ETSI TS 103 964 V1.1.1 (2025-02)

Cyber Security (CYBER);
A Verifiable Credentials extension using Attribute-Based Encryption

Reference
DTS/CYBER-0098

Keywords
access control, confidentiality, portability, privacy

ETSI

650 Route des Lucioles
F-06921 Sophia Antipolis Cedex - FRANCE

Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Siret N° 348 623 562 00017 - APE 7112B
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° w061004871

Important notice

The present document can be downloaded from the
ETSI Search & Browse Standards application.

The present document may be made available in electronic versions and/or in print. The content of any electronic and/or print versions of the present document shall not be modified without the prior written authorization of ETSI. In case of any existing or perceived difference in contents between such versions and/or in print, the prevailing version of an ETSI deliverable is the one made publicly available in PDF format on ETSI deliver repository.

Users should be aware that the present document may be revised or have its status changed, this information is available in the Milestones listing.

If you find errors in the present document, please send your comments to the relevant service listed under Committee Support Staff.

If you find a security vulnerability in the present document, please report it through our Coordinated Vulnerability Disclosure (CVD) program.

Notice of disclaimer & limitation of liability

The information provided in the present deliverable is directed solely to professionals who have the appropriate degree of experience to understand and interpret its content in accordance with generally accepted engineering or other professional standard and applicable regulations.
No recommendation as to products and services or vendors is made or should be implied.
No representation or warranty is made that this deliverable is technically accurate or sufficient or conforms to any law and/or governmental rule and/or regulation and further, no representation or warranty is made of merchantability or fitness for any particular purpose or against infringement of intellectual property rights.
In no event shall ETSI be held liable for loss of profits or any other incidental or consequential damages.

Any software contained in this deliverable is provided "AS IS" with no warranties, express or implied, including but not limited to, the warranties of merchantability, fitness for a particular purpose and non-infringement of intellectual property rights and ETSI shall not be held liable in any event for any damages whatsoever (including, without limitation, damages for loss of profits, business interruption, loss of information, or any other pecuniary loss) arising out of or related to the use of or inability to use the software.

Copyright Notification

No part may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm except as authorized by written permission of ETSI.
The content of the PDF version shall not be modified without the written authorization of ETSI.
The copyright and the foregoing restriction extend to reproduction in all media.

© ETSI 2025.
All rights reserved.

Contents

Intellectual Property Rights

Essential patents

IPRs essential or potentially essential to normative deliverables may have been declared to ETSI. The declarations pertaining to these essential IPRs, if any, are publicly available for **ETSI members and non-members**, and can be found in ETSI SR 000 314: *"Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards"*, which is available from the ETSI Secretariat. Latest updates are available on the ETSI IPR online database.

Pursuant to the ETSI Directives including the ETSI IPR Policy, no investigation regarding the essentiality of IPRs, including IPR searches, has been carried out by ETSI. No guarantee can be given as to the existence of other IPRs not referenced in ETSI SR 000 314 (or the updates on the ETSI Web server) which are, or may be, or may become, essential to the present document.

Trademarks

The present document may include trademarks and/or tradenames which are asserted and/or registered by their owners. ETSI claims no ownership of these except for any which are indicated as being the property of ETSI, and conveys no right to use or reproduce any trademark and/or tradename. Mention of those trademarks in the present document does not constitute an endorsement by ETSI of products, services or organizations associated with those trademarks.

DECT™, **PLUGTESTS™**, **UMTS™** and the ETSI logo are trademarks of ETSI registered for the benefit of its Members. **3GPP™**, **LTE™** and **5G™** logo are trademarks of ETSI registered for the benefit of its Members and of the 3GPP Organizational Partners. **oneM2M™** logo is a trademark of ETSI registered for the benefit of its Members and of the oneM2M Partners. **GSM®** and the GSM logo are trademarks registered and owned by the GSM Association.

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 defines a new proof method via a challenge-response authentication protocol based on predicate encryption, in particular on Ciphertext Policy-Attribute Based Encryption (CPABE). In this proof method, CP-ABE keys encode attributes, while the expressiveness of CP-ABE policies enables their selective disclosure and/or anonymous proof of predicates over them. Relationship with existing Zero Knowledge Proof [i.16] methods is highlighted.

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 in the ETSI docbox.

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] IETF RFC 8017: "PKCS #1: RSA Cryptography Specifications Version 2.2", November 2016.

[2] IETF RFC 7517: "JSON Web Key (JWK)", May 2015.

[3] IETF RFC 7516: "JSON Web Encryption (JWE)", May 2015.

[4] IETF RFC 7515: "JSON Web Signature (JWS)", May 2015.

[5] IETF RFC 4648: "The Base16, Base32, and Base64 Data Encodings", October 2006.

[6] ETSI TS 103 532 (V1.2.1) (05-2021): "CYBER; Attribute Based Encryption for Attribute Based Access Control".

[7] IEEE 1363.3™-2013: "Standard for Identity-Based Cryptographic Techniques using Pairings".

[8] W3C®: "Verifiable Credentials Data Model v2.0". W3C Candidate Recommendation Draft, 19 January 2025.

[9] W3C®: "Verifiable Credential Data Integrity 1.0, Securing the Integrity of Verifiable Credential Data", W3C Candidate Recommendation Draft, 03 August 2024.

[10] W3C®: "Controlled Identifiers (CIDs) v1.0", W3C Working Draft, 26 January 2025.

[11] IETF RFC 7519: "JSON Web Token (JWT)", May 2015.

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] A. Lewko and B. Waters: "Decentralizing Attribute-Based Encryption", Cryptology ePrint Archive, Paper 2010/351, 2010.

[i.2] T. Looker, V. Kalos, A. Whitehead, M. Lodder: "The BBS Signature Scheme", Internet Draft, draft-irtf-cfrg-bbs-signatures-07, June 2024.

[i.3] Sakemi, Y., Kobayashi, T., Saito, T. and R. S. Wahby: "Pairing-Friendly Curves", Internet-Draft, draft-irtf-cfrg-pairing-friendly-curves-11, 6 November 2022.

[i.4] B. Waters: "Ciphertext-policy attribute-based encryption: An expressive, efficient, and provably secure realization", Cryptology ePrint Archive, Paper 2008/290, 2008.

[i.5] Zeutro LLC Encryption & Data Security: "The OpenABE Design Document", Version 1.0.

[i.6] Deuber D., Maffei M., Malavolta G., Schröder M.R.D., Simkin M.: "Functional credentials". Proceedings on Privacy Enhancing Technologies (04-2018).

[i.7] Boneh D., Boyen X., Shacham H.: "Short Group Signatures". In: Franklin, M. (eds) Advances in Cryptology - CRYPTO 2004. Lecture Notes in Computer Science, vol. 3152. Springer, Berlin, Heidelberg, 2004.

[i.8] Dan Boneh and Hovav Shacham: "Group Signatures with Verier-Local Revocation". In ACM CCS, 2004.

[i.9] Tessaro S., Zhu C.: "Revisiting BBS Signatures". Cryptology ePrint Archive, Paper 2023/275 (2023).

[i.10] Song D.X.: "Practical forward secure group signature schemes". In: Proceedings of the 8th ACM Conference on Computer and Communications Security. pp. 225-234. CCS '01, Association for Computing Machinery, New York, USA (2001).

[i.11] D. Chaum: "Security without identification: Transaction systems to make big brother obsolete". Commun. ACM, (10), 1985.

[i.12] J. Camenisch and A. Lysyanskaya: "An Efficient System for Nontransferable Anonymous Credentials with Optional Anonymity Revocation". In EUROCRYPT, LNCS. Springer, 2001.

[i.13] Jan Camenisch and Els Van Herreweghen: "Design and implementation of the idemix anonymous credential system". In ACM CCS. ACM, 2002.

[i.14] Christian Paquin and Greg Zaverucha: "U-Prove Cryptographic Specification V1.1", Revision 3. Technical report, Microsoft Corporation, 2013.

[i.15] Hyperledger: "AnonCreds Specification".

[i.16] Goldwasser-Micali-Rackoff: "The knowledge complexity of interactive proof systems", 1985.

[i.17] J. Groth and A. Saha: "Efficient non-interactive proof systems for bilinear groups". In N. P. Smart, editor, EUROCRYPT 2008, vol. 4965 of LNCS, pp 415-432, Istanbul, Turkey, April 13-17, 2008. Springer, Heidelberg, Germany.

[i.18] E. Fasllija: "The Role of Verifiable Data Registries in the Verifiable Credential Ecosystem", online report by the Secure Information Technology Centre, Austria, 2024.

3 Definition of terms, symbols and abbreviations

3.1 Terms

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

Anonymous authentication: process or action of proving a credential is valid, without tracing back the Owner

Anonymous proof : proof for which the Verifier has no reasonable chance to trace back the Prover

Claim: assertion made about a (Data) Subject

Claimant: For the purpose of the present document, same as Prover.

Credentials: data attesting to the truth of certain stated facts

NOTE: For the purpose of the present document, a credential is in form of a set of one or more Claims.

Credential holder: entity which receives credentials from an Issuer

Data Subject: identified or identifiable natural person to which the data relates, or device that produces data that can be linked to a natural person

Holder: entity possessing of one or more Credentials

Issuer: issuing authority accredited or supervised for issuing Credentials

Key management: administration and use of the generation, registration, certification, deregistration, distribution, installation, storage, archiving, revocation, derivation and destruction of keying material in accordance with a security policy

Membership proof: method by which a Prover can prove to Verifier that an element is in a given set of elements without disclosing the actual element

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

Presentation: act of presenting a Credential to a Verifier

Presentation Policy: policy used by a Verifier defining access control criteria Credentials shall satisfy

Proof of predicate: method by which a Prover can prove to Verifier that an attribute satisfies a given Boolean assertion, without disclosing the actual value of the attribute

Prover: entity presenting a credential to a Verifier

Range proof: method by which a Prover can prove to Verifier that a numeric value is in a given range (lower and upper bound) without disclosing the actual value

Selective disclosure: capability that enables the user to present only a subset of user-provided attributes

Subject: For the purpose of the present document, same as Data Subject.

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

Unlinkability: capability of ensuring that a user may make multiple uses of resources or services without others being able to link these uses together

Verifiable Credentials: tamper-evident credential whose authorship can be cryptographically verified

Verifiable Presentation: tamper-evident presentation of information encoded in such a way that authorship of the data can be trusted after a process of cryptographic verification

NOTE: Certain types of verifiable presentations might contain data that is synthesized from, but does not contain, the original verifiable credentials.

Verifier: entity that receives verifiable presentations and process them via cryptographic verification

Zero Knowledge Proof: mathematical proof that conveys no additional knowledge other than the correctness of the proposition in need of proving

3.2 Symbols

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

A, B, C, D	Attributes
l, λ	Security parameter (natural number)
Z_p	Set of integers modulo prime number p
i	Imaginary unit (such that $i^2 + 1 = 0$ in Z_p)
P(x,y)	Point on elliptic curve E_1
P(x',y')	Point on elliptic curve E_2
E_1, E_2	Pairing friendly elliptic curves
GF(p)	Cyclic group of order p
$GF(p^2)$	Cyclic group of order p^2
r	Order of the subgroup of E_1 over GF(p) and of E_2 over $GF(p^2)$
G_1, G_2	Cyclic subgroups of order r
G_T	Target group
g_1, g_2	Generators for G_1, G_2
$GF(p^{12})$*	Multiplicative Cyclic group of order p^{12}
G_T	Target group (in bilinear pairing)
e(•,•)	Bilinear pairing function
α or alpha, a, b, u, v, t	Secret key components in Z_p (in CP-WATERS-ABE construction)
S	Set of attributes
K, L, K_x	key components (in CP-WATERS-ABE construction)
C', C_k, D_k	Ciphertext components (in CP-WATERS-ABE construction)
AP	String representing an access policy
(M; ρ)	Access structure made of a LSSS matrix M and a mapping function ρ() (in CP-WATERS-ABE construction)
$\vec{v} = (s, y_2, \ldots, y_m)$	Vector of random elements in Z_p used to produce the ciphertext (in CP-WATERS-ABE construction)
H, H'	Hash functions (e.g. SHA-256)
u	A seed (normally obtained from the application of a collision-resistant hash function)
s, r_k	Pseudo random generated values
λ_k, ω_k	Algorithmically computed exponents
K, r	Bit strings
A = (M; ρ), A'	Access structure
F	Policy
Φ	The (infinite) universe of policies
K_0, K_1	User keys
⊥	null
Ω	The universe of attributes (may be finite or infinite)
Pr[]	Probability evaluation function
⊕	XOR (bitwise operator)
||	Concatenation operator

3.3 Abbreviations

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

ABE	Attribute Based Encryption
AP	Access Policy
BBS	Boneh, Boyen, and Shacham
CCA	Chosen Ciphertext Attack
CP-ABE	Ciphertext Policy-Attribute Based Encryption
CT	CipherText
LSSS	Linear Secret Sharing Scheme

M	Message to encrypt
MPK	Master Public Key
MSK	Master Secret Key
PK	Public Key
PRG	Pseudo Random Generator
SK	(User) Secret Key

4 ABE challenge/response authorization method

4.1 Introduction

4.1.1 Overview

The present document defines a challenge-response authentication and authorization method based on Attribute Based Encryption (ABE). The method was originally described in ETSI TS 103 532 [6], clause 7.6 as a method for authentication, however the present document extends and generalize its purpose by introducing authentication by anonymous proof of predicates. Depending on the specific chosen predicate encryption schema, the resulting protocol may provide different expressiveness. For example, some initial ABE schemas were not be able to (efficiently) support negation ("NOT") operator, while recent ones do. By using the hereafter described method, a Prover (formerly named "Claimant" in ETSI TS 103 532 [6]) can prove the possession of attributes fulfilling a certain policy - referred to as "presentation policy" - to a Verifier without necessarily revealing her identity. To do so, the Prover receives a credential (i.e. an ABE secret key) enabling her to resolve a challenge proposed by the Verifier.

While afore mentioned ETSI TS 103 532 [6], clause 7.6 focuses on the challenge-response abstract protocol, mechanisms necessary to provide additional properties for *anonymous authentication* and user *unlikability* are object of the present document.

Born with the seminal work of [i.11], anonymous authentication has experimented waves of renewed interest during the years and is now becoming popular due to the raise of various user "wallet" models. The attempt to provide standard cryptographic primitives for privacy-preserving identity credentials is today ongoing at various standardization fora. For example, at the time of writing, IEFT is intended to standardize a recent efficient construction of the Boneh, Boyen, and Shacham (BBS) signature [i.9], originally introduced with the paper [i.7].

Historically, various anonymous credentials built on specialized signatures have been proposed as a mean to implement anonymous authentication. Generally speaking, in such a scheme, the Prover, after obtaining a signature over a set of attributes from an Issuer, randomizes it and proves in *zero knowledge* its possession to a Verifier, optionally revealing a subset of those attributes. This is referred to as *selective disclosure*. Note that the Verifier is unable to determine which signature was used to generate the proof, removing any source of correlation (unlinkability).

However, in some contexts, selective disclosure is not the only desired feature. For example, a service may require that their users are over 18 years old and that they are based in one of the European Countries, without learning the actual values of these user's attributes. In such a case, it is necessary to implement an *anonymous proof of predicates* proving that user's attributes "age" and "Country" satisfy the following presentation policy:

```
age GT 18 AND country ONEOF {Austria, Belgium,…, Sweden}
```

Where `GT` (greater than), `AND`, `ONEOF` represent *predicates*. Note that in this example the user has not to disclose the value of her attributes `age` and `country`, while the Prover does not learn other information than the fact that the user's attributes satisfy the above presentation policy.

4.1.2 Functional Credentials

Functional credentials, introduced by [i.6], are a generalization and unification of anonymous credentials and their derivates. Functional credentials use a scheme almost exclusively built on top of *predicate encryption*, providing a very natural way to prove predicates. Given a ciphertext encoding a presentation policy, e.g. CP-ABE ciphertext, a Prover can simply decrypt such a ciphertext to convince a Verifier that she knows a key embedding a set of attributes matching the policy. As opposite to signature-based credentials, Functional credentials are natively anonymous, i.e. the set of attributes issued to a user by an issuing Authority is always kept private by default, being embedded in the user's secret key. By combining various operators, predicates may natively achieve wide expressiveness.

4.1.3 Verifiable Credentials

The present document maps Functional credentials to W3C® defined Verifiable Credentials [8]. At the time of writing, existing or ad-hoc invented additional signature schemes are being progressively introduced in Verifiable Credentials to fit zero knowledge requirements. Essentially these new signature schemes turn a two-party relationship Signer/Verifier into a three party one: Issuer, Prover, Verifier. Predicate encryption natively implements this three-party relationship, being based on an Authority/Encrypting party/Decrypting party model. Featured with a native policy definition language, a verification protocol based on predicate encryption may efficiently support several kinds of anonymous proofs (including *selective disclosure*, *proof of membership*, *range proof* and a whole variety of complex predicate proofs combining the aforementioned ones).

4.2 Protocol

4.2.1 Description

The present clause amends the protocol presented in ETSI TS 103 532 [6], clause 7.6 as follows:

1) The prover **shall** issue a resource request including the resource identifier to be accessed. If desired, the prover **may** include the subject's identity.

NOTE 1: In the original protocol, an optional parameter was the subject's identity. However, in the present document, which support anonymous authentication, the subject identity is not necessarily disclosed.

2) The verifier **shall** choose a nonce (and proper randomness to initiate CCA-secure ABE encryption. See clause A.3.1).

3) Using a presentation policy, the verifier **shall** execute CCA-secure ABE encryption and **shall** respond with a message indicating that an ABE challenge-response protocol will be initiated. The response **shall** include the ABE ciphertext and the presentation policy in clear.

NOTE 2: In CP-ABE the presentation policy consists in an access structure, which is made public by the encrypting party. A compiler is used to translate the high-level presentation policy into an access structure. A simple example is reported in Annex C.

4) Using an ABE secret key fulfilling the presentation policy, the prover **shall** decrypt the ABE ciphertext and recover the secret token.

5) The prover **shall** then reiterate the request at step 1) including the decrypted secret token.

6) The verifier **shall** compare the received secret token with the original one. If the two tokens match, the authentication is successful, and the verifier **shall** give the prover access to the requested resource.

7) If authentication is not successful, the verifier **shall** repeat the procedure from step 2.

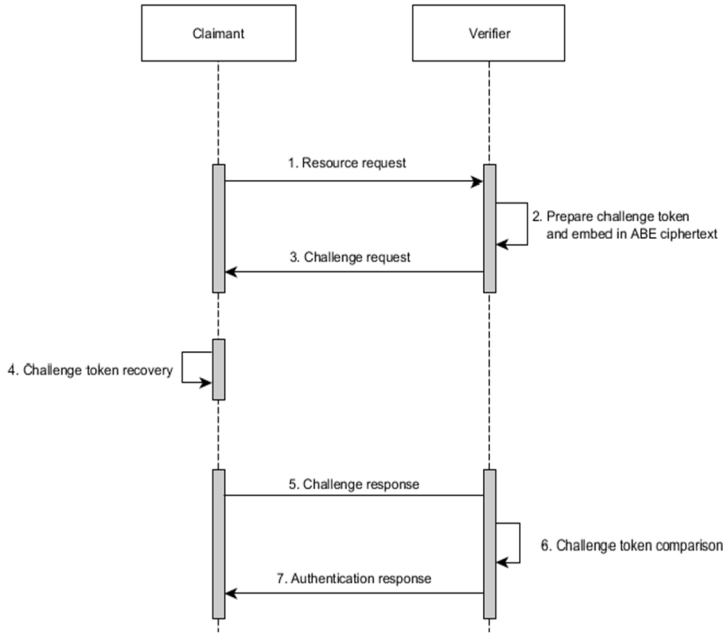

Figure 4.2.1-1: High-level view of the protocol described in ETSI TS 103 532 [6], clause 7.6

Note that to prevent ciphertext forgeability, the predicate encryption schema used in this protocol **shall** be CCA-resistant.

4.2.2 Predicate Encryption Schema

The present document builds on Cyphertext Policy Attribute-Based Encryption (CP-ABE) as predicate encryption schema. In CP-ABE [i.4], decryption keys embed user's attributes while an arbitrary predicate may be embedded in the ciphertext. The message is disclosed to users holding a key only if the attributes match the predicate.

4.2.3 Running Example

To illustrate the protocol, the present document proposes a scenario used as a running example throughout the whole document. The scenario has three actors: an Issuer, a Prover and a Verifier and it is "standalone", meaning that specific integration to higher level protocols (e.g. HTTP, OAuth, etc.) is not a goal of the present document.

The flow of events is as follows:

1) The Prover (decrypting party) receives from the Issuer (ABE Authority) a verifiable credentials containing a set of attributes.

2) The Prover requests access to a resource at the Verifier.

3) The Verifier (encrypting party) creates a challenge by encrypting a secret using a presentation policy and send it to the Prover.

4) The Prover is able to decrypt the ciphertext and recover the secret.

5) The Prover presents a verifiable presentation containing the decrypted secret to the Verifier.

The example uses the CP-ABE schema described in [i.4] and ETSI TS 103 532 [6] and reported in Annex A, which explicitly considers constructions implemented over elliptic curves and pairings. In particular, state-of-art BLS12-381 curves (Annex B) are used.

Attributes generated for the Prover are simple literals `A` and `D`. Presentation policy is a simple combination of Boolean operators:

```
A AND (D OR (B AND C))
```

and may be compiled using the simple compiler proposed in Annex C.

To prevent ciphertext forgeability, the predicate encryption schema used in the protocol is secure under chosen ciphertext attacks. Following recommendations in ETSI TS 103 532 [6], the construction referred to as CCA-secure CP-ABKEM is adopted. For details, the reader may refer to Annex A.

NOTE: The present document - as well as ETSI TS 103 532 [6] - considers ABE constructions over pairings only. Despite additional implementations (i.e. post-quantum lattice-based) are being proposed for ABE, their concrete adoption is still under discussion and therefore are not hereby considered.

4.3 Main Concepts

4.3.1 Anonymous Credentials and Zero Knowledge Proof

Anonymous credentials initially proposed by Chaum [i.11] and firstly implemented by Camenisch and Lysyanskaya [i.12] allow users to prove the possession of a set of attributes in zero knowledge. Goldwasser, Micali, and Rakoff defined proposed the first Zero Knowledge Proof (ZKP) scheme back in 1985 [i.16] and defined ZKP as "*those proofs that convey no additional knowledge other than the correctness of the proposition in need of proving"*. A survey on ZKP is out of scope for the present document. The Zero Knowledge property is of particular interest for credentials because it ensures that two Verifiers cannot share any common piece of data that could allow them to infer that they are dealing with the same user (a property known as *unlikability*). Such a condition also applies between Issuers and Verifiers.

Several today existing anonymous credentials schemas are based on encrypted or committed signatures in combination with Zero Knowledge Proofs. The most prevalent approaches are based on IBM's Idemix [i.13] issued in early 2000 followed by Microsoft's U-Prove [i.14] released in 2013. In the context of distributed applications, an example of a real-world deployed signature-based anonymous credentials is AnonCreds, specified by Hyperledger [i.15] in 2018 and built on top of the Camenisch-Lysyanskaya signatures and using Camenisch's cryptographic accumulators for revocation. At the present time, version 2 of AnonCreds specifications is under development.

4.3.2 Functional Credentials

The present document aims at simplifying construction, deployment and adoption of *credentials schemas supporting anonymous proof of predicates* by building on encryption schemas rather than on signature-based schemas. [i.6] refers to this kind of credential as *Functional credentials*. A predicate encryption scheme allows one to embed an arbitrary set of attributes *S* in the user decryption keys and produce a ciphertext for predicates *f* so that the plaintext may be recovered only by users holding a key such that $f(S) = 1$. The scheme guarantees that the set of attributes S is kept private, except for the information trivially revealed by the validity of the predicate for the encoded attributes.

In such a scheme, a way to verify credentials anonymously is to prove in zero knowledge the successful decryption of a given ciphertext. However, this direct application of generic zero knowledge proofs over predicate encryption schemes may result computationally prohibitive, as proving in zero knowledge the successful decryption of a given ciphertext (using Groth-Sahai proofs [i.17]) would make the proof scaling with the size of the ciphertext of a predicate encryption making it unusable for practical purposes.

Functional credentials implement anonymity using a different approach: a verifier composes a challenge simply by encrypting a random secret and sends it to the prover. In the schema proposed in [i.6], in order to prove the knowledge of a key (thus possession of attributes), the prover participates in building the ciphertext by choosing a random number which the verifier uses to build the ciphertext. The verifier chooses a second random number and sends a commitment of this parameter to the prover. Finally, the verifier combines the two numbers into a XOR expression and obtain a random secret and the randomness to be used to build a ciphertext.

After this "setup" phase, the prover decrypts the ciphertext obtaining the random secret. However, she does not send it immediately to the verifier, rather sends back a commitment of the decrypted message.

In the final step, the verifier reveals the randomness that she used to compute the ciphertext so that the prover can locally verify that the ciphertext has been not forger. If this holds true, the prover finally sends the opening information. If the opening reconstructs to the original message, the verifier is convinced that the prover possesses the correct key, thus the correct attributes.

Note that this approach introduces some extra rounds in the protocol presented in clause 4.2, which are essentially intended to guarantee that the ciphertext produced by the verifier was truly randomly generated and was not forged to allow the verifier to extract information from the presenter's key.

However, for some predicate encryption schemas, the extra steps in the above presented protocol might be redundant. This is of benefit for many existing authentication protocols (such as HTTP and OAuth) which rely on a three steps procedure (request-challenge-response).

NOTE: In the case of CCA-secure CP-ABKEM of ETSI TS 103 532 [6], used as a running example throughout the present document, the protocol is in fact simplified omitting the commitment steps. Clause D.1 illustrates a proof showing that the correctness, unforgeability and anonymity properties are still preserved in the simplified version.

4.3.3 Presentation Policy Verification

In CP-ABE, each policy is implemented through an access structure containing attributes to be matched. A presentation policy is translated in an access structure using a compiler (the algorithm reported in Annex C is an example of a simple compiler supporting basic operators). To avoid that an attacker, impersonating the Verifier, might use specially crafted policies to gain more knowledge than necessary from a prover, it is needed to ensure that the presentation policy the Verifier is claiming exactly translates into access structure made available with the ciphertext.

The decrypting party (the Prover) **may** verify if the used access structure matches a given policy as follows:

1) The Prover **shall** obtain the access structure A = *extractAccessStructure()* from the ciphertext.

2) The Prover **shall** recompute the access structure A' starting from the presentation policy the Verifier claims.

3) If $A == A'$ the Prover **shall** proceed, otherwise it **shall** stop.

A and A' **shall** be calculated using the same compiler, therefore details about the compiler **should** be publicly available.

4.3.4 Credential Revocation

The present document considers three kinds of revocation: time-based, list-based and accumulator-based.

Time-based revocation uses timestamps attributes contained in the credentials. Attributes "issued at", "not before than", "expires at" defined in [11] **may** be used for this purpose. A Verifier **shall not** accept credentials presented before the date and time specified under the attribute "issued at" attribute if present, nor under the attribute "not before than" if present. Similarly, a Verifier **shall not** accept credentials presented after the date and time specified under the attributes "expires at" if present. This check **may** be implemented through the presentation policy.

Revocation based on whitelists **may** use specific attributes embedded in the credentials (i.e. unique identifiers or serial numbers). When an attribute-based whitelist mechanism is used, the Issuer **shall** maintain a public whitelist of attributes and **shall** embed whitelisted attributes in the credentials. The Verifier **shall not** accept credentials whose attributes values are not matching the ones in the whitelist. To invalidate revoked credentials, the Issuer **shall** periodically update the whitelist.

To enforce a whitelist approach, the Verifier **may** implement a specific policy accepting only non-revoked credentials to be valid.

An alternative is to use a blacklist approach which works for predicate encryption schemas supporting negations by defining a policy excluding blacklisted credentials. Using blacklists, the Verifier **shall not** accept credentials whose attributes values are matching the ones in the blacklist.

One drawback to implement whitelists or blacklists is the length of the policy that grows linearly with the number of involved credentials. Also, the size of the list could allow for privacy threats. The smaller is the list, the easier for the verifier to provide correlation between two or more presentation of the same credential.

Furthermore, when implementing lists in policies, the entity that control revocation is in fact the Verifier, not the Issuer. Therefore, this approach, called "verifier-local revocation", reveals to be formally incorrect. In fact, by exploiting verifier-local revocation, malicious Verifiers could potentially craft a set of special policies able to create correlation when the same credential is used with two or more different verifiers. Note that this issue affects also anonymous credentials based on signature schemas [i.8].

NOTE 1: To mitigate this threat in Functional credentials, the Prover should always be able to compare the list in the presentation policy with the one maintained by the Issuer.

NOTE 2: The verifier-local revocation approach also introduces some problems with backward unlinkability [i.10], as the revocation of a credential may imply the linkability of past credential presentations.

When using revocation based on cryptographic accumulators, each issued credential embeds a secret value that a Verifier **shall** check against a public value that **shall** be maintained by the Issuer (the accumulator). If the check is not successfully, the verifier **shall not** accept the presented credential. When the accumulator changes, Provers **shall** update the secret value embedded in their credentials. Usually this happens by using public values, without the need for the Prover to be reissued with new credentials.

4.4 Architecture and Reference Points (normative)

4.4.1 Architecture

The system architecture presented in the present document consists of three entities: an Issue, a Prover and a Verifier. In case of CCA-secure CP-ABE based on CP-WATERS-KEM, these are mapped to an ABE Authority, a Decrypting party, and an Encrypting party. These three entities, therefore, **shall be** able to execute the algorithm described in Annexes A, B and C of the present document.

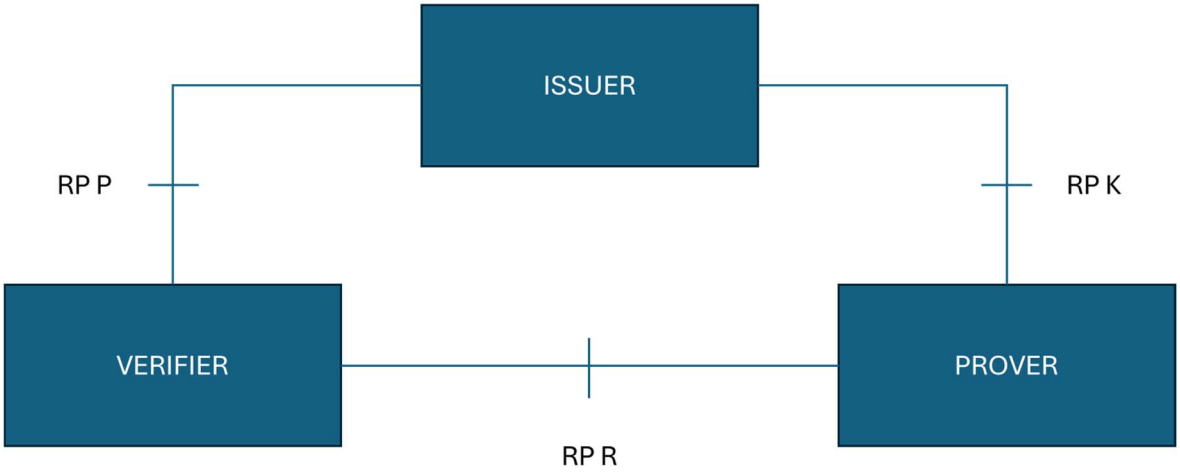

Figure 4.4.1-1: Architecture

In particular, the ABE Authority **shall** be able to execute the ABE set-up algorithm for CP-ABE, generating the corresponding master public key MPK and master secret key MSK. The Authorization Server **shall** also be able to generate keys based on a set of attributes.

The Encrypting party **shall** be able to run ABE encryption algorithm using the master public key MPK generated by the Authority and a presentation policy (encoded as in Annex C).

The Decrypting party **shall** have a protected environment where it is able to store confidentially the secret keys received from the Authority and run the ABE decryption algorithm.

The interactions between the three aforementioned entities happens at Reference Points K, P and R, hereafter specified.

NOTE: Requirements for interfaces implementing Reference Points K, P and R are generally satisfied when using HTTPS connections.

4.4.2 Reference Point K (key distribution)

The Reference Point K allows the Authority to distribute secret keys to the Decrypting party. The corresponding interface **shall** ensure that the entity corresponding to the Authority is authenticated as well as confidentiality and integrity protection of exchanged keys and metadata. The involved parties **may** also mutually authenticate.

4.4.3 Reference Point P (public parameters distribution)

Reference Point P is used by the Authority which makes publicly available the master public key and any other needed public parameter (e.g. attributes, accumulators) and metadata. The corresponding interface **shall** ensure that the entity corresponding to the Authority is authenticated as well as integrity protection of exchanged data. Any party **shall** be able to publicly access data exposed though this interface.

4.4.4 Reference Point R (challenge response)

Reference Point between the Decrypting party and the Encrypting party. The corresponding interface **shall** execute the protocol specified in clause 4.2. The entity corresponding to the Prover (Decrypting party) **may** require the entity corresponding to the Verifier (Encrypting party) to authenticate. The corresponding interface **shall** ensure confidentiality and integrity protection of exchanged data.

5 Verifiable Credentials

5.1 Introduction

This clause provides an informal outline of the Verifiable Credentials model defined in [8]. In this model:

- A Verifiable Credential is a set of one or more claims made by the same entity.
- A claim is a statement about a subject. A subject is a thing about which claims can be made.
- Claims are expressed using subject-property-value relationships.
- Issuers release Verifiable Credentials to Holders.
- Any Holder may transfer Verifiable Credentials to other Holders.
- The Holder stores the Verifiable Credentials in a repository and may delete them.
- To authenticate or provide access to a service, the Holder which assume the role of a Prover, generates a Verifiable Presentation and presents it to a Verifier.

NOTE 1: Throughout the present document, the role of the Prover is assumed to be generally matching with the role of a "Holder", if the credentials have not been transferred. Mechanisms to transfer credentials and power of attorney are out of the scope for the present document.

- The Verifier verifies the authenticity of the presented Verifiable Presentation. This verification shall include checking the credential status for revocation of the Verifiable Credential the presentation has been generated from.
- Each Verifier trusts the Issuer to produce valid Verifiable Credentials.
- Holders and Verifiers trust the Issuer to produce (through a proof establishing that the Issuer generated the credential) and release valid Verifiable Credentials, and to revoke them when appropriate.
- An Issuer does not necessarily need to trust (or even know) the Verifier.
- Verifiable Credentials are securely stored in a repository. Any Holder trusts the repository to not release credentials to anyone other than the Holder, and to not corrupt or lose them while they are in its care. Issuer and the Verifier do not necessarily need to trust the repository.
- A verifiable data registry contains information about which data is controlled by which entities. All entities shall trust the registry to be tamper-evident and correct.

NOTE 2: Report [i.18] contains more information and implementation considerations for verifiable data registries.

- Contrary to different trust models (e.g. a Certificate Authority trust models), in Verifiable Credentials trust relationships are not transitive.

5.2 Interface Implementing Reference Point K

5.2.1 Overview

The present clause uses the definition of Verifiable Credentials (whose description and the semantics is detailed in [8]) to implement Reference Point K. While a Verifiable Credential may consist in several claims, expressed using subject-property-value relationships, this clause is not intended to report a comprehensive description of them, but only of those of interest for mapping Functional credentials. Both in Verifiable Credential and in Verifiable Presentation, the claim "proof" [9] serves this purpose.

5.2.2 Claim proof

This claim **shall** contain at least one cryptographic proof that **may** be used to detect tampering and verify the authorship of a credential or presentation. The proof appears is in the form of a composite object, whose specific format and semantics **shall** be defined by the "type" property.

Credential proofs are traditionally signature-based: credential subjects' attributes are represented as signed plaintext messages. In Functional credentials, instead, attributes are "embedded" in keys issued by an Authority, therefore each proof contains such keys.

type

The mandatory claim refers to the proof type. The claim **shall** consist in a property whose value is a single string identifying the proof type.

The present document defines the new types **`FunctionalCredential`** for Functional credentials and **`FunctionalCredentialPresentation`** for Functional credential presentation format and the following subtypes:

`FunctionalCredential_2023_CP_WATERS_KEM`

`FunctionalCredentialPresentation_2023_CP_WATERS_KEM`

proofPurpose

The purpose of this mandatory property is to prevent the proof from being misused for a purpose other than the one it was intended for. This claim **shall** consist in a property whose value is an array of strings identifying the proof purpose. The present document allows the following two proof purpose values among the several predefined ones described in [8]: **`authentication`** and **`capabilityInvocations`**.

verificationMethod

This mandatory claim contains a set of parameters required to verify the proof.

The value of the "verificationMethod" claim property **shall** be an URI referencing an object of type "verificationMethod", hosted inside an external document, which is called "controller document".

A specific implementation **shall** provide content integrity protection. When the "verificationMethod" is referenced through an URI, the protocol used to acquire the "verificationMethod" object **may** use a valid HTTPS connection.

NOTE 1: Referencing a "verificationMethod" rather than embedding it in the credentials may allow for more flexibility in key management, therefore it is currently the only recommended best practice adopted by the present document. Clause 5.3 specifies the use of referenced controller documents to implement Reference Point P.

proofValue

This mandatory claim **shall** consist in a property whose value is of type string and express base-encoded binary data necessary to verify the digital proof using the specified "verificationMethod". It **shall** use the Multibase encoding as described in Section 2.4 of the Controller Documents 1.0 specification [10] to express the binary data.

The Multibase encoding consists of a string including a single character header which identifies the base and encoding alphabet used to encode a binary value, followed by the encoded binary value (using that base and alphabet).

For the purposes of the present document, the Base64URL without padding encoding **shall** be used. The assigned character header which identifies this latter is "**u**".

The claim **shall** contain one Authority-generated CP-ABE secret key, embedding attributes referred to any claim to prove in the Verifiable credential object. Each claim **shall be** mapped to one (or more) ABE attributes. In particular:

- claims whose datatype are just strings **shall** be one-to-one mapped to ABE attributes;
- depending on the specific ABE scheme and the compiler used to translate policies, claims with other datatypes (e.g. integers, date and time, geolocation) **may** be supported as well and **should** follow the standard translation defined in ETSI TS 103 532 [6].

Inside each credential proof, a CP-ABE secret key **shall** be a Base64URL encoded JSON object; each key component **shall** correspond to a JSON sub-object. The syntax presented in JSON Web Key (JWK) [2] **shall** be used to represent these components.

NOTE 2: W3C® [8] specifications allows for more than one credential proof as part of this property. While the present document limits to only one credential proof, corresponding to a single Authority-generated CP-ABE secret key, such "parallel credentials" may be useful in various scenarios outside the scope of the present document.

5.2.3 Example: Verifiable Credentials (informative)

The following example refers to the sample scenario presented throughout the present document and uses BLS12-381 elliptic curves. In this example, the Prover receives attributes A and D from the Issuer. Annex B provides more details about the serialization format used to represent any point on the two elliptic curves used in the pairing. For efficiency reasons, key components embedding attributes are translated into point on E_1 curve, while other key components are on E_2 curve. Point compression (i.e. sending the x coordinate of a point but omitting the y coordinate - which is later reconstructed at the receiver's side) is used to reduce the overall size of the object.

Assume the Issuer is an ABE Authority using the construction CP-ABE-KEM (reported in clause A.2). The Authority randomly generates the following (private) parameters:

```
a:0x3b07e4a6ccad2b6d74038739fa3674adb29793a3476f8790308867e40697678d
alpha:0x83cc59a4be42cdd0c26db275c569bbe2a63fa4fb861319aadb567e057c767ab
t:0x68e7098bdaf2f5c74ae20e61348ebd7baeb6f85bb5a6e8cc493fad91bdb268bc
```

And the following public attributes, as point on the curve E_1, encoded in a JSON document following IETF RFC 7517 [2]:

```
{
   "attributes":[
      {
         "kty":"EC",
         "crv":"BLS-12-381",
         "x":"pCWSh5v9KXv46isfWAg83SkVl7_B-4K68prY3ukopKRBJR-k1PDri2rDZCZsrqYh",
         "kid":"H(A)"
      },
      {
         "kty":"EC",
         "crv":"BLS-12-381",
         "x":"hj2lzvfcYevX9RWMU2zZottNnBqQz1JbXgDPv5ZPZmhNfUAM-pEXy3wWHc40QwH1",
         "kid":"H(B)"
      },
      {
         "kty":"EC",
         "crv":"BLS-12-381",
         "x":"gVJ__Y-3op8jhvaGBBEPb0JDkp49dF2kmjlz3xrVQmWh3NTEuocC9uCAR7-jrnoK",
         "kid":"H(C)"
      },
      {
         "kty":"EC",
         "crv":"BLS-12-381",
         "x":"iRYg8UgCPuLExMwspRuxqJnmABCz68iJTpweKp5j0A-CDtUrVpET9rjdzZD_3jk_",
         "kid":"H(D)"
      }
   ]
}
```

NOTE 1: Throughout the present document, whitespaces are added to JSON objects for readability purpose. They are ignored in any processing.

Hence, the Authority generates the following key components $K = g_2^{\alpha} g_2^{at}$, $L = g_2^{t}$ and for each $x \in S$ $K_x = H(x)^t$:

```
{
   "key":[
      {
         "kty":"EC",
         "crv":"BLS-12-381",
"x":"pGr_pIroAW2RvIViPCPlEWRN_Fn3M1_tkzZsoxaggQW2BV7wXuhBKJurh1zMZhePA3pLAeZO3E6SbsJKWbN7_6VPu5xv8fs
Pk7ZZqGAkiK8dzxNFhEBPlg1aHddBwVRo",
         "kid":"K=g_2^{\\alpha+a*t}"
      },
      {
         "kty":"EC",
         "crv":"BLS-12-381",
         "x":"tCdv9mpnA38RqRe9RCkbd3oHmaw-eODN8-jABZmokxJcG3hwy43uz4l9kX1LiC5IENr-
HX5ECgi3Oyka3fY4MHosLHxgWmOyumQ_tiQOKiXUR6p-xUSVQ4BZlWZAMx6E",
         "kid":"L=g_2^t"
      },
      {
         "kty":"EC",
         "crv":"BLS-12-381",
         "x":"olscdTuqCDgeQNcbFt-aCDyxQaQCuWgzSmny2vxznuPdBZVoCjt_DwTD7KDyMUbs",
         "kid":"K_A=H(A)^t"
      },
      {
         "kty":"EC",
         "crv":"BLS-12-381",
         "x":"kIriJ1B1XiyPVLBDGh8cvDJHs1bnx1j0ARAQ0BZkMLD0dmSmh42bV2kqf2jxlqlO",
         "kid":"K_D=H(D)^t"
      }
   ]
}
```

NOTE 2: Conventionally, the present document names each key components after the Latex expression used in the officially recognized scientific article describing the original CP-ABE schema. This is not normative and other conventions may be used as well.

The key is released to the Decrypting party encoded as a Multibase document, as follows:

```
{
   "proof":{
      "type":"FunctionalCredential_2023_CP_WATERS_KEM",
      "created":"2023-01-01T00:00:01Z",
      "proofPurpose":[
         "capabilityInvocations"
      ],
      "verificationMethod":"https://www.example.org/vc/cp-abe/v1/public-parameters#1",
"proofValue":"ueyJrZXkiOlt7Imt0eSI6IkVDIiwiY3J2IjoiQkxTLTEyLTM4MSIsIngiOiJwR3JfcElyb0FXMlJ2SVZpUENQb
EVXUk5fRm4zTTFfdGt6WnNveGFnZ1FXMkJWN3dYdWhCS0p1cmgxek1aaGVQQTNwTEFlWk8zRTZTYnNKS1diTjdfNlZQdTV4djhmc
1BrN1pacUdBa2lLOGR6eE5GaEVCUGxnMWFIZGRCd1ZSbyIsImtpZCI6Iks9Z18yXntcXGFscGhhK2EqdH0ifSx7Imt0eSI6IkVDI
iwiY3J2IjoiQkxTLTEyLTM4MSIsIngiOiJ0Q2R2OW1wbkEzOFJxUmU5UkNrYmQzb0htYXctZU9ETjgtakFCWm1va3hKY0czaHd5N
DN1ejRsOWtYMUxpQzVJRU5yLUhYNUVDZ2kzT3lrYTNmWTRNSG9zTEh4Z1dtT3l1bVFfdGlRT0tpWFVSNnAteFVTVlE0QlpsV1pBT
Xg2RSIsImtpZCI6Ikw9Z18yXnQifSx7Imt0eSI6IkVDIiwiY3J2IjoiQkxTLTEyLTM4MSIsIngiOiJvbHNjZFR1cUNEZ2VRTmNiR
nQtYUNEeXhRYVFDdVdnelNtbnkydnh6bnVQZEJaVm9DanRfRHdURDdLRHlNVWJzIiwia2lkIjoiS19BPUgoQSledCJ9LHsia3R5I
joiRUMiLCJjcnYiOiJCTFMtMTItMzgxIiwieCI6ImtJcmlKMUIxWGl5UFZMQkRHaDhjdkRKSHMxYm54MWowQVJBUTBCWmtNTEQwZ
G1TbWg0MmJWMmtxZjJqeGxxbE8iLCJraWQiOiJLX0Q9SChEKV50In1dfQ"
   }
}
```

Note the prepended character header **u** in the "proofValue" encoding and the following Base64URL no padding encoding the JSON Web Key.

5.3 Interface Implementing Reference Point P

5.3.1 Overview

The present clause defines the interface implementing Reference Point P using the definition of the "controller document" object specified in Verifiable Credentials [8].

A controller document [10] **shall** consist in one or more "verificationMethod". Each "verificationMethod" contains verification material. For the present document, the verification material **shall** consist in any public parameters available from the specific adopted schema, plus other meta information, namely:

1) JSON Web Keys (JWK) [RFC7517] to represent any public key component.

NOTE 1: [8] recommends that verification methods using JWKs [2] to represent their public keys using the value of "kid" property as their fragment identifier. However, as Functional credentials present more complex public key, made of a set of components, alternative ways (e.g. Latex representation as proposed in the example) are accepted.

2) A reference to the used compiler: **may** be provided as a trusted web application translating a policy into an access structure or as a definition of this translation (e.g. a lex/yacc command grammar file).

3) Any attribute definition in case allowed attributes are limited (so called "small universe" ABE schemas), or reference to any function converting attributes from their string representation format to group elements (generally points on elliptic curve) for ABE "large universe" schemas.

NOTE 2: [8] mandates that verification methods should be registered in the Verifiable Data Registries. However, at the time there is no evidence of established specifications for Verifiable Data Registries. However, the reader may refer to report [i.18] for several implementation considerations.

5.3.2 Verification Method Claims

Each "verificationMethod" **shall** contain the following mandatory claims:

verificationMethod.Id

This claim **shall** consist in a property whose value is a single URI identifying the "verificationMethod".

verificationMethod.Controller

This claim **shall** consist in a property whose value is a single URI identifying the Controller, i.e. the Authority.

verificationMethod.Type

This claim **shall** consist in a property whose value is a single string that references exactly one verification method type. The type of a verification method **shall** be used to determine the process (i.e. algorithm) performing the verification method.

In addition, "verificationMethod" **shall**:

- contain a claim "MPK", that **shall** consist in a property whose value is an array of specific public parameter for each ABE scheme. Each parameter **shall** be encoded using JSON Web Key (JWK) [2];
- contain a claim "compiler", which consists in a property whose value is a URI referencing to the compiler used to encode presentation policies.

5.3.3 Example: Controller Document (informative)

Continuing the running example, the Authority releases the master public key: $MPK = g_1, g_1^a , e(g_1, g_2)^\alpha, H(x)$ *for each* x *in* $\{A,B,C,D\}$, encodes it in the following controller document, specifies the compiler to be used to encode presentation policies and makes it publicly available at the Controller's URI. The reader may refer to Annex B for the encoding formats and in particular clause B.6 for the encoding format of the element $e(g_1, g_2)^\alpha$.

```
{
   "id":"https://www.example.org/vc/cp-abe/v1/public-parameters",
   "verificationMethod":[
      {
         "id":"https://www.example.org/vc/cp-abe/v1/public-parameters#1",
         "type":"FunctionalCredential_2023_CP_WATERS_KEM_PublicParameters",
         "controller":"https://www.example.org/",
         "compiler":"https://www.example.org/vc/cp-abe/v1/compiler.yacc",
         "MPK":[
            {
               "kty":"EC",
```

```
            "crv":"BLS12-381",
            "x":"l_HTpzGX15QmlWOMT6msD8NojE-XdLkFoU46PxcbrFhsVeg_-Xoa7_s68ArbIsa7",
            "kid":"g_1"
         },
         {
            "kty":"EC",
            "crv":"BLS12-381",
            "x":"k-ArYFJxn2B9rNOgiCdPZVlr0NCZILYatdphu9x_UEkzTPESE5RdV-WsfQVdBCt-
AkqisvCPCpEmCAUnLcUQUcbketT6QDsCtFELZHrj0XcLrAMmqAW779SAVsjBIb24",
            "kid":"g_2"
         },
         {
            "kty":"EC",
            "crv":"BLS12-381",
            "x":"pSqZnHHwO5hhshpuBZHN9Tj9380nn68CuKZ2ZIsImXFvNqBH0a6t5IK7tRtYdw8U",
            "kid":"g_1^a"
         },
         {
            "kty":"EC",
            "crv":"BLS12-381",
            "x":[
               [
                  [
                     "Dkd1IZACkrXkbeUYdaOdZekWgpLL3t2WCulGkjb3wj8CI1ggvyIdTlalAslFSEVK",
                     "CaLTZm7ELLr0o5nEc6n-YwnKrY-eIg63ktUg-jUOnfJVK2FZZ8GS2UdW6WpiowpF"
                  ],
                  [
                     "AHRuXaBkvJPAv6qpcoC7pr2K2uG0pueF4okwatY_mAdSx3LYWyhUAlqMPbTgXroI",
                     "EzdC4cdKBSgTRT_vwoJxgNUZrPfifltQAV58B4EK2yVbr_SJhiEXcOZG_bSodoyV"
                  ]
               ],
               [
                  [
                     "DRm6bpiriusbaS8nKDWZzYLiGfvpr9XbRGP7xM4lOiuDldV0kpxUCTCOcrjDyhTv",
                     "EJKHLoBbqn9g2ViMNI_6FhV77GoIUm7lQgzU4_aAxNpogZdulxx2HMmS_2wx0xq6"
                  ],
                  [
                     "BIrXVuhF78CaT8F19VUjzWYT4npZ1U3qAma7ruzLMI1MgLeIiiH30ov5xlFfYDfb",
                     "AZRfhNKuUjfq--Q6w7hyN10RW3gJish6MeY4MuGj_pYrEgA_J7JXJNdIMTwC8oE0"
                  ]
               ],
               [
                  [
                     "F-U0_oaJIm1af21glzuoZNRO9HsIK95VMMKjKjbfMlmgLJRgfefY32Avo1L6epPs",
                     "E7v0q1cRuocQwmtEbTowG0E5kWHy3MimDaatxhMnLOLwoS9tXUrz0qdmU2D7vkYP"
                  ],
                  [
                     "EuVkk3JOctQAfVTQ5rAwwYD_Y7ybXo_Fv_GFk_bkhFaT7Ycs9127PpnPhFZkGw1N",
                     "BFWJJlS0HiPKsVs2onD2r9Yi45TtWQqoHcSqfYKKYS2200KiU2vE2koeQ4pZwu58"
                  ]
               ]
            ],
            "kid":"e(g_1,g_2)^\\alpha"
         },
         {
            "kty":"EC",
            "crv":"BLS-12-381",
            "x":"pCWSh5v9KXv46isfWAg83SkVl7_B-4K68prY3ukopKRBJR-k1PDri2rDZCZsrqYh",
            "kid":"H(A)"
         },
         {
            "kty":"EC",
            "crv":"BLS-12-381",
            "x":"hj2lzvfcYevX9RWMU2zZottNnBqQz1JbXgDPv5ZPZmhNfUAM-pEXy3wWHc40QwH1",
            "kid":"H(B)"
         },
         {
            "kty":"EC",
            "crv":"BLS-12-381",
            "x":"gVJ__Y-3op8jhvaGBBEPb0JDkp49dF2kmjlz3xrVQmWh3NTEuocC9uCAR7-jrnoK",
            "kid":"H(C)"
         },
         {
            "kty":"EC",
            "crv":"BLS-12-381",
            "x":"iRYg8UgCPuLExMwspRuxqJnmABCz68iJTpweKp5j0A-CDtUrVpET9rjdzZD_3jk_",
            "kid":"H(D)"
         }
```

```
        ]
      }
   ]
}
```

6 Interface Implementing Reference Point R

6.1 Introduction

While the previous clause defined the format of verifiable credentials and presentations, together with a method to retrieve public parameters, implementing reference point K and P, the present clause focuses on the format of data at the Reference Point R.

At Reference Point R, the Verifier provides the Prover with a challenge, i.e. a CP-ABE encrypted secret that shall be decrypted. The Prover's response is the decrypted secret (see the protocol illustrated in clause 4.2).

This clause is intended to provide a standard format for the challenge and for the response appearing in the challenge-response protocol.

6.2 Challenge

6.2.1 Overview

The challenge **shall** be a cyphertext encoded via JSON Web Encryption (JWE) compact serialization [3], which for reference, is below reported. This serialization format is made of five Base64 URL safe encoded string segments concatenated by the dot "." character.

The present document uses three out of five "segments": the JWE Protected Header, the JWE Encrypted Key, the JWE Ciphertext. The JWE Initialization Vector, and the JWE Authentication Tag (reported as deleted statements below) are not used:

```
BASE64URL(UTF8(JWE Protected Header)) || '.' ||
BASE64URL(JWE Encrypted Key) || '.' ||
~~BASE64URL(JWE Initialization Vector) || '.' ||~~
BASE64URL(JWE Ciphertext) || '.' ||
~~BASE64URL(JWE Authentication Tag)~~
```

NOTE: The name "Protected Header" derives from historical reasons since, in the original JWE specifications, parameters inside this segment are used as "additional authenticated data" of the authenticated encryption algorithm defined in the original JWE specifications:

```
additional authenticated data = ASCII(BASE64URL(UTF8(JWE Protected Header)))
```

The "additional authenticated data" is passed to the authenticated encryption algorithm to obtain the "JWE Authentication Tag". As ABE schemas use their own authenticated encryption, no "Authentication Tag" is used in the present document.

6.2.2 JWE Protected Header

The "JWE Protected Header" object **shall** contain the following two entries defined by the JWE standard [3]:

- The algorithm used to provide key encapsulation **shall** appear as value of the "alg" property.
- The algorithm used to encrypt the ciphertext **shall** appear as value of the "enc" property.

In addition, the present document defines a new property "query" as follows:

- The presentation policy used for encryption **shall** appear as value of the "query" property:

```
{"query":<cp-abe-policy>}
```

To build the final JWE compact serialized object, the "JWE Protected Header" object **shall** be encoded as BASE64URL(UTF8(JWE Protected Header)), where BASE64URL() denotes the Base64URL encoding and UTF8() denotes the octets encoding the UTF-8 [RFC3629] representation of any string containing a sequence of zero or more Unicode characters.

NOTE: For the purposes of the present document, the "alg" property value matches the value **`CP-WATERS-KEM`** and the "enc" property value matches the value **`CP-WATERS-ABE`**, thus referring to the construction reported in, respectively, clauses A.2 and A.3. In addition, the "query" property value appearing in the JWE Protected Header matches the access policy string generated by the encrypting party in the CCA-secure Encryption algorithm (clause A.3.1).

6.2.3 JWE Encrypted Key

The "JWE Encrypted Key" object represents components of the encapsulated key generated by the CP-WATERS-KEM algorithm.

These components are points over elliptic curves. The syntax presented in [2] **shall** be used to represent these components.

To build the final JWE compact serialized object, the "JWE Encrypted Key" object **shall** be encoded as BASE64URL(JWE Encrypted Key).

6.2.4 JWE Ciphertext

The "JWE Ciphertext" object represent the final ciphertext produced by the CP-WATERS-ABE algorithm, which **shall** be plain bit string made of two components: the nonce and the randomness used in the challenge-response protocol.

To build the final JWE compact serialized object, the "JWE Ciphertext" string **shall** be encoded as BASE64URL(JWE Ciphertext).

6.2.5 Example: JWE Protected Header

According to clause 4.1.3, the presentation policy in the example shown in the present document is:

```
A AND (D OR (B AND C))
```

Therefore, the JSON representation inside the JWE Protected Header is:

```
{
   "alg":"CP-WATERS-KEM",
   "enc":"CP-WATERS-ABE",
   "query":"A AND (D OR (B AND C))"
}
```

Encoding using Base64URL without padding gives:

```
eyJhbGciOiJDUC1XQVRFUlMtS0VNIiwiZW5jIjoiQ1AtV0FURVJTLUFCRSIsInF1ZXJ5IjoiQSBBTkQgKEQgT1IgKEIgQU5EIEMp
KSJ9
```

6.2.6 Example: JWE Encrypted Key

In the running example presented throughout the present document, `K` and `r` are 128 bit strings, defined as bytes in hexadecimal notations as follows:

```
K=02x000102030405060708090a0b0c0d0e0f
r=02x101112131415161718191a1b1c1d1e1f
```

Concatenating **`K`**, **`r`** and the presentation policy (the presentation policy in the example is **`A AND (D OR (B AND C))`**) in a single byte array gives:

```
K||r||policy=02x000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f4120414e44202844204f
5220284220414e4420432929
```

Hashing using SHA256 gives:

```
HASH256(K||r||policy):02xc79257ec424db354115569bdcee5cacaf9529cc6593899c8c8d7f2f8312a6ce0
```

This seed, known as **u**, is used to initialize the pseudo random generator (see clause A.3.1).

Given the pseudo random generated values:

```
s:02x25c05d2f19c8cc490dfbfafa82d9745b3cf6b6619641cea1b3e9c5fbf56ddcf0
lambda1:02x92329ecf05ad3a8fb548f1085cf4c3d38aedebb77c2fb3f2c6d09354df9ac401
lambda2:02x51034784a9c43b6f2942e62b7fce0c4af3036d258c6ed494cb55bb5f1120f1de
lambda3:02x2a65c681bd9250da95e493d6b95a54476670a58a8d9ffe1821c3774604b22713
lambda4:02x77b66b33db90f018bece1fa2f86888d05c66ead1a1076aded1932a615d318f0
r1:02x38963f822119730ebceab3cc6ec008ef0ebf1942b5a530a5a7e11eef7423cd9c
r2:02x5ac4661c0879cf2f8c0f7d629f53b2ad7c5d8bdea851f056d096cfd462fd7238
r3:02x60d5327364706a2c4004902b4c631fa4a9d8110da00b5d0e01bb6af0d5eec817
r4:02x21ccefedd32bb4887a58b40556b75262875ae19e6a0a327c8b7625085867f76e
```

and the following LSSS matrix (obtained by compiling the presentation policy using the simple compiler presented in Annex C):

A	1	1	0
B	0	-1	1
C	0	0	-1
D	0	-1	0

the final ciphertext components:

$$C' = {g_1}^{s}$$

$$C_k = {g_1}^{a\lambda_k} H(\rho_k)^{-r_k}$$

$$D_k = {g_2}^{r_k}$$

are represented as compressed points over elliptic curves E_1 and E_2 (refer to Annex B for more details about the serialization format used to represent any point on the two elliptic curves).

```
{
   "ciphertext":[
      {
         "kty":"EC",
         "crv":"BLS12-381",
         "x":"htimHzBHB4lywoep0Rfr6RDpEtaQSo_8KkwcWM102FsfvTluiF3q89G3-5eONrrQ",
         "kid":"C^\\prime=g_1^s"
      },
      {
         "kty":"EC",
         "crv":"BLS12-381",
         "x":"kpE-bqeIjXjjtgeA0iV4OSlsRYU8lATp-L6spVd43m6Q3RJQOO7cldYuHcW1Yoep",
         "kid":"C_1=g_1^{a*\\lambda_1}*H(\rho(1))^{-r_1}"
      },
      {
         "kty":"EC",
         "crv":"BLS12-381",
         "x":"iVMSL2LQ4haEgyuL6buhqZGpPGerBE-HjgoPy7BO-V8BM5QJVx9Syu5-q8Fr3NYG",
         "kid":"C_2=g_1^{a*\\lambda_2}*H(\rho(2))^{-r_2}"
      },
      {
         "kty":"EC",
         "crv":"BLS12-381",
         "x":"gKKVfGsudf1VRz0dSN80EqKxlg6To9Vn8Hz6lr7dtbj06V6Y1X4wEn8Ute1Axa3_",
         "kid":"C_3=g_1^{a*\\lambda_3}*H(\rho(3))^{-r_3}"
      },
      {
         "kty":"EC",
         "crv":"BLS12-381",
         "x":"qAnHeUkqBOKHZy-eBuq4ed0ucWRFDMEY2OKh-u_JUQhwXh1URJxfvtebOZ2kVBwQ",
         "kid":"C_4=g_1^{a*\\lambda_4}*H(\rho(4))^{-r_4}"
      },
      {
         "kty":"EC",
         "crv":"BLS12-381",
```

```
      "x":"tfQ7YsBwq-
Trm8KWKrhxUOfu4bJsq51c19HOFCXIWHdciz2ZnWaslGW3GnqS91uqESM2JS0aFGywA2dLatkEvie9_FHc_C2ruMsIVj6DpZ_p7W
TpkAcMNh-e6wvBdfS-",
         "kid":"D_1=g_2^r1"
      },
      {
         "kty":"EC",
         "crv":"BLS12-381",

"x":"tteZ3MjOaPc2oV9r17mJV9Zbe4i0ifO6xt4kcat7GgkJu1R5dXztx9AlxiRC4BS4DVi1OfusHqZNKdG2m0HMgDFNledpi3s
sXqtrr6QOzzfHeF4IhHw8ZQyWlQwFxRkT",
         "kid":"D_2=g_2^r2"
      },
      {
         "kty":"EC",
         "crv":"BLS12-381",
         "x":"mcoNUQZB9qc-JQB46M7R8t00qLrmVtzB1yeJpBDJ-XhXu6ZeF8-
hjjlloYEmRIZkDzl2TjEpVz_DAThNXkz0Kv9qFP63YlDn-e5FuNtAMjog4ZaVhjvTyUgfeMQgZeft",
         "kid":"D_3=g_2^r3"
      },
      {
         "kty":"EC",
         "crv":"BLS12-381",
         "x":"jxIZ2_z2dhrUXqd5Aa_B1ni3VQeXK5psMCwfj6FKl3exb66YX-
cVuM1mEGjkcIuvB80bWRXNlhqMvbOUY_GeVDLsb5RVznCx5hAfEmHa78C-L3mMgxK6XfHAVLJ3kj-L",
         "kid":"D_4=g_2^r4"
      }
   ]
}
```


Encoding the whole JSON object to Base64URL without padding gives:


```
eyJjaXBoZXJ0ZXh0IjpbeyJrdHkiOiJFQyIsImNydiI6IkJMUzEyLTM4MSIsIngiOiJodGltSHpCSEI0bHl3b2VwMFJmcjZSRHBF
dGFRU29fOEtrd2NXTTEwMkZzZnZUbHVpRjNxODlHMy01ZU9OcnJRIiwia2lkIjoiQ15cXHByaW1lPWdfMV5zIn0seyJrdHkiOiJF
QyIsImNydiI6IkJMUzEyLTM4MSIsIngiOiJrcEUtYnFlSWpYamp0Z2VBMGlWNE9TbHNSWVU4bEFUcC1MNnNwVmQ0M202UTNSSlFP
TzdjbGRZdUhjVzFZb2VwIiwia2lkIjoiQ18xPWdfMV57YSpcXGxhbWJkYV8xfSpIKFxyaG8oMSkpXnstcl8xfSJ9LHsia3R5Ijoi
RUMiLCJjcnYiOiJCTFMxMi0zODEiLCJ4IjoiaVZNU0wyTFE0aGFFZ3l1TDZidWhxWkdwUEdlckJFLUhqZ29QeTdCTy1WOEJNNVFK
Vng5U3l1NS1xOEZyM05ZRyIsImtpZCI6IkNfMj1nXzFee2EqXFxsYW1iZGFfMn0qSChccmhvKDIpKV57LXJfMn0ifSx7Imt0eSI6
IkVDIiwiY3J2IjoiQkxTMTItMzgxIiwieCI6ImdLS1ZmR3N1ZGYxVlJ6MGRTTjgwRXFLeGxnNlRvOVZuOEh6NmxyN2R0YmowNlY2
WTFYNHdFbjhVdGUxQXhhM18iLCJraWQiOiJDXzM9Z18xXnthKlxcbGFtYmRhXzN9KkgoXHJobygzKSleey1yXzN9In0seyJrdHki
OiJFQyIsImNydiI6IkJMUzEyLTM4MSIsIngiOiJxQW5IZVVrcUJPS0haeS1lQnVxNGVkMHVjV1JGRE1FWTJPS2gtdV9KVVFod1ho
MVVSSnhmdnRlYk9aMmtWQndRIiwia2lkIjoiQ180PWdfMV57YSpcXGxhbWJkYV80fSpIKFxyaG8oNCkpXnstcl80fSJ9LHsia3R5
IjoiRUMiLCJjcnYiOiJCTFMxMi0zODEiLCJ4IjoidGZRN1lzQndxLVRybThLV0tyaHhVT2Z1NGJKc3E1MWMxOUhPRkNYSVdIZGNp
ejJabldhc2xHVzNHbnFTOTF1cUVTTTJKUzBhRkd5d0EyZExhdGtFdmllOV9GSGNfQzJydU1zSVZqNkRwWl9wN1dUcGtBY01OaC1l
Nnd2QmRmUy0iLCJraWQiOiJEXzE9Z18yXnIxIn0seyJrdHkiOiJFQyIsImNydiI6IkJMUzEyLTM4MSIsIngiOiJ0dGVaM01qT2FQ
YzJvVjlyMTdtSlY5WmJlNGkwaWZPNnh0NGtjYXQ3R2drSnUxUjVkWHp0eDlBbHhpUkM0QlM0RFZpMU9mdXNIcVpOS2RHMm0wSE1n
REZObGVkcGkzc3NYcXRycjZRT3p6ZkhlRjRJaEh3OFpReVdsUXdGeFJrVCIsImtpZCI6IkRfMj1nXzJecjIifSx7Imt0eSI6IkVD
IiwiY3J2IjoiQkxTMTItMzgxIiwieCI6Im1jb05VUVpCOXFjLUpRQjQ2TTdSOHQwMHFMcm1WdHpCMXllSnBCREotWGhYdTZaZUY4
LWhqamxsb1lFbVJJWmtEemwyVGpFcFZ6X0RBVGhOWGt6MEt2OXFGUDYzWWxEbi1lNUZ1TnRBTWpvZzRaYVZoanZUeVVnZmVNUWda
ZWZ0Iiwia2lkIjoiRF8zPWdfMl5yMyJ9LHsia3R5IjoiRUMiLCJjcnYiOiJCTFMxMi0zODEiLCJ4IjoianhJWjJfejJkaHJVWHFk
NUFhX0IxbmkzVlFlWEs1cHNNQ3dmajZGS2wzZXhiNjZZWC1jVnVNMW1FR2prY0l1dkI4MGJXUlhObGhxTXZiT1VZX0dlVkRMc2I1
UlZ6bkN4NWhBZkVtSGE3OEMtTDNtTWd4SzZYZkhBVkxKM2tqLUwiLCJraWQiOiJEXzQ9Z18yXnI0In1dfQ
```


6.2.7 Example: JWE Ciphertext and JWE object

In the running example presented throughout the present document, the element $e(g,g)^{\alpha s}$ is a group field element that is represented in towered format:


```
[[[13a3bd4f132b2010dcb1818deed44e8c0ce0f24f9939a4fc1087c030d30d8b70c04f2ec675eddcf2191d7ca2e84372fa,
0b828dcd917be9e80c0245a07851a045afe4b4f9a2a1383adb5212ced57ad29a5ee7e5ef61c12c9438c374f4097823dc],[1
330ffb482dea1c4fe2ec7aefb0856bfbb80ae81a306d11e500f2946229ccf087c1a70a45d4aca534651f37952968555,018e
00b4fc2601808ee8cea338139e3365ab35e9daaf0d189e4cd14850bc68d63953682cd2ee901fe357e7906cdf938e]],[[07b
141a211d3288ef93a0dea859ba02c70afab4a5daf7d296af121b607b0acff72e7cec2a9db5ca2e073cb7d8ad7b567,0bbd47
02a3009fa8f08b525549d78a352cb019c000a9b72604dd43a83e617795ec8febb5953dc1e4929bb7f6d3b58bd9],[05eddf0
d57f761b309a30c354b6413880dd0de7a000b1f6c9fbabd430bbe6722dfda1532e752bf792c7b952ebe013a8c,181cff89fb
04f53f03e5d5000bd2faeb39b3995b916bc347a6ad931e01bd1a045a03ae21f61daf9f9d0d170fbaf2a6bf]],[[01da0cc71
f2b71dd20c61990d3d30e212e618c1d73be6664ec28540a78e69cea2de6ebedfc6422bb0372593e7cef7a14,0165bf5a86b9
5bb12cd8c0ce2a81276bd05efddb636737b360d2e9f8966501dde2478505c9c5c5a52ad79e0a8cabfe50],[0f71180460df8
f7c5334401d705ff268d7c45d111fd51a3a65b9ff67c1082e82ff981a486ff1a564ea63b783fcadc1da,15f2e1056fdef784
9a24fec7022bd0d1616333bed25db51c60ccc0a3a543a5f6434f76203f552827c9b79cac16714ba0]]]
```

Walking through the towered representation and concatenating each array element encoded in hexadecimal notation (see clause B.6), gives:

```
13a3bd4f132b2010dcb1818deed44e8c0ce0f24f9939a4fc1087c030d30d8b70c04f2ec675eddcf2191d7ca2e84372fa,0b8
28dcd917be9e80c0245a07851a045afe4b4f9a2a1383adb5212ced57ad29a5ee7e5ef61c12c9438c374f4097823dc,1330ff
b482dea1c4fe2ec7aefb0856bfbb80ae81a306d11e500f2946229ccf087c1a70a45d4aca534651f37952968555,018e00b4f
c2601808ee8cea338139e3365ab35e9daaf0d189e4cd14850bc68d63953682cd2ee901fe357e7906cdf938e,07b141a211d3
288ef93a0dea859ba02c70afab4a5daf7d296af121b607b0acff72e7cec2a9db5ca2e073cb7d8ad7b567,0bbd4702a3009fa
8f08b525549d78a352cb019c000a9b72604dd43a83e617795ec8febb5953dc1e4929bb7f6d3b58bd9,05eddf0d57f761b309
a30c354b6413880dd0de7a000b1f6c9fbabd430bbe6722dfda1532e752bf792c7b952ebe013a8c,181cff89fb04f53f03e5d
5000bd2faeb39b3995b916bc347a6ad931e01bd1a045a03ae21f61daf9f9d0d170fbaf2a6bf,01da0cc71f2b71dd20c61990
d3d30e212e618c1d73be6664ec28540a78e69cea2de6ebedfc6422bb0372593e7cef7a14,0165bf5a86b95bb12cd8c0ce2a8
1276bd05efddb636737b360d2e9f8966501dde2478505c9c5c5a52ad79e0a8cabfe50,0f71180460df8f7c5334401d705ff2
68d7c45d111fd51a3a65b9ff67c1082e82ff981a486ff1a564ea63b783fcadc1da,15f2e1056fdef7849a24fec7022bd0d16
16333bed25db51c60ccc0a3a543a5f6434f76203f552827c9b79cac16714ba0
```

Where commas have been used to keep separated each element. Removing the commas and using the value resulting from the concatenation of elements to feed the SHA256 algorithm gives:

```
y=02x09b1266b67f42cfdfa2132e1821607209a5428d3319b4ae226d132472c93c2c4
```

The final encrypted (K||r) $\oplus$ y is given by:

```
K||r ⊕ y = 02x09b0246863f12afaf22838ea8e1b092f8a453ac0258e5cf53ec8285c308edcdb
```

Which is converted in the following Base64URL representation:

```
CbAkaGPxKvryKDjqjhsJL4pFOsAljlz1PsgoXDCO3Ns
```

Where trailing padding has been omitted. The final JSON Web Encryption (JWE) object in its compact form is:

```
eyJhbGciOiJDUC1XQVRFUlMtS0VNIiwiZW5jIjoiQ1AtV0FURVJTLUFCRSIsInF1ZXJ5IjoiQSBBTkQgKEQgT1IgKEIgQU5EIEMp
KSJ9.eyJjaXBoZXJ0ZXh0IjpbeyJrdHkiOiJFQyIsImNydiI6IkJMUzEyLTM4MSIsIngiOiJodGltSHpCSEI0bHl3b2VwMFJmcjZ
SRHBFdGFRU29fOEtrd2NXTTEwMkZzZnZUbHVpRjNxODlHMy01ZU9OcnJRIiwia2lkIjoiQ15cXHByaW1lPWdfMV5zIn0seyJrdHk
iOiJFQyIsImNydiI6IkJMUzEyLTM4MSIsIngiOiJrcEUtYnFlSWpYamp0Z2VBMGlWNE9TbHNSWVU4bEFUcC1MNnNwVmQ0M202UTN
SSlFPTzdjbGRZdUhjVzFZb2VwIiwia2lkIjoiQ18xPWdfMV57YSpcXGxhbWJkYV8xfSpIKFxyaG8oMSkpXnstcl8xfSJ9LHsia3R
5IjoiRUMiLCJjcnYiOiJCTFMxMi0zODEiLCJ4IjoiaVZNU0wyTFE0aGFFZ3l1TDZidWhxWkdwUEdlckJFLUhqZ29QeTdCTy1WOEJ
NNVFKVng5U3l1NS1xOEZyM05ZRyIsImtpZCI6IkNfMj1nXzFee2EqXFxsYW1iZGFfMn0qSChccmhvKDIpKV57LXJfMn0ifSx7Imt
0eSI6IkVDIiwiY3J2IjoiQkxTMTItMzgxIiwieCI6ImdLS1ZmR3N1ZGYxVlJ6MGRTTjgwRXFLeGxnNlRvOVZuOEh6NmxyN2R0Ymo
wNlY2WTFYNHdFbjhVdGUxQXhhM18iLCJraWQiOiJDXzM9Z18xXnthKlxcbGFtYmRhXzN9KkgoXHJobygzKSleey1yXzN9In0seyJ
rdHkiOiJFQyIsImNydiI6IkJMUzEyLTM4MSIsIngiOiJxQW5IZVVrcUJPS0haeS1lQnVxNGVkMHVjV1JGRE1FWTJPS2gtdV9KVVF
od1hoMVVSSnhmdnRlYk9aMmtWQndRIiwia2lkIjoiQ180PWdfMV57YSpcXGxhbWJkYV80fSpIKFxyaG8oNCkpXnstcl80fSJ9LHs
ia3R5IjoiRUMiLCJjcnYiOiJCTFMxMi0zODEiLCJ4IjoidGZRN1lzQndxLVRybThLV0tyaHhVT2Z1NGJKc3E1MWMxOUhPRkNYSVd
IZGNpejJabldhc2xHVzNHbnFTOTF1cUVTTTJKUzBhRkd5d0EyZExhdGtFdmllOV9GSGNfQzJydU1zSVZqNkRwWl9wN1dUcGtBY01
OaC1lNnd2QmRmUy0iLCJraWQiOiJEXzE9Z18yXnIxIn0seyJrdHkiOiJFQyIsImNydiI6IkJMUzEyLTM4MSIsIngiOiJ0dGVaM01
qT2FQYzJvVjlyMTdtSlY5WmJlNGkwaWZPNnh0NGtjYXQ3R2drSnUxUjVkWHp0eDlBbHhpUkM0QlM0RFZpMU9mdXNIcVpOS2RHMm0
wSE1nREZObGVkcGkzc3NYcXRycjZRT3p6ZkhlRjRJaEh3OFpReVdsUXdGeFJrVCIsImtpZCI6IkRfMj1nXzJecjIifSx7Imt0eSI
6IkVDIiwiY3J2IjoiQkxTMTItMzgxIiwieCI6Im1jb05VUVpCOXFjLUpRQjQ2TTdSOHQwMHFMcm1WdHpCMXllSnBCREotWGhYdTZ
aZUY4LWhqamxsb1lFbVJJWmtEemwyVGpFcFZ6X0RBVGhOWGt6MEt2OXFGUDYzWWxEbi1lNUZ1TnRBTWpvZzRaYVZoanZUeVVnZmV
NUWdaZWZ0Iiwia2lkIjoiRF8zPWdfMl5yMyJ9LHsia3R5IjoiRUMiLCJjcnYiOiJCTFMxMi0zODEiLCJ4IjoianhJWjJfejJkaHJ
VWHFkNUFhX0IxbmkzVlFlWEs1cHNNQ3dmajZGS2wzZXhiNjZZWC1jVnVNMW1FR2prY0l1dkI4MGJXUlhObGhxTXZiT1VZX0dlVkR
Mc2I1UlZ6bkN4NWhBZkVtSGE3OEMtTDNtTWd4SzZYZkhBVkxKM2tqLUwiLCJraWQiOiJEXzQ9Z18yXnI0In1dfQ..CbAkaGPxKvr
yKDjqjhsJL4pFOsAljlz1PsgoXDCO3Ns.
```

Where three out of five segments have been used.

6.3 Response

6.3.1 Overview

The Prover's response is implemented using a Verifiable Presentation proof. The definition of the proof **shall** be the same as for Verifiable Credentials reported in clause 5.2, except for the property "proofValue".

The "proofValue" property **shall** present a Base64URL string (refer to clause B.5) encoding the value matching the response (i.e. the decrypted secret) of the challenge-response protocol presented in clause 4.2.

Note that, contrary to signature-based verifiable presentation, a proof in Functional credentials does not need a specific "nonce" as a protection from reply attacks. In fact, due to the randomness used to generate the challenge, each challenge (and each response) is unique in time.

6.3.2 Example: Verifiable Presentation

Continuing the running example, the Prover responds to the challenge received by the Verifier with the following proof embedded in a Verifiable Presentation.

```
{
  "proof": {
      "type":"FunctionalCredentialPresentation_2023_CP_WATERS_KEM",
      "created": "2023-01-01T00:00:01Z",
      "proofValue": "uAAECAwQFBgcICQoLDA0ODw",
      "proofPurpose": ["capabilityInvocations"],
      "verificationMethod": "https://www.example.org/vc/cp-abe/v1/public-parameters#1"
    }
}
```

Note the prepended character header "u" (multi-base encoding) in the "proofValue" and the following encoding of the value of *K* defined in clause 6.2.6 via Base64URL no padding.

Annex A (informative): Attribute Based Encryption

A.1 CP-ABE schema

Traditional public-key encryption uses a public key to target ciphertexts to a specific user. The user holds a secret key and can decrypt the message. In predicate encryption, and Attribute Based Encryption in particular, instead, ciphertexts are not necessarily encrypted to one particular user. Instead, both users' secret keys and ciphertexts can be associated with a set of attributes or a policy over attributes. A user is able to decrypt a ciphertext if there is a "match" between his secret key and the ciphertext.

This feature enables a party to encrypt data even without knowing a priori the identity of each individual user which needs to access the data. ABE can be efficiently used whenever ciphertexts have to be targeted to a class of users rather than a single user, because encryption is performed once.

In particular, in most Ciphertext Policy-Attribute Based Encryption (CP-ABE) schemes a monotonic tree access structure representing a policy is encoded into the cyphertext, while the user's secret keys are computed with respect to a set of attributes S the user has obtained.

Attributes are assigned to users and the access structures are used to label different sets of encrypted data. A user is able to decrypt the ciphertext with a given key if and only if there is an assignment of attributes from the secret key to nodes of the tree such that the tree is satisfied.

The underlying mathematical construction is based on a secret sharing scheme embedded in the ciphertext that prevents collusion attacks, i.e. attacks from two or more different users that have obtained keys encoding different sets of attributes that, by their own, would not satisfy the access tree structure, but whose union would do.

The decryption algorithm works by masquerading original values from each user's secret key in a randomized fashion. Thus, the key components associated to the same attribute in two different secret keys are not the same and cannot be interchanged in order to decrypt a cyphertext.

The scheme consists of four algorithms:

- *Setup(l) -> MSK, MPK*: Takes global parameters l as input and outputs the public parameters *MPK* and a master key *MSK*.
- *Key-Gen(MSK, S) -> SK*: Takes as input the master secret key *MSK* and a set of attributes S that describe the key and outputs a secret key *SK*.
- *Encrypt(MPK, M, A) -> CT*: Takes as input the public parameters *MPK*, a message *M*, and an access structure *A* over the universe of attributes (that represents the policy to be satisfied to access the message). The algorithm encrypts *M* and produces a ciphertext *CT* such that only a user that possesses a set of attributes that satisfies the access structure will be able to decrypt the message. The ciphertext implicitly contains *A*.
- *Decrypt(CT, SK)*: Takes as input the ciphertext *CT* (which contains the access structure *A*), and a secret key *SK*, which is a secret key for a set *S* of attributes. If the set *S* of attributes satisfies the access structure *A* then the algorithm will decrypt the ciphertext and return a message *M*.

A.2 CP-WATERS-KEM construction

The present construction is reported in [i.4]. It is adapted in clause 4.2.2 of ETSI TS 103 532 [6] with only few changes in used symbols.

Let G_1, G_2 and G_T be two multiplicative cyclic groups of prime order. Let g_1 and g_2 be generators of G_1, G_2 and e be a bilinear map, $e: G_1 \times G_2 \rightarrow G_T$. The bilinear map e has the following properties:

1) Bilinearity: for all $u, v \in G$ and $a, b \in Z_p$, the equality $e(u^a,v^b) = e(u^b,v^a) = e(u,v)^{ab}$ holds.

2) Non-degeneracy: $e(g_1, g_2) \neq 1$.

If the group operation in G_1 and G_2 and the bilinear map e are both efficiently computable, G_1 and G_2 are said bilinear groups.

CP-WATERS-ABE consists of four algorithms:

1) *Setup()* → *{MPK, MSK}*: The algorithm outputs the master secret key *MSK* and the master public key *MPK*, and publishes the *MPK*.

- The algorithm chooses bilinear groups G_1 and G_2 and random exponents α (alpha), $a \in Z_p$.
- The public key is $MPK = \{g_1, g_2, g_1^a, e(g_1, g_2)^\alpha\}$ and the master secret key is $MSK = \{g^\alpha, \alpha, a\}$.
- A function $H: \{0, 1\}^* \rightarrow G_1$ modelled as a random oracle hashes any attributes $x \in \{0, 1\}^*$ and produce attribute parameters on request.

2) *Key-Gen(MPK,MSK, S)* → *secret key*: Key generation happens by taking as input the master secret key *MSK* and a set of attributes *S* that describe the key. The output is a randomized secret decryption key.

- Chosen a random $t \in Z_p$ the algorithm simply generates and releases: $K = g_2^\alpha g_2^{at}$, $L = g_2^t$ and, for each $x \in S$ $K_x = H(x)^t$ as the secret decryption key.

3) *Encrypt(MPK, (M; ρ))* → *{random secret, ciphertext}:* The algorithm takes as input an access structure *(M; ρ)* the public key *MPK*. *M* is an *l x m* matrix, while ρ is an injective function associating each row of *M* to an attribute ρ_k (i.e. $\rho_k = \rho(k) \in S$); note that in this construct one attribute is associated with at most one row ([i.4] proposes off-the-shelf techniques to cope with this limitation). The output is a random secret and the ciphertext.

- Chosen a random vector $\vec{v} = (s, y_2, \ldots, y_m)$ in $(Z_p)^m$ and being M_k the k-th row of *M*, the algorithm computes $\lambda_k = \vec{v} M_k$. In addition, the algorithm chooses random $r_1, \ldots, r_l \in Z_p$.

NOTE: In order to use pseudo-randomness, the algorithm can take as input an optional input seed $u \in \{0, 1\}^k$ to a pseudo-random generator PRG. This feature is later used (see clause A.3) to transform the ABKEM schema into a hybrid CCA-IND2 encryption schema.

- Together with a with the access structure *(M; ρ)*, the algorithm makes public the ciphertext:

$$C' = g_1^{\ s}$$
$$C_k = g_1^{\ a\lambda_k} H(\rho_k)^{-r_k}$$
$$D_k = g_2^{\ r_k}$$

- The algorithm computes the random secret $e(g_1,g_2)^{\alpha s}$ and keeps it private.

4) *Decrypt(SK, C)* → *random secret:* Dually, the decryption takes as input the ciphertext *C* generated at step 3 and the secret key *SK* generated at step 2. The output is the common secret $e(g_1, g_2)^{\alpha s}$ if and only if the set of attributes *S* satisfies the access structure, null otherwise.

- For each *k* such that ρ_k is in *S* (i.e. consider only attributes in S), compute ω_k such that $\Sigma_k \omega_k \lambda_k = s$ (there could be different sets of $\{\omega_k\}$ satisfying this equation).
- Compute the random secret:

$$\frac{e(C', K)}{\prod_k [e(C_k, L) e(K_{\rho_k}, D_k)]^{\omega_k}} = e(g_1, g_2)^{\alpha s}$$

A.3 CCA-secure CP-ABE construction

A.3.1 CCA-secure Encryption Algorithm

The present construction is reported in [i.5]. It is adapted in clause 4.5.2 of ETSI TS 103 532 [6] with only few changes in adopted symbols.

The CCA-secure encryption algorithm is specified by the following steps:

- The encrypting party (prover) shall choose a random number r_{chosen}, an access structure AP (as a string) and a nonce K_{chosen} and concatenates them to form the string $r_{chosen}||K_{chosen}||A$.
- The encrypting party runs the encryption algorithm of the original CP-WATERS-KEM to get a random secret and the ciphertext. The seed $r_{chosen}||K_{chosen}||AP$ is used as a source of randomness for CP-WATERS-KEM encryption algorithm with $u = PRG(H'(r_{chosen}||K_{chosen}||AP), l)$, where PRG is a Pseudo Random Generator, l is the length of the returned random bit string ($u \in \{0,1\}^l$) and H' is a collision-resistant hash function.
  - The random secret is $e(g_1, g_2)^{\alpha s}$. Keep it private and use in the next step.
  - Release the ABKEM ciphertext C_{ABKEM}.
- Use random secret above for XORing the concatenation $r_{chosen}||K_{chosen}$.
  - Transform $r_{chosen}||K_{chosen}$ into bytes (octets).
  - Using the Pseudo Random Generator (PRG), get $r = PRG(H(e(g_1, g_2)^{\alpha s}), l)$ for CP-WATERS-KEM or $r = PRG(H((e(acc_V, g_2)^{\alpha} e(g_1{}^{b}, g_2{}^{\gamma^{n+1}}))^{s}), l)$ for the modified CP-WATERS-KEM, with H being a collision-resistant hash function.
  - Finally, compute $C = r \oplus (K_{chosen}||r_{chosen})$.

A.3.2 CCA-secure Decryption Algorithm

The present construction is reported in [i.5]. It is adapted in clause 4.5.2 of ETSI TS 103 532 [6] with only few changes in adopted symbols.

The CCA-secure decryption algorithm is specified by the following steps:

- Run decryption of the original CP-WATERS-KEM or of the modified schema to decrypt the ABKEM ciphertext C_{ABKEM} and to obtain the shared secret: $e(g_1, g_2)^{\alpha s}$ for CP-WATERS-KEM or $(e(acc_V, g_2)^{\alpha} e(g_1{}^{b}, g_2{}^{\gamma^{n+1}}))^{s}$ for the modified CP-WATERS-KEM.
- Use that shared secret to generate randomness $r = PRG(H(e(g_1, g_2)^{\alpha s}), l)$.
- Use generated randomness r for XORing the ciphertext and retrieve K_{chosen} and r_{chosen}: $C \oplus r = (K_{chosen}||r_{chosen})$.
- Verify r_{chosen} matches the random number chosen at beginning.
- Run again the ABKEM encryption as in clause A.3.1 using $r_{chosen}||K_{chosen}||AP$ as a source of randomness and verify the result is equal to the received ciphertext C_{ABKEM}.

Annex B (informative): Pairing friendly BLS12-381 Curve and its Encoding

B.1 BLS12-381 Curve

While IEEE 1363.3-2013 [7] contains useful definitions and conventions that may be used for the implementation of the present document, this latter follows some conventions to match the more recent work done by IETF/W3C® community for the Elliptic Curve Cryptography. Therefore, the present document aligns with [i.2] that suggests that vast majority of current libraries implementing BLS12-381 use the encoding method defined in Annex C of [i.3] (although other encoding exists).

When using BLS12-381 pairing friendly curve, the elliptic curves E_1 and E_2 are defined over the group fields of modulo p: $GF(p)$ and $GF(p^2)$ with:

```
p =
0x1a0111ea397fe69a4b1ba7b6434bacd764774b84f38512bf6730d2a0f6b0f6241eabfffeb153ffffb9feffffffffaaab
```

The first group field is of integers, the second one is an "extension field" of "complex" integers $y' = y'_0 + y'_1 \times i$ group elements with base $(1,i)$ where $i^2 + 1 = 0$.

Two elliptic curves are defined by:

$$E_1 : y^2 = x^3 + 4$$

$$E_2 : y'^2 = x'^3 + 4 \times (i + 1)$$

The group G_1 and G_2 are defined as the order r subgroups of E_1 over $GF(p)$ and E_2 over $GF(p^2)$ respectively, with r prime factor of p:

```
r = 0x73eda753299d7d483339d80809a1d80553bda402fffe5bfeffffffff00000001
```

The generators are:

```
g1:
(0x17f1d3a73197d7942695638c4fa9ac0fc3688c4f9774b905a14e3a3f171bac586c55e83ff97a1aeffb3af00adb22c6bb,
0x08b3f481e3aaa0f1a09e30ed741d8ae4fcf5e095d5d00af600db18cb2c04b3edd03cc744a2888ae40caa232946c5e7e1)
g2:([0x024aa2b2f08f0a91260805272dc51051c6e47ad4fa403b02b4510b647ae3d1770bac0326a805bbefd48056c8c121b
db8,0x13e02b6052719f607dacd3a088274f65596bd0d09920b61ab5da61bbdc7f5049334cf11213945d57e5ac7d055d042b
7e],[0x0ce5d527727d6e118cc9cdc6da2e351aadfd9baa8cbdd3a76d429a695160d12c923ac9cc3baca289e193548608b82
801,0x606c4a02ea734cc32acd2b02bc28b99cb3e287e85a763af267492ab572e99ab3f370d275cec1da1aaa9075ff05f79b
e])
```

A pairing e is defined by taking G_1 as a subgroup of $E(GF(p))$ of order r, G_2 as an order r subgroup of $E'(GF(p^2))$ for BLS12 and G_T as an order r subgroup of the multiplicative group $GF(p^{12})^*$.

B.2 Point encoding with compression

Serialized points include three metadata bits that indicate whether a point is compressed or not, whether a point is the point at infinity or not, and the sign of the point's y-coordinate (ordinate) for point compression.

Point compression refers to the ability to represent a point by only one of its coordinates, x-coordinate (the abscissa), leaving the recipient the task of calculating the other one, y-coordinate (the ordinate), using the curve equation plus an additional information about the sign of the ordinate. Compressed points on E_1 are serialized into 48 bytes, while compressed points on E_2 are serialized into 96 bytes.

Serialization and deserialization algorithms use the following functions:

- The functions *LEFT(str, n)* and *RIGHT(str, n)* respectively return the n leftmost or rightmost bytes from string str.

- The function *sqrt(y)* or *sqrt(y')*, given *y* an element in G_1 or *y'* an element in G_2, returns the square root of element *y* in the respective group, i.e. an element a such that $a^2 = y$, or the error signal INVALID.
- The function $sign_{GF_p}(y)$ defined in [i.3], returns one bit representing the sign of an element of *GF(p)*:
  1) if y > (p - 1) / 2 output 1.
  2) else output *0*.
- The function $sign_{GF_{p^2}}(y') = sign_{GF_{p^2}}(y'_0 + y'_1 \times i)$defined in [i.3], returns one bit representing the sign of an element in $GF(p^2)$:
  1) If y'_1 equals *0*, output *sign_GF_p(y'₀)*.
  2) Else if $y'_1 > (p - 1) / 2$ output *1*.
  3) else output *0*.
- The function *OS2IP(X)*, defined in [1], given an octet string to be converted *X*, converts it to a nonnegative integer:
  1) Let $X = X_1 \| X_2 \| \ldots \| X_{xLen}$ and $x_{(xLen-i)}$ be the integer value of the octet X_i for $1 \leq i \leq xLen$.
  2) Let $x = x_{(xLen-1)} \times 256^{(xLen-1)} + x_{(xLen-2)} \times 256^{(xLen-2)} + \ldots + x_1 \times 256 + x_0$.
  3) Output x.
- The function *I2OSP (x, xLen)*, defined in [1], converts a nonnegative integer x to an octet string of a specified length *xLen<256*:
  1) If $x \geq 256^{xLen}$, output error "integer too large" and stop.
  2) Write the integer *x* in its unique xLen-digit representation in base 256:

     $x = x_{(xLen-1)} \times 256^{(xLen-1)} + x_{(xLen-2)} \times 256^{(xLen-2)} + \ldots + x_1 \times 256 + x_0$

     where $0 \leq x\_i < 256$ (note that one or more leading digits will be zero if x is less than $256^{(xLen-1)}$).
  3) Let the octet X_i have the integer value *x_(xLen-i) for 1 ≤ i ≤ xLen.* Output the octet string:

     $X = X_1 \| X_2 \| \ldots \| X_{xLen}$

B.3 Serialization

1) For any point *P = (x,y)* to be serialized, set:

- C_bit (compression) is set to 1 (indicating that point compression is used).
- I_bit (infinity) is 1 if P is either the point at infinity of E_1 or E_2, otherwise it is 0. The serialization of a point at infinity results in a string of zero bytes, except C_bit and I_bit which are set to 1.
- S_bit (sign) is 0 if I_bit is 1, otherwise if *P* is a point on E_1, set S_bit = $sign_{GF_p}(y)$, else if *P* is a point on E_2, S_bit = $sign_{GF_{p^2}}(y')$.

2) Group the metadata bits into m_byte = (C_bit × 2^7) + (I_bit × 2^6) + (S_bit × 2^5).

3) Let x_string be the serialization of x, which is defined as follows:

- If *P* is the point at infinity on E_1, let x_string = *I2OSP(0, 48)*.
- If *P* is a point on E_1 other than the point at infinity, then *x* is an element of *GF(p)*, i.e. an integer in the inclusive range *[0, p - 1]*. In this case, let x_string = *I2OSP(x, 48)*.
- If *P* is the point at infinity on E_2, let x_string = *I2OSP(0, 96)*.

- If *P* is a point on E_2 other than the point at infinity, then $x' = x_0 + x_1 \times i$ and can be represented as (x_0, x_1) where x_0 and x_1 are elements of *GF(p)*. Let x_string = *I2OSP(x_1, 48)* || *I2OSP(x_0, 48)*.

Notice that in all of the above cases, the 3 most significant bits of x_string[0], i.e. the first byte of x_string, are guaranteed to be *0*.

4) Let s_string[0] = x_string[0] OR m_byte, in order to include the metadata bits.

5) Output x_string.

B.4 Deserialization

1) Let m_byte = s_string[0] & 0xE0, where & is bitwise AND operator.

2) If m_byte equals any of 0x20, 0x60, or 0xE0, output error "INVALID" and stop decoding.

3) Else:

- Let C_bit equal the most significant bit of m_byte;

- Let I_bit equal the second most significant bit of m_byte; and

- Let S_bit equal the third most significant bit of m_byte.

4) If C_bit is 0 output error "INVALID" and stop decoding.

5) If s_string has length 48 bytes, the output point *P(x,y)* is on the curve E_1; else if s_string has length 96 bytes, the output point *P(x',y')* is on the curve E_2; else output error "INVALID" and stop decoding.

6) Let s_string[0] = s_string[0] & 0x1F.

7) If I_bit is 1 then:

- If s_string == *0* (s_string is the all zeros string), output the point at infinity and stop decoding.

- else output error "INVALID" and stop decoding.

8) Let x = OS2IP(s_string) or x' = (x_0, x_1) = (OS2IP(RIGHT(s_string),48), OS2IP((LEFT(s_string,48))) depending P lays on E_1 or E_2 (see step 5).

9) To determine the y-coordinate (ordinate), compute y^2 or y'^2 using equation E_1 or E_2 depending whether point P lays on the first or the latter curve (see step 5):

- If the computed value is not a square, output error "INVALID" and stop decoding.

- Else, if the point is on E_1 set Y_bit = $sign_{GF_p}(sqrt(y^2))$.

- Else, if the point is on E_2 set Y_bit = $sign_{GF_{p^2}}(sqrt(y'^2))$.

10) If S_bit equals Y_bit, output *P = (x, y)* in E_1 or *P = (x',y')* in E_2. Otherwise, output *P = (x, -y)* in E_1 or *P = (x',-y')* in E_2.

B.5 Base64URL Encoding and Decoding

Conforming to [4], when appearing as values inside JSON objects, EC points and group field elements are further processed using Base64:

- EC points are first encoded as per the above clauses, then Base64 encoding as below is applied.
- Group field elements are converted to byte arrays using the *I2OSP(x, 48)* function ([1]), then Base64 encoding as below is applied.

The Base64 encoding specified in [4] uses the URL- and filename-safe character set defined in Section 5 of [5], with all trailing '=' characters omitted and without the inclusion of any line breaks, whitespace, or other additional characters. The Base64URL encoding of the empty octet sequence is the empty string.

B.6 Encoding elements of the multiplicative group $GF(p^{12})^*$

Elements of the multiplicative group $GF(p^{12})^*$ are polynomials of degree 12 represented through their 12 coefficients. However, they may be conveniently represented in the so called "towered" format as an array of three arrays of two arrays of two group field elements GF(p). For the purposes of the present document, this is the preferred format. The following two encodings are used:

- When appearing inside of JSON objects, the towered format is mapped directly into JSON array of arrays. Non-array (i.e. group field) elements are encoded using the Base64 encoding specified in [4] (which uses the URL- and filename-safe character set defined in Section 5 of [5], with all trailing '=' characters omitted and without the inclusion of any line breaks, whitespace, or other additional characters).

- When appearing in binary objects, each non-array element is represented in hexadecimal notation. These non-array elements are concatenated exactly in the order they appear inside each array. Each tower of arrays is traversed starting from the outmost array to the inmost array.

Annex C (informative): A Simple Compiler for Basic CP-ABE policies

The present annex reports a simple technique for converting simple Boolean formulas to an CP-ABE access structure represented as a LSSS Matrices. This technique is described in Annex G of [i.1].

Input: a Boolean formula represented as a tree, where nodes are AND and OR gates and the leaf nodes correspond to attributes.

Output: a LSSS Matrix, where by convention the vector (1, 0, …, 0) is the "sharing vector" (the vector allowing to reconstruct the secret) for the LSSS matrix.

In the following the symbol | represents a concatenation operator:

1) Label the root node of the tree with the vector (1) - a vector of a single element, which is 1. Use a global counter c, and initialize it to 1.

2) Go down next tree level:

a) If the parent node is an OR gate labelled by the vector v, then its children are labelled by v (and the value of c stays the same).

b) If the parent node is an AND gate labelled by the vector v, pad v with 0's at the end (if necessary) to make it of length c; label one of its children with the vector v|1 and the other with the vector (0, …, 0)| - 1, where (0, …, 0) denotes the zero vector of length c.

3) Increment c by 1 and repeat from step 2 till each leaf of the tree is labelled.

4) Consider the vectors labelling each leaf. If these vectors have different lengths, pad the shorter ones with 0's at the end to arrive at vectors of the same length.

5) The vectors labelling the leaf nodes form the rows of the LSSS matrix.

Annex D (informative): Functional Credentials

D.1 Definitions

A Functional credentials scheme for an attribute universe Ω and a family of policies Φ consists of the following (polynomial time) algorithms and protocols:

1) *CKGen(1^{λ}) → (MSK, MPK)*: The key generation algorithm gets as input the security parameter and outputs a key pair *(MSK, MPK)* of an issuer (master key pair).

2) *GrantCred(MSK, S) → cred*: The grant credential algorithm gets input the master secret key *MSK* and a non-empty set of attributes $S \subset \Omega$ and it outputs a credential cred for the corresponding set of attributes.

3) *< ShowCred(MPK, cred, f), VrfyCred(MPK, f) >→ b*: ShowCred takes as input the master public key MPK, a credential cred, and a policy *f*; *VrfyCred* inputs the master public key *MPK* and a policy *f*. At the end, *VrfyCred* outputs either *0* or *1*.

By definition, for all $\lambda \in N$, for all *(MSK, MPK)* $\in$ *CKGen(1^{λ})* for all $S \subset \Omega$, for all *cred* $\in$ *GrantCred(osk, S)*, for all $f \in \Phi$ such that *f(S) = 1*, a Functional credentials scheme:

- is said correct if it holds that Pr[< ShowCred(MPK, cred, f), VrfyCred(MPK, f) >→ 1] = 1;
- is said unforgeable if, chosen an arbitrary policy f, any adversary having access to all system issued credentials cred but the ones satisfying the policy (i.e. for each credential cred, evaluation f(cred)$\neq$ 1 has a negligible probability to succeed in the credential verification process);
- is said anonymous if, arbitrarily chosen a policy f and two provers P0 and P1 showing credentials cred0 and cred1, both satisfying or not the policy (i.e. f(cred1) = f(cred2)) any adversary acting as a verifier cannot distinguish between them).

D.2 Correctness, unforgeability and anonymity of the protocol defined in the present document

Theorem. When using the protocol defined throughout the present document, a polynomial time adversary, acting as a Verifier, cannot distinguish between any two Provers P_0 and P_1 with different CP-WATERS-KEM keys K_0 and K_1, if these keys both satisfy (or not satisfy) the same access structure they are tested against.

Proof. Consider the following security game (adapted from [i.6]):

1) The Setup algorithm of CP-WATERS-KEM or the modified schema takes place. The public key PK is given to the adversary.

2) Any Prover P_i receives distinct secret keys K_i embedding some attributes.

3) The adversary is allowed to submit queries in the form $(r_{chosen}||K_{chosen}||AP)$ to an oracle which produces a random output *u* if this is the first time the input has been queried on. Otherwise, it gives back the previous response. In addition, the oracle computes the ciphertext *C* using the CCA-secure encryption algorithm (clause A.3.1) and records the couple $((r_{chosen}||K_{chosen}||AP), (C, u))$ in a table. This oracle operation is run throughout the whole game.

4) The adversary, acting as a Verifier *V*, arbitrarily chooses an access structure *AP* and two Provers P_0 and P_1, such that their corresponding keys either both satisfy, or both not satisfy the chosen access structure.

5) Depending on an internal coin toss *b*, the oracle impersonates prover P_b in the verification algorithm.

6) Verifier V sends a ciphertext *C* to the oracle.

7) The oracle either decrypts the ciphertext and return the correct message $m = (K_{chosen}||r_{chosen})$ or responds with ⊥.

8) The aforementioned steps (except the Setup) are repeated adaptively for any polynomial number of times on arbitrarily chosen access structure and arbitrarily chosen pairs of provers.

9) The Verifier try a guess b' and wins the game if $b == b'$ (i.e. she is able to guess which Prover has responded).

Modify this game as follows:

- At step 7, when given a ciphertext *C*, the oracle checks if *C* appears in the random oracle table. If so, it outputs the corresponding $m = (K_{chosen}||r_{chosen})$ value in the table; otherwise, it outputs ⊥ and rejects.

The difference between the original game and the modified one is negligible, as in the original game the oracle may decrypt even in case of a forged ciphertext (i.e. a ciphertext not computed using the CCA-secure encryption algorithm). However, since the oracle was not queries on $(r_{chosen}||K_{chosen}||AP)$, the probability that this event happens is bounded by the probability of apriori guessing a ciphertext output by an encryption for a given message without knowing the randomness used to encrypt.

The following observations apply to the modified game:

- If the Verifier produces a genuine ciphertext *C* following the CCA-secure Encryption algorithm, she gets a correct decryption *m* if the attributes embedded in the secret key K_b satisfy the chosen access structure *AP*, i.e. $AP(K_b) = 1$. Thus, the presented schema satisfies the correctness property (by definition).

- Vice versa, if the attributes embedded in the secret key K_b do not satisfy the chosen access structure *AP*, i.e. $AP(K_b) = 0$, the ciphertext would not decrypt at all except for a negligible probability ϵ. Hence, the presented scheme satisfies the unforgeability property (by definition).

Furthermore, it is possible to observe that:

- The access structure *AP* associated to the ciphertext *C* is always known to the challenger (given as input after being chosen by the adversary).

- Because a pseudo random generator is used, the ciphertext *C* is deterministically computed from the public key *PK* and the access structure *AP*.

- When the Verifier produces a genuine ciphertext *C* following the CCA-secure Encryption algorithm, the ciphertext C is uniformly distributed on the ciphertext space, because computed using the uniformly distributed randomness (step 3 of this clause).

- No decryption happens when the Verifier produces a forged ciphertext.

Under the three conditions above, suppose to modify the previous game replacing prover P_b's behaviour as follows:

- if key K_b embeds attributes satisfying the access structure *AP*, then message *m* is returned;

- otherwise ⊥ is returned.

That is, P_b no longer evaluates the decryption using the key K_b rather it (deterministically) returns *m* or ⊥ depending on the internal bit $AP(K_b)$. The introduced modification does not alter the advantage of the verifier *V* except for at most a negligible probability.

Since $AP(K_0) = AP(K_1)$ (by assumption, both keys satisfy or not satisfy the access structure), in the latter game the random coin *b* of the oracle remains hidden in an information-theoretic sense. This finally implies that the advantage of Verifier *V* is *1/2* in distinguish between P_0 and P_1.

Annex E (informative): ETSI Forge

A collection of JSON schemas that have been used for the running example presented throughout the present document is available in the ETSI Forge at the following URL: https://forge.etsi.org/rep/cyber/103964.

History

Document history		
V1.1.1	February 2025	Publication