a secret key holds)	Linear in number of attributes in access structure	Generic group model, selectively secure	Yes	Yes (numerical key-revocation only)
[i.11] **Herranz, Laguillaumie, & Ràfols, 2010** Constant Size Ciphertexts in Threshold Attribute-Based Encryption	CP	Monotonic (threshold-gates only)	Constant	Linear in number of all possible attributes	Linear in number of all possible attributes	Constant	Standard model, aMSE-DDH, selectively secure	No	No
[i.20] **Rouselakis & Waters, 2016** Efficient Statically-Secure Large-Universe Multi-Authority Attribute-Based Encryption	CP	Monotonic	Constant	Constant	Constant	Linear in size of access structure	Random oracle model, staticly secure	Yes	No
[i.12] **Attrapadung & Imai, 2009** Conjunctive Broadcast and Attribute-Based Encryption	CP	Monotonic	Linear in number of all possible attributes	Constant (or linear in number of attributes)	Linear in number of all possible attributes	Linear in number of all policy size	Standard model, DBDDH, selectively secure	No	Yes
[i.13] **Ostrovsky, Sahai, & Waters, 2007** Attribute-Based Encryption with Non-Monotonic Access Structures	KP	Monotonic	Linear in number of attributes (that a ciphertext can hold)	Constant	Linear in number of attributes (that a ciphertext can hold)	Linear in number of attributes (that a ciphertext can hold)	Standard model, BDH, selectively secure	No	No
[i.15] **Lewko, Sahai, & Waters, 2010** Revocation Systems with Very Small Private Keys	KP	Non-Monotonic	Constant	Constant	Linear in number of attributes in access structure	Linear in number of attributes (that a ciphertext can hold)	Random oracle model, q-MEBDH, selectively secure	Yes	No
[i.14] **Attrapadung, Libert, & de Panafieu, 2011** Expressive Key-Policy Attribute-Based Encryption with Constant-Size Ciphertexts	KP	Monotonic	Linear in number of all possible attributes	Constant	Quadratic in the number of attributes	Constant (3 group elements)	Generic from Linear IBBE, n-DBDHE, selectively secure	No	No

Scheme	Type	Expressi ve-ness	MPK	MSK	SK	Ciphertex t	Security	Large Univers e	Key revocatio n
[i.12] **Attrapadung & Imai, 2009** Conjunctive Broadcast and Attribute-Based Encryption	KP	Monotonic	Linear in number of all possible attributes	Constant (or linear in number of attributes)	Linear in number of all policy size	Linear in number of all possible attributes	Standard model, DBDDH, selectively secure	No	Yes

Annex B (informative): Applicable features of traditional ABAC

Access control systems manage the access to resources ("objects") by its users ("subjects") for given purposes. An access control system provides means to recognize subjects, specify their authorizations, decide on access requests and maintains accountability.

In Attribute Based Access Control (ABAC), subject entitled to access a resource can be described by attributes they own. Attributes are assigned to subjects by one or more authorities. An attribute can be unique to a subject, thus to identify it, or shared by multiple subjects. The actual access ("policy enforcement") happens after the positive evaluation of policies defined over attributes ("policy decision").

The term "encrypted access control" is sometimes used to refer to ABE, because several analogies with ABAC systems. As ABE is a cryptographic technique, there are several differences with a conventional access control system. The main differences are enumerated below:

- Scope: ABAC has a more generic scope, covering both functional access control and data access control. ABE, natively, does not provide functional access control (however, it is possible to use any ABE cryptographic schema to protect the confidentiality of credentials that, in turns, are used to provide access to functions). Furthermore, ABE policies grant or deny read permissions, they do not provide any mean to govern other kind of permissions (delete, write, create, execute).

- Expressiveness: ABAC policies can express more detailed access rules and conditions than the ABE policies, which are mostly based on threshold gates in a tree based access structure.

- Support for Boolean operators: non-monotonic ABE schemes support negations, but inefficiently. Monotonic schemas support only AND and OR gates.

- Support for comparisons: numbers and dates comparison might be implemented indirectly, considering a sufficient number of threshold gates in the access structure of the ABE policy.

- Efficiency: The ABE policies and their corresponding tree access structures can be optimized for encryption/decryption by normalizing their nodes (by removing redundant Boolean operators).

- Accountability: ABE schemas do not provide means for accountability.

- Policy enforcement: In ABAC systems the access request is evaluated by a Policy Decision Point, and enforced by a Policy Enforcement Point. In ABE the decision is equivalent to the satisfaction of the access tree through the presented attributes and the enforcement is represented by the resulting decryption of the ciphertext.

- Data in motion: protection of data transfer between the system and the subject requesting the data is not part of a conventional ABAC system. Usually, state-of-art communication encryption techniques ensure such confidentiality. On the contrary, ABE can protect the confidentiality of data in motion.

Annex C (informative): Common semantics

C.1 Introduction

The common semantics leverages on the assignment of URI to metadata. This practice is compatible with the approach used in the definition of XML schemas and Web ontologies. Common metadata are applicable in different parts of the present document, e.g.:

- In the context of a policy, when attributes of the subject or object (resource) have to be specified.
- In assertions providing contextual data used to evaluate the polices such as: timestamp; location information (geocoordinates, country code, etc.); IP source addresses; etc.
- To provide common semantics for revocation and expiration (e.g. in X.509 certificates or inside policies).

C.2 Primitive data types

Primitive data types are converted from their string representations to values that can be compared with values in their domain of discourse, such as numbers, dates, time and Boolean values. The present document adopts primitive data type representations conforming to the ones used in XML schema specifications [8], in particular the following types:

- http://www.w3.org/2001/XMLSchema#string
- http://www.w3.org/2001/XMLSchema#boolean
- http://www.w3.org/2001/XMLSchema#integer
- http://www.w3.org/2001/XMLSchema#double

C.3 Time

In XML schema specifications [8], time attributes may take the following form:

- http://www.w3.org/2001/XMLSchema#time
- http://www.w3.org/2001/XMLSchema#date
- http://www.w3.org/2001/XMLSchema#dateTime
- http://www.w3.org/2001/XMLSchema#dayTimeDuration
- http://www.w3.org/2001/XMLSchema#yearMonthDuration

C.4 Location

The IETF GeoPriv Location Object (LO) is an object extending the "status" element of the PIDF (Presence Information Data Format) object defined for the Session Initiation Protocol (SIP). The geopriv element is based on the Geography Markup Language (GML). The GeoPriv defines two major mandatory subelements: one for location information and one for usage rules (IETF RFC 4119 [i.21]). In particular, the location-info element consists of one or more chunks of location Information. The format of the location information is identified by the imported XML Schema, which describes the namespace in use. Two relevant schemas are:

- Geographic coordinates - GML is a system for modelling geographic object types, topologies, metadata, coordinate reference systems, and units of measurement. The simplest package for GML supporting location is defined in the schema identified by the URN "urn:opengis:specification:gml:schema-xsd:feature:v3.0". By importing this schema, GML baseline schemas are transitively imported.

- Civic addresses - To define the encoding of civic location, it is possible to use the schema defined by the URN "urn:ietf:params:xml:ns:pidf:geopriv10:civicAddr".

C.5 Identifiers for resources

These types appear in several standard applications, such as TLS/SSL and electronic mail. Valid attributes may take the following form:

- X.500 directory name ("urn:oasis:names:tc:xacml:1.0:data-type:x500Name"): an ITU-T Rec. X.520 [i.30] Distinguished Name. The syntax is described in IETF RFC 2253 [i.22].

- IETF RFC 822 [i.29] name ("urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name): an electronic mail address. The syntax is described in IETF RFC 2821 [i.23].

- IP address ("urn:oasis:names:tc:xacml:2.0:data-type:ipAddress"): an IPv4 or IPv6 network address. IPv4 address address and mask is formatted in accordance with the syntax for a "host" presented in [i.25]. IPv6 addresses and masks follow the syntax described in IETF RFC 2732 [i.24].

- DNS name ("urn:oasis:names:tc:xacml:2.0:data-type:dnsName"): represents a Domain Name Service (DNS) host name, with optional port or port range. The hostname is formatted in accordance with IETF RFC 2396 [i.25].

- XPath expression ("urn:oasis:names:tc:xacml:3.0:data-type:xpathExpression"): The syntax is defined by the XPath W3C recommendation [i.26].

- SIP URI (syntax defined inside IETF RFC 3261 [i.27]): All public service identifiers need to meet the specific requirements of services such as: Voice, Instant Messaging Service, Presence Service, Location Service, etc.

- Any URI ("http://www.w3.org/2001/XMLSchema#anyURI"): any URI, syntax described in XML schema specification [8].

C.6 Domain specific ontologies

C.6.1 Introduction

An ontology is a vocabulary with a structure, which applies to a domain of interest with additional properties. An ontology provides a formal model a machine can manipulate and allow the interpretation of constructs ETSI TS 118 112 [i.16]. The present clause describes ontologies that provides common semantics for different domains.

C.6.2 OneM2M Base Ontology

The purpose of the oneM2M Base Ontology is to provide a minimal set of conventions so that other ontologies can be mapped to oneM2M (such as SAREF). Interoperability is thus ensured via mapping between ontologies ETSI TS 118 112 [i.16].

When an access control policy based on a given ontology is evaluated against another ontology, several translation mechanisms are required:

- data type conversion,
- variable substitution, expansion, and translation (e.g. for subjects and objects),
- operator and predicate translation (a condition is expressed differently),
- clause translation (e.g. a location expressed as a building identifier may need to be translated into a polygon with coordinates).

These translation mechanisms are particularly relevant for the distribution of attribute keys (CP-ABE) and of policy keys (KP-ABE) to subjects.

C.6.3 ISO/IEC 19944

ISO/IEC 19944 [i.9] does not provide a specific ontology, however this standard provides a taxonomy that can be used to develop ontologies. Built on several ISO/IEC standards [i.1], [i.10] and [i.8], ISO/IEC 19944 [i.9] introduces the concept of Device Platform Provider, a specialized Cloud Service Provider implementing identity management for users and interfaces for third party services. Noticeably, ISO/IEC 19944 [i.9] specifies a taxonomy of individual user's data categories and a vocabulary defining data identification qualifiers, i.e. qualifiers that indicates the degree to which data are linked to an individual.

Annex D (normative): Grammars for the attribute based access control layer

D.1 Introduction

The present annex defines ABNF [13] grammars for various declarations related to clause 7. Definitions of content types are not in the scope of the present document.

D.2 Universe and attribute declarations

A file containing a ABKEM universe declaration shall be formatted according to the ABNF [13] grammar below.

```
universe = version SP uni-type SP uni-id SP crypto-params CRLF 1*(attribute-decl)
version = 1*(ALPHA / DIGIT)
uni-type = "CP-ABKEM" / "KP-ABKEM"
uni-id = uni-name "." uni-version
uni-name = 1*(ALPHA / DIGIT)
uni-version = 1*(ALPHA / DIGIT / ":")
crypto-params = 1*(VCHAR)

attribute-decl = "define" SP attribute-def "." max-occurence [SP source-datatype] CRLF
attribute-def = attribute-type "." attribute-name
max-occurence = %x31-39 *DIGIT
source-datatype = 1*(VCHAR)

attribute-name = component *(sep-namespace component) *1(sep-extension component)
component = 1*(ALPHA / DIGIT)
sep-namespace = ":"
sep-extension = "-"

; Layer 1 attributes types
attribute-type = attribute-type-uint
attribute-type-uint = "UINT(" parameter *(comma parameter) ")"
parameter = 1*DIGIT
comma = ","

attribute-type =/ attribute-type-bool
attribute-type-bool = "BOOL"

attribute-type =/ attribute-type-string
attribute-type-string = "STRING"

; Layer 2 attributes
attribute-decl =/ double-decl / timestamp-decl / duration-decl / cycles-decl
attribute-decl =/ zone-decl / grid-decl / d1-point-decl / d2-point-decl / d3-point-decl
attribute-decl =/ circle-decl / sphere-decl / freestring-decl / clearance-decl / role-decl
attribute-decl =/ user-decl / device-decl / function-decl / datatype-decl / origin-decl

double-decl = "define" SP double-type "." (attribute-name) "." max-occurence [SP source-datatype]
CRLF
double-type = "DOUBLE(" uint-value "," uint-value ")"

timestamp-decl = "define" SP timestamp-type "." (attribute-name) "." max-occurence [SP source-
datatype] CRLF
timestamp-type = "TIMESTAMP(" uint-value ")"
timestamp-type =/ "TIMESTAMP(" uint-value "," string-plain ")"
timestamp-type =/ "TIMESTAMP(" uint-value "," string-plain "," string-plain ")"

duration-decl = "define" SP duration-type "." (attribute-name) "." max-occurence [SP source-
datatype] CRLF
duration-type = "DURATION(" uint-value ")"
duration-type =/ "DURATION(" uint-value "," string-plain ")"

cycles-decl = "define" SP cycles-type "." (attribute-name) "." max-occurence [SP source-datatype]
CRLF
cycles-type =/ "CYCLES(" uint-value "," string-plain ")"

zone-decl = "define" SP zone-type "." (attribute-name) "." max-occurence SP zone-allowed-values [SP
source-datatype] CRLF
```

```
cycles-type =/ "ZONE(" uint-value ")"
zone-allowed-values = "allowed values (" string-allowed-value-component *("," string-allowed-value-
component) ")"

string-allowed-value-component = string-value-plain / string-value-base64

grid-decl = "define" SP grid-type "." (attribute-name) "." max-occurence [SP source-datatype] CRLF
grid-type =/ "GRID(" uint-value "," uint-value "," string-plain ")"

d1-point-decl = "define" SP d1-point-type "." (attribute-name) "." max-occurence [SP source-
datatype] CRLF
d1-point-type =/ "1D-POINT(" uint-value "," string-plain "," string-plain ")"

d2-point-decl = "define" SP d2-point-type "." (attribute-name) "." max-occurence [SP source-
datatype] CRLF
d2-point-type =/ "2D-POINT(" uint-value "," uint-value "," string-plain "," string-plain ")"

d3-point-decl = "define" SP d3-point-type "." (attribute-name) "." max-occurence [SP source-
datatype] CRLF
d3-point-type =/ "3D-POINT(" uint-value "," uint-value "," uint-value "," string-plain "," string-
plain ")"

circle-decl = "define" SP circle-type "." (attribute-name) "." max-occurence [SP source-datatype]
CRLF
circle-type =/ "CIRCLE(" uint-value "," string-plain "," string-plain ")"

sphere-decl = "define" SP sphere-type "." (attribute-name) "." max-occurence [SP source-datatype]
CRLF
sphere-type =/ "SPHERE(" uint-value "," string-plain "," string-plain ")"

freestring-decl = "define" SP freestring-type "." attribute-name "." max-occurence SP source-
datatype CRLF

clearance-decl = "define" SP clearance-type "." (attribute-name) "." max-occurence SP clearance-
allowed-values [SP source-datatype] CRLF
clearance-type = "CLEARANCE"
clearance-allowed-values = "allowed values (" string-allowed-value-component *("," string-allowed-
value-component) ")"

role-decl = "define" SP role-type "." (attribute-name) "." max-occurence SP role-allowed-values [SP
source-datatype] CRLF
role-type = "ROLE"
role-allowed-values = "allowed values (" string-allowed-value-component *("," string-allowed-value-
component) ")"

user-decl = "define" SP user-type "." (attribute-name) "." max-occurence [SP source-datatype] CRLF
user-type = "USER"

device-decl = "define" SP device-type "." (attribute-name) "." max-occurence [SP source-datatype]
CRLF
device-type = "DEVICE"

function-decl = "define" SP function-type "." (attribute-name) "." max-occurence [SP source-
datatype] CRLF
function-type = "FUNCTION"

datatype-decl = "define" SP datatype-type "." (attribute-name) "." max-occurence [SP source-
datatype] CRLF
datatype-type = "DATATYPE"

origin-decl = "define" SP origin-type "." (attribute-name) "." max-occurence [SP source-datatype]
CRLF
origin-type = "ORIGIN"
```

D.3 Policy declarations

A file containing a ABKEM policy declaration shall be formatted according to the ABNF [13] grammar below.

```
policy-doc = universe-ref 1*(policy CRLF)
universe-ref = "universe:" SP uni-id uni-version CRLF

policy = policy-id SP policy-version SP logical-statement
policy-id = 1*(ALPHA / DIGIT) *(("-" / ".") 1*(ALPHA / DIGIT))
policy-version = 1*(DIGIT) *(("-" / ".") 1*(DIGIT))
```

```
logical-statement = relational-statement / "(" logical-statement log-op logical-statement ")"
logical-statement =/ threshold-gate

relational-statement = "(" attribute-name SP rel-op-int SP uint-constant ")"
rel-op-int = "<" / "<=" / ">" / ">=" / "==" / "!="
uint-constant = 1*DIGIT

relational-statement =/ "(" attribute-name SP rel-op-bool ")"
rel-op-bool = "is_true" / "is_false"

relational-statement =/ "(" attribute-name SP rel-op-string SP string-constant ")"
rel-op-string = "eq"
string-constant = string-plain / string-base64

log-op = "AND" / "OR"
threshold-gate = threshold "_OF(" logical-statement 1*(comma logical-statement) ")"
threshold = %x31-39 *DIGIT
```

D.4 Attribute assignments at Layer 1

A file containing a ABKEM attribute assignment shall be formatted according to the ABNF [13] grammar below.

```
assignement-l1-doc = universe-ref 1*(assignment CRLF)
universe-ref = "universe:" SP uni-id uni-version CRLF

assignment = "set:" SP attribute-type-bool "." attribute-name SP bool-value
bool-value = ("0" / "1") ; true or false

assignment =/ "set:" SP attribute-type-uint "." attribute-name SP uint-value
uint-value = 1*(DIGIT)

assignement =/ "set:" SP attribute-type-string "." attribute-name SP string-value-plain
string-value-plain = string-plain
string-plain = "string:plain:" string-plain-value
string-plain-value = 1*(SP / VCHAR)

assignement =/ "set:" SP attribute-type-string "." attribute-name SP string-value-base64
string-value-base64 = string-base64
string-base64 =/ "string:encoded:base64:" charset ":" string-base64-value
charset = 1*(ALPHA / DIGIT / "-")
string-base64-value = (1*(4base64-char) [base64-pad]) / base64-pad
base64-pad = (2base64-char "==") / (3base64-char "=")
base64-char = ALPHA / DIGIT / "+" / "/"
```

D.5 ABKEM attribute encoding

A file containing a ABKEM attribute encoding shall be formatted according to the ABNF [13] grammar below.

```
abkem-encoding-doc = universe-ref 1*(abkem-encoding CRLF)
universe-ref = "universe:" SP uni-id uni-version CRLF

attribute-id = attribute-name "." id
id = %x31-39 *DIGIT

abkem-encoding = abkem-encoding-uint
abkem-encoding =/ abkem-encoding-bool
abkem-encoding =/ abkem-encoding-string-plain
abkem-encoding =/ abkem-encoding-string-base64

abkem-encoding-uint = attribute-type-uint "." attribute-id "." bit-position "." bit-value
bit-position = 1*DIGIT
bit-value = "0" / "1"

abkem-encoding-bool = attribute-type-bool "." attribute-id "." bool-value

abkem-encoding-string-plain = attribute-type-string "." attribute-id "." string-plain

abkem-encoding-string-base64 = attribute-type-string "." attribute-id "." string-base64
```

Annex E (informative): Bibliography

Agrawal, S., Chase, M. (2017): "FAME: Fast Attribute-based Message Encryption". Cryptology ePrint Archive, Report 2017/807.

Attrapadung, N., & Imai, H. (2009): "Conjunctive Broadcast and Attribute-Based Encryption", Pairing, (p. 248-265).

Attrapadung, N., Libert, B., & de Panafieu, E. (2011): "Expressive Key-Policy Attribute-Based Encryption with Constant-Size Ciphertexts", Public Key Cryptography, (p. 90-108).

Chase, M. (2007): "Multi-authority Attribute Based Encryption", TCC, (p. 515-534).

Goyal, V., Jain, A., Pandey, O., & Sahai, A. (2008): "Bounded Ciphertext Policy Attribute Based Encryption", ICALP, (p. 579-591).

Goyal, V., Pandey, O., Sahai, A., & Waters, B. (2006): "Attribute-based encryption for fine-grained access control of encrypted data", ACM Conference on Computer and Communications Security, (p. 89-98).

Herranz, J., Laguillaumie, F., & Ràfols, C. (2010): "Constant Size Ciphertexts in Threshold Attribute-Based Encryption", Public Key Cryptography, (p. 19-34).

Lewko, A. B., & Waters, B. (2011): "Decentralizing Attribute-Based Encryption", EUROCRYPT, (p. 568-588).

Lewko, A. B., Okamoto, T., Sahai, A., Takashima, K., & Waters, B. (2010): "Fully Secure Functional Encryption: Attribute-Based Encryption and (Hierarchical) Inner Product Encryption", EUROCRYPT, (p. 62-91).

Ostrovsky, R., Sahai, A., & Waters, B. (2007): "Attribute-based encryption with non-monotonic access structures", ACM Conference on Computer and Communications Security, (p. 195-203).

Waters, B. (2011): "Ciphertext-Policy Attribute-Based Encryption: An Expressive, Efficient, and Provably Secure Realization", Public Key Cryptography, (p. 53-70).

History

Document history		
V1.1.1	March 2018	Publication