Workgroup: LAMPS
Internet-Draft:
draft-ounsworth-pq-composite-keys-02
Published: 7 June 2022
Intended Status: Standards Track
Expires: 9 December 2022
Authors: M. Ounsworth M. Pala J. Klaussner
Entrust CableLabs D-Trust GmbH

Composite Public and Private Keys For Use In Internet PKI

Abstract

The migration to post-quantum cryptography is unique in the history of modern digital cryptography in that neither the old outgoing nor the new incoming algorithms are fully trusted to protect data for the required data lifetimes. The outgoing algorithms, such as RSA and elliptic curve, may fall to quantum cryptalanysis, while the incoming post-quantum algorithms face uncertainty about both the underlying mathematics as well as hardware and software implementations that have not had sufficient maturing time to rule out classical cryptanalytic attacks and implementation bugs.

Cautious implementors may wish to layer cryptographic algorithms such that an attacker would need to break all of them in order to compromise the data being protected. For digital signatures, this is referred to as "dual", and for encryption key establishment this as reffered to as "hybrid". This document, and its companions, defines a specific instantiation of the dual and hybrid paradigm called "composite" where multiple cryptographic algorithms are combined to form a single key, signature, or key encapsulation mechanism (KEM) such that they can be treated as a single atomic object at the protocol level.

EDNOTE: the terms "dual" and "hybrid" are currently in flux. We anticipate an Informational draft to normalize terminology, and will update this draft accordingly.

This document defines the structures CompositePublicKey and CompositePrivateKey, which are sequences of the respective structure for each component algorithm. The generic composite variant is defined which allows arbitrary combinations of key types to be placed in the CompositePublicKey and CompositePrivateKey structures without needing the combination to be pre-registered or pre-agreed. The explicit variant is also defined which allows for a set of algorithm identifier OIDs to be registered together as an explicit composite algorithm and assigned an OID.

This document is intended to be coupled with corresponding documents that define the structure and semantics of composite signatures and encryption, such as [draft-ounsworth-pq-composite-sigs-05] and draft-ounsworth-pq-composite-kem (yet to be published).

Status of This Memo

Copyright Notice

Table of Contents

1. Changes in version -02

*Merged Generic Composite (Section 4.1) and Explicit Composite (Section 4.2) into one document and made them share a wire encoding (only differing by the OIDs used).

*Removed Composite-OR Public Key.

*Synced document structure with -sigs

*Added Section 5.3 addressing backwards compatibility and ease of migration concerns.

TODO diff this against the public version and see if there are any more changes.

2. Introduction

During the transition to post-quantum cryptography, there will be uncertainty as to the strength of cryptographic algorithms; we will no longer fully trust traditional cryptography such as RSA, Diffie-Hellman, DSA and their elliptic curve variants, but we may also not fully trust their post-quantum replacements until further time has

passed to allow additional scrutiny and the discovery of implementation bugs. Unlike previous cryptographic algorithm migrations, the choice of when to migrate and which algorithms to migrate to, is not so clear. Even after the migration period, it may be advantageous for an entity's cryptographic identity to be composed of multiple public-key algorithms.

The deployment of composite public keys, and composite signatures and composite encryption using post-quantum algorithms will face two challenges:

* Algorithm strength uncertainty: During the transition period, some post-quantum signature and encryption algorithms will not be fully trusted, while also the trust in legacy public key algorithms will start to erode. A relying party may learn some time after deployment that a public key algorithm has become untrustworthy, but in the interim, they may not know which algorithm an adversary has compromised.

* Migration: During the transition period, systems will require mechanisms that allow for staged migrations from fully classical to fully post-quantum-aware cryptography.

This document provides a mechanism to address algorithm strength uncertainty concerns by providing formats for encoding multiple public key and private key values into existing public key and private key fields. Backwards compatibility is not directly addressed via the composite mechanisms defined in the document, but some notes on how it can be obtained can be found in Section 5.3.

This document is intended for general applicability anywhere that keys are used within PKIX or CMS structures.

2.1. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

The following terms are used in this document:

ALGORITHM: A standardized cryptographic primitive, as well as any ASN.1 structures needed for encoding data and metadata needed to use the algorithm. This document is concerned with algorithms for producing either digital signatures or ciphertexts for the purpose of key exchange.

BER: Basic Encoding Rules (BER) as defined in [X.690].

CLIENT: Any software that is making use of a cryptographic key. This includes a signer, verifier, encrypter, decrypter.

COMPONENT ALGORITHM: A single basic algorithm which is contained within a composite algorithm.

COMPOSITE ALGORITHM: An algorithm which is a combination of two or more component algorithms.

DER: Distinguished Encoding Rules as defined in [X.690].

LEGACY: For the purposes of this document, a legacy algorithm is any cryptographic algorithm currently in use which is not believed to be resistant to quantum cryptanalysis.

PKI: Public Key Infrastructure, as defined in [RFC5280].

POST-QUANTUM AGLORITHM: Any cryptographic algorithm which is believed to be resistant to classical and quantum cryptanalysis, such as the algorithms being considered for standardization by NIST.

PUBLIC / PRIVATE KEY: The public and private portion of an asymmetric cryptographic key, making no assumptions about which algorithm.

3. Composite Key Structures

In order to represent public keys and private keys that are composed of multiple algorithms, we define encodings consisting of a sequence of public key or private key primitives (aka "components") such that these structures can be used directly in existing public key fields such as those found in PKCS#10 [RFC2986], CMP [RFC4210], X.509 [RFC5280], CMS [RFC5652], and the Trust Anchor Format [RFC5914].

A composite key is a single key object that performs an atomic cryptographic operation -- such a signing, verifying, encapsulating, or decapsulating -- using its encapsulated sequence of component keys as if it was a single key. This generally means that the complexity of combining algorithms can be deferred from the protocol layer to the cryptographic library layer.

3.1. pk-Composite

The PUBLIC-KEY ASN.1 information object class is defined in [RFC5912]. The PUBLIC-KEY information object for generic (Section 4.1) and explicit (Section 4.2) composite public and private keys has the following form:

```
pk-Composite PUBLIC-KEY ::= {
    id <identifier>,
    KeyValue CompositePublicKey,
    Params ARE ABSENT,
    PrivateKey CompositePrivateKey,
}
```

The identifier may be an OID representing any composite key type.

Section 4.1 defines the object identifier id-composite-key which indicates that this is a "generic composite key" which allows arbitrary combinations of key types to be placed in the CompositePublicKey and CompositePrivateKey structures without needing the combination to be pre-registered or pre-agreed.

Section 4.2 defines a framework for defining new "explicit" combinations that use the same wire encoding structures as generic, but with OIDs that dictate specific combinations of component algorithms.

3.1.1. Key Usage

For protocols such as X.509 [RFC5280] that specify key usage along with the public key, any key usage may be used with composite keys, with the requirement that the specified key usage MUST apply to all component keys. For example if a composite key is marked with a KeyUsage of digitalSignature, then all component keys MUST be capable of producing digital signatures. The composite mechanism MUST NOT be used to implement mixed-usage keys, for example, where a digitalSignature and a keyEncipherment key are combined together into a single composite key.

3.2. CompositePublicKey

Composite public key data is represented by the following structure:

```
CompositePublicKey ::= SEQUENCE SIZE (2..MAX) OF SubjectPublicKeyInfo
```

A composite key MUST contain at least two component public keys.

A CompositePublicKey MUST NOT contain a component public key which itself describes a composite key; i.e. recursive CompositePublicKeys are not allowed.

EDNOTE: unclear that banning recursive composite keys actually accomplishes anything other than a general reduction in complexity and therefore reduction in attack surface.

Each component SubjectPublicKeyInfo SHALL contain an AlgorithmIdentifier OID which identifies the public key type and

parameters for the public key contained within it. See Appendix B for examples.

Each element of a CompositePublicKey is a SubjectPublicKeyInfo object encoding a component public key. When the CompositePublicKey must be provided in octet string or bit string format, the data structure is encoded as specified in Section 3.4.

3.3. CompositePrivateKey

EDNOTE: we need to put a bit more effort into private keys, specifically defining what OIDs to use in the generic and explicit cases.

This section provides an encoding for composite private keys intended for PKIX protocols and other applications that require an interoperable format for transmitting private keys, such as PKCS #12 [RFC7292] or CMP / CRMF [RFC4210], [RFC4211]. It is not intended to dictate a storage format in implementations not requiring interoperability of private key formats.

In some cases the private keys that comprise a composite key may not be represented in a single structure or even be contained in a single cryptographic module. The establishment of correspondence between public keys in a CompositePublicKey and private keys not represented in a single composite structure is beyond the scope of this document.

The composite private key data is represented by the following structure:

```
CompositePrivateKey ::= SEQUENCE SIZE (2..MAX) OF OneAsymmetricKey
```

Each element is a OneAsymmetricKey [RFC5958] object for a component private key.

The parameters field MUST be absent.

A CompositePrivateKey MUST contain at least two component private keys, and they MUST be in the same order as in the corresponding CompositePublicKey.

EDNOTE: does this also need an explicit version? It would probably reduce attack surface of tricking a client into running the wrong parser and a given piece of data.

3.4. Encoding Rules

Many protocol specifications will require that the composite public key and composite private key data structures be represented by an octet string or bit string.

When an octet string is required, the DER encoding of the composite data structure SHALL be used directly.

```
CompositePublicKeyOs ::= OCTET STRING (CONTAINING CompositePublicKey ENC
```

EDNOTE: will this definition include an ASN.1 tag and length byte inside the OCTET STRING object? If so, that's probably an extra uneccessary layer.

When a bit string is required, the octets of the DER encoded composite data structure SHALL be used as the bits of the bit string, with the most significant bit of the first octet becoming the first bit, and so on, ending with the least significant bit of the last octet becoming the last bit of the bit string.

```
CompositePublicKeyBs ::= BIT STRING (CONTAINING CompositePublicKey ENCOD
```

4. Algorithm Identifiers

This section defines the algorithm identifier for generic composite, as well as a framework for defining explicit combinations. This section is not intended to be exhaustive and other authors may define others so long as they are compatible with the structures and processes defined in this and companion signature and encryption documents.

Some use-cases desire the flexibility for client to use any combination of supported algorithms, while others desire the rigidity of explicitly-specified combinations of algorithms.

4.1. id-composite-key (Generic Composite Keys)

The id-composite-key algorithm identifier is used for identifying a generic composite public key and a generic composite private key. This allows arbitrary combinations of key types to be placed in the CompositePublicKey and CompositePrivateKey structures without needing the combination to be pre-registered or pre-agreed.

```
id-composite-key OBJECT IDENTIFIER ::= {
    joint-iso-itu-t(2) country(16) us(840) organization(1) entrust(11402
    Algorithm(80) Composite(4) CompositeKey(1) }
```

EDNOTE: this is a temporary OID for the purposes of prototyping. We are requesting IANA to assign a permanent OID, see Section 6.

Which yields an information object:

```
pk-Composite PUBLIC-KEY ::= {
    id id-composite-key,
    KeyValue CompositePublicKey,
    Params ARE ABSENT,
    PrivateKey CompositePrivateKey,
}
```

The motivation for this variant is primarily for prototyping work prior to the standardization of algorithm identifiers for explicit combinations of algorithms. However, the authors envision that this variant will remain relevant beyond full standardization for example in environments requiring very high levels of crypto agility, for example where clients support a large number of algorithms or where a large number of keys will be used at a time and it is therefore prohibitive to define algorithm identifiers for every combination of pairs, triples, quadruples, etc of algorithms.

4.2. Explicit Composite Keys

This variant provides a rigid way of specifying supported combinations of key types. This document does not define any explicit combinations, but provides a framework for doing so.

The motivation for this variant is to make it easier to reference and enforce specific combinations of algorithms. The authors envision this being useful for client-server negotiated protocols, protocol designers who wish to place constraints on allowable algorithm combinations in the protocol specification, as well as audited environments that wish to prove that only certain combinations will be supported by clients.

Profiles need to define an explicit composite key type which consists of:

- A new algorithm identifier OID for the explicit algorithm.

- The PUBLIC-KEY information object of each component public key type.

See Appendix A for guidance on creating and registering OIDs for specific explicit combinations.

In this variant, the public key is encoded as defined in Section 3 and Section 3.2, however the PUBLIC-KEY.id SHALL be an OID which is registered to represent a specific combination of component public key types. See Appendix B for examples.

The SubjectPublicKeyInfo.algorithm for each component key is redundant information which MUST match -- and can be inferred from -- the specification of the explicit algorithm. It has been left here for ease of implementation as the component SubjectPublicKeyInfo structures are the same between generic and explicit, as well as with single-algorithm keys. However, it introduces the risk of mismatch and leads to the following security consideration:

Security consideration: Implementations MUST check that the component AlgorithmIdentifier OIDs and parameters match those expected by the definition of the explicit algorithm. Implementations SHOULD first parse a component's SubjectPublicKeyInfo.algorithm, and ensure that it matches what is expected for that position in the explicit key, and then proceed to parse the SubjectPublicKeyInfo.subjectPublicKey. This is to reduce the attack surface associated with parsing the public key data of an unexpected key type, or worse; to parse and use a key which does not match the explicit algorithm definition. Similar checks MUST be done when handling the corresponding private key.

5. Implementation Considerations

This section addresses practical issues of how this draft affects other protocols and standards.

EDNOTE 10: Possible topics to address:

- The size of these certs and cert chains.
- In particular, implications for (large) composite keys / signatures / certs on the handshake stages of TLS and IKEv2.
- If a cert in the chain is a composite cert then does the whole chain need to be of composite Certs?
- We could also explain that the root CA cert does not have to be of the same algorithms. The root cert SHOULD NOT be transferred in the authentication exchange to save transport overhead and thus it can be different than the intermediate and leaf certs.

5.1. Textual encoding of Composite Private Keys

CompositePrivateKeys can be encoded to the Privacy-Enhanced Mail (PEM) [RFC1421] format by placing a CompositePrivateKey into the privateKey field of a PrivateKeyInfo or OneAsymmetricKey object, and then applying the PEM encoding rules as defined in [RFC7468] section 10 and 11 for plaintext and encrypted private keys, respectively.

5.2. Asymmetric Key Packages (CMS)

The Cryptographic Message Syntax (CMS), as defined in [RFC5652], can be used to digitally sign, digest, authenticate, or encrypt the asymmetric key format content type.

When encoding composite private keys, the privateKeyAlgorithm in the OneAsymmetricKey SHALL be set to id-composite-key or to an OID corresponding to an explicit composite key.

The parameters of the privateKeyAlgorithm SHALL be a sequence of AlgorithmIdentifier objects, each of which are encoded according to the rules defined for each of the different keys in the composite private key.

The value of the privateKey field in the OneAsymmetricKey SHALL be set to the DER encoding of the SEQUENCE of private key values that make up the composite key. The number and order of elements in the sequence SHALL be the same as identified in the sequence of parameters in the privateKeyAlgorithm.

The value of the publicKey (if present) SHALL be set to the DER encoding of the corresponding CompositePublicKey. If this field is present, the number and order of component keys MUST be the same as identified in the sequence of parameters in the privateKeyAlgorithm.

The value of the attributes is encoded as usual.

EDNOTE: I wonder whether this has value as its own section, or if we should take what's relevant and merge it into Section 3.3?

5.3. Backwards Compatibility

As noted in the introduction, the post-quantum cryptographic migration will face challenges in both ensuring cryptographic strength against adversaries of unknown capabilities, as well as providing ease of migration. The composite mechanisms defined in this document primarily address cryptographic strength, however this section contains notes on how backwards compatibility may be obtained.

The term "ease of migration" is used here to mean that existing systems can be gracefully transitioned to the new technology without requiring large service disruptions or expensive upgrades. The term "backwards compatibility" is used here to mean something more specific; that existing systems as they are deployed today can interoperate with the upgraded systems of the future.

These migration and interoperability concerns need to be thought about in the context of various types of protocols that make use of

X.509 and PKIX with relation to public key objects, from online negotiated protocols such as TLS 1.3 [RFC8446] and IKEv2 [RFC7296], to non-negotiated asynchronous protocols such as S/MIME signed and encrypted email [RFC8551], document signing such as in the context of the European eIDAS regulations [eIDAS2014], and publicly trusted code signing [codeSigningBRsv2.8], as well as myriad other standardized and proprietary protocols and applications that leverage CMS [RFC5652] signed or encrypted structures.

5.3.1. OR modes

This document purposefully does not specify how clients are to combine component keys together to form a single cryptographic operation; this is left up to the specifications of signature and encryption algorithms that make use of the composite key type. One possible way to combine component keys is through an OR relation, or OR-like client policies for acceptable algorithm combinations, where senders and / or receivers are permitted to ignore some component keys. Some envisioned uses of this include environments where the client encounters a component key for which it does not possess a compatible algorithm implementation but wishes to proceed with the cryptographic operation using the subset of component keys for which it does have compatible implementations. Such a mechanism could be designed to provide ease of migration by allowing for composite keys to be distributed and used before all clients in the environment are fully upgraded, but it does not allow for full backwards compatibility since clients would at least need to be upgraded from their current state to be able to parse the composite structures.

5.3.2. Parallel PKIs

We present the term "Parallel PKI" to refer to the setup where a PKI end entity possesses two or more distinct public keys or certificates for the same key type (signature, key establishment, etc) for the same identity (name, SAN), but containing keys for different cryptographic algorithms. One could imagine a set of parallel PKIs where an existing PKI using legacy algorithms (RSA, ECC) is left operational during the post-quantum migration but is shadowed by one or more parallel PKIs using pure post quantum algorithms or composite algorithms (legacy and post-quantum).

Equipped with a set of parallel public keys in this way, a client would have the flexibility to choose which public key(s) or certificate(s) to use in a given cryptographic operation.

For negotiated protocols, the client could choose which public key(s) or certificate(s) to use based on the negotiated algorithms, or could combine two of the public keys for example in a non-composite hybrid method such as [draft-becker-guthrie-noncomposite-

hybrid-auth-00] (NOTE: need kramdown formatting help with this ref) or [draft-guthrie-ipsecme-ikev2-hybrid-auth-00]. Note that it is possible to use the signature algorithm defined in [draft-ounsworth-pq-composite-sigs-06] as a way to carry the multiple signature values generated by a non-composite public mechanism in protocols where it is easier to support the composite signature algorithms than to implement such a mechanism in the protocol itself. There is also nothing precluding a composite public key from being one of the components used within a non-composite authentication operation; this may lead to greater convenience in setting up parallel PKI hierarchies that need to service a range of clients implementing different styles of post-quantum migration strategies.

For non-negotiated protocols, the details for obtaining backwards compatibility will vary by protocol, but for example in CMS [RFC5652], the inclusion of multiple SignerInfo or RecipientInfo objects is often already treated as an OR relationship, so including one for each of the end entity's parallel PKI public keys would, in many cases, have the desired effect of allowing the receiver to choose one they are compatible with and ignore the others, thus achieving full backwards compatibility.

6. IANA Considerations

The ASN.1 module OID is TBD. The id-composite-key and id-composite-or-key OIDs are to be assigned by IANA. The authors suggest that IANA assign an OID on the id-pkix arc:

```
id-composite-key OBJECT IDENTIFIER ::= {
    iso(1) identified-organization(3) dod(6) internet(1) security(5)
    mechanisms(5) pkix(7) algorithms(6) composite(??) }
```

7. Security Considerations

7.1. Reuse of keys in a Composite public key

There is an additional security consideration that some use cases such as signatures remain secure against downgrade attacks if and only if component keys are never used outside of their composite context and therefore it is RECOMMENDED that component keys in a composite key are not to be re-used in other contexts. In particular, the components of a composite key SHOULD NOT also appear in single-key certificates. This is particularly relevant for protocols that use composite keys in a logical AND mode since the appearance of the same component keys in single-key contexts undermines the binding of the component keys into a single composite key by allowing messages signed in a multi-key AND mode to be presented as if they were signed in a single key mode in what is known as a "stripping attack".

7.2. Key mismatch in explicit composite

This security consideration copied from Section 4.2.

Implementations MUST check that that the component AlgorithmIdentifier OIDs and parameters match those expected by the definition of the explicit algorithm. Implementations SHOULD first parse a component's SubjectPublicKeyInfo.algorithm, and ensure that it matches what is expected for that position in the explicit key, and then proceed to parse the SubjectPublicKeyInfo.subjectPublicKey. This is to reduce the attack surface associated with parsing the public key data of an unexpected key type, or worse; to parse and use a key which does not match the explicit algorithm definition. Similar checks MUST be done when handling the corresponding private key.

7.3. Policy for Deprecated and Acceptable Algorithms

Traditionally, a public key, certificate, or signature contains a single cryptographic algorithm. If and when an algorithm becomes deprecated (for example, RSA-512, or SHA1), it is obvious that clients performing signature verification or encryption operations should be updated to fail to validate or refuse to encrypt for these algorithms.

In the composite model this is less obvious since implementers may decide that certain cryptographic algorithms have complementary security properties and are acceptable in combination even though one or both algorithms are deprecated for individual use. As such, a single composite public key, certificate, signature, or ciphertext MAY contain a mixture of deprecated and non-deprecated algorithms.

Specifying behaviour in these cases is beyond the scope of this document, but should be considered by implementers and potentially in additional standards.

EDNOTE: Max is working on a CRL mechanism to accomplish this.

7.4. Protection of Private Keys

Structures described in this document do not protect private keys in any way unless combined with a security protocol or encryption properties of the objects (if any) where the CompositePrivateKey is used (see next Section).

Protection of the private keys is vital to public key cryptography. The consequences of disclosure depend on the purpose of the private key. If a private key is used for signature, then the disclosure allows unauthorized signing. If a private key is used for key management, then disclosure allows unauthorized parties to access

the managed keying material. The encryption algorithm used in the encryption process must be at least as 'strong' as the key it is protecting.

7.5. Checking for Compromised Key Reuse

Certification Authority (CA) implementations need to be careful when checking for compromised key reuse, for example as required by WebTrust regulations; when checking for compromised keys, you MUST unpack the CompositePublicKey structure and compare individual component keys. In other words, for the purposes of key reuse checks, the composite public key structures need to be un-packed so that primitive keys are being compared. For example if the composite key {RSA1, PQ1} is revoked for key compromise, then the keys RSA1 and PQ1 need to be individually considered revoked. If the composite key {RSA1, PQ2} is submitted for certification, it SHOULD be rejected because the key RSA1 was previously declared compromised even though the key PQ2 is unique.

8. References

8.1. Normative References

[RFC1421] Linn, J., "Privacy Enhancement for Internet Electronic Mail: Part I: Message Encryption and Authentication Procedures", RFC 1421, DOI 10.17487/RFC1421, February 1993, <https://www.rfc-editor.org/info/rfc1421>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC2986] Nystrom, M. and B. Kaliski, "PKCS #10: Certification Request Syntax Specification Version 1.7", RFC 2986, DOI 10.17487/RFC2986, November 2000, <https://www.rfc-editor.org/info/rfc2986>.

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <https://www.rfc-editor.org/info/rfc5280>.

[RFC5652] Housley, R., "Cryptographic Message Syntax (CMS)", STD 70, RFC 5652, DOI 10.17487/RFC5652, September 2009, <https://www.rfc-editor.org/info/rfc5652>.

[RFC5912] Hoffman, P. and J. Schaad, "New ASN.1 Modules for the Public Key Infrastructure Using X.509 (PKIX)", RFC 5912,

DOI 10.17487/RFC5912, June 2010, <https://www.rfc-editor.org/info/rfc5912>.

[RFC5914] Housley, R., Ashmore, S., and C. Wallace, "Trust Anchor Format", RFC 5914, DOI 10.17487/RFC5914, June 2010, <https://www.rfc-editor.org/info/rfc5914>.

[RFC5958] Turner, S., "Asymmetric Key Packages", RFC 5958, DOI 10.17487/RFC5958, August 2010, <https://www.rfc-editor.org/info/rfc5958>.

[RFC7468] Josefsson, S. and S. Leonard, "Textual Encodings of PKIX, PKCS, and CMS Structures", RFC 7468, DOI 10.17487/RFC7468, April 2015, <https://www.rfc-editor.org/info/rfc7468>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8411] Schaad, J. and R. Andrews, "IANA Registration for the Cryptographic Algorithm Object Identifier Range", RFC 8411, DOI 10.17487/RFC8411, August 2018, <https://www.rfc-editor.org/info/rfc8411>.

[X.690] ITU-T, "Information technology - ASN.1 encoding Rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)", ISO/IEC 8825-1:2015, November 2015.

8.2. Informative References

[Bindel2017] Bindel, N., Herath, U., McKague, M., and D. Stebila, "Transitioning to a quantum-resistant public key infrastructure", 2017, <https://link.springer.com/chapter/10.1007/978-3-319-59879-6_22>.

[codeSigningBRsv2.8] CAB Forum, ., "Baseline Requirements for the Issuance and Management of Publicly-Trusted Code Signing Certificates v2.8", May 2022, <https://cabforum.org/wp-content/uploads/Baseline-Requirements-for-the-Issuance-and-Management-of-Code-Signing.v2.8.pdf>.

[eIDAS2014] "REGULATION (EU) No 910/2014 OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL of 23 July 2014 on electronic identification and trust services for electronic transactions in the internal market and repealing Directive 1999/93/EC", July 2014, <https://ec.europa.eu/futurium/en/system/files/ged/eidas_regulation.pdf>.

[I-D.becker-guthrie-noncomposite-hybrid-auth] Becker, A., Guthrie, R., and M. J. Jenkins, "Non-Composite Hybrid Authentication in PKIX and Applications to Internet Protocols", Work in Progress, Internet-Draft, draft-becker-guthrie-noncomposite-hybrid-auth-00, 22 March 2022, <https://www.ietf.org/archive/id/draft-becker-guthrie-noncomposite-hybrid-auth-00.txt>.

[I-D.guthrie-ipsecme-ikev2-hybrid-auth] Guthrie, R., "Hybrid Non-Composite Authentication in IKEv2", Work in Progress, Internet-Draft, draft-guthrie-ipsecme-ikev2-hybrid-auth-00, 25 March 2022, <https://www.ietf.org/archive/id/draft-guthrie-ipsecme-ikev2-hybrid-auth-00.txt>.

[I-D.ounsworth-pq-composite-sigs] Ounsworth, M. and M. Pala, "Composite Signatures For Use In Internet PKI", Work in Progress, Internet-Draft, draft-ounsworth-pq-composite-sigs-05, 12 July 2021, <https://www.ietf.org/archive/id/draft-ounsworth-pq-composite-sigs-05.txt>.

[RFC3279] Bassham, L., Polk, W., and R. Housley, "Algorithms and Identifiers for the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 3279, DOI 10.17487/RFC3279, April 2002, <https://www.rfc-editor.org/info/rfc3279>.

[RFC4210] Adams, C., Farrell, S., Kause, T., and T. Mononen, "Internet X.509 Public Key Infrastructure Certificate Management Protocol (CMP)", RFC 4210, DOI 10.17487/RFC4210, September 2005, <https://www.rfc-editor.org/info/rfc4210>.

[RFC4211] Schaad, J., "Internet X.509 Public Key Infrastructure Certificate Request Message Format (CRMF)", RFC 4211, DOI 10.17487/RFC4211, September 2005, <https://www.rfc-editor.org/info/rfc4211>.

[RFC7292] Moriarty, K., Ed., Nystrom, M., Parkinson, S., Rusch, A., and M. Scott, "PKCS #12: Personal Information Exchange Syntax v1.1", RFC 7292, DOI 10.17487/RFC7292, July 2014, <https://www.rfc-editor.org/info/rfc7292>.

[RFC7296] Kaufman, C., Hoffman, P., Nir, Y., Eronen, P., and T. Kivinen, "Internet Key Exchange Protocol Version 2

(IKEv2)", STD 79, RFC 7296, DOI 10.17487/RFC7296, October 2014, <https://www.rfc-editor.org/info/rfc7296>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

[RFC8551] Schaad, J., Ramsdell, B., and S. Turner, "Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 4.0 Message Specification", RFC 8551, DOI 10.17487/RFC8551, April 2019, <https://www.rfc-editor.org/info/rfc8551>.

Appendix A. Creating explicit combinations

The following ASN.1 Information Objects may be useful in defining and parsing explicit pairs of public key types. Given an ASN.1 2002 compliant ASN.1 compiler, these Information Objects will enforce the binding between the public key types specified in the instantiation of pk-explicitComposite, and the wire objects which implement it. The one thing that is not enforced automatically by this Information Object is that publicKey.params are intended to be absent if and only if they are absent for the declared public key type. This ASN.1 module declares them OPTIONAL and leaves it to implementers to perform this check explicitly.

EDNOTE this ASN.1 needs to change. The current definition doesn't put a component AlgorithmIdentifier with each component key. Once we agree as a group that the text accurately describes what we want, we can spend a bit of time figuring out if the ASN.1 machinery lets us express it in a readable way and/or a way that will actually help people creating explicit pairs.

```
-- pk-explicitComposite - Composite public key information object

pk-explicitComposite{OBJECT IDENTIFIER:id, PUBLIC-KEY:firstPublicKey,
 FirstPublicKeyType, PUBLIC-KEY:secondPublicKey, SecondPublicKeyType}
 PUBLIC-KEY ::= {PUBLIC-KEYPUBLIC-KEY
    IDENTIFIER id
    KEY ExplicitCompositePublicKey{firstPublicKey, FirstPublicKeyType,
     secondPublicKey, SecondPublicKeyType}
    PARAMS ARE absent
    CERT-KEY-USAGE {digitalSignature, nonRepudiation, keyCertSign,
      cRLSign}
}
```

The following ASN.1 object class then automatically generates the public key structure from the types defined in pk-explicitComposite.

```
-- ExplicitCompositePublicKey - The data structure for a composite
-- public key sec-composite-pub-keys and SecondPublicKeyType are needed
-- because PUBLIC-KEY contains a set of public key types, not a single
-- type.
-- TODO The parameters should be optional only if they are marked
-- optional in the PUBLIC-KEY.


ExplicitCompositePublicKey{PUBLIC-KEY:firstPublicKey, FirstPublicKeyType
  PUBLIC-KEY:secondPublicKey, SecondPublicKeyType} ::= SEQUENCE {
    firstPublicKey SEQUENCE {
        params firstPublicKey.&Params OPTIONAL,
        publicKey FirstPublicKeyType
    },
    secondPublicKey SEQUENCE {
        params secondPublicKey.&Params OPTIONAL,
        publicKey SecondPublicKeyType
    }
}
```

Using this module, it becomes trivial to define explicit pairs. For an example, see Appendix B.2.

To define explicit triples, quadruples, etc, these Information Objects can be extended to have thirdPublicKey, fourthPublicKey, etc throughout.

Appendix B. Examples

B.1. Generic Composite Public Key Examples

This is an example generic composite public key

```
-----BEGIN PUBLIC KEY-----
MIIBmDAMBgpghkgBhvprUAQBA4IBhgAwggGBMFkwEwYHKoZIzj0CAQYIKoZIzj0D
AQcDQgAExGPhrnuSG/fGyw1FN+l5h4p4AGRQCS0LBXnBO+djhcI6qnF2TvrQEaIY
GGpQT5wHS+7y5iJJ+dE5qjxcv8loRDCCASIwDQYJKoZIhvcNAQEBBQADggEPADCC
AQoCggEBANsVQK1fcLQObL4ZYtczWbObECAFSsng0OLpRTPr9VGV3SsS/VoMRZqX
F+sszz6I2UcFTaMF9CwNRbWLuIBczzuhbHSjn65OuoN+Om2wsPo+okw46RTekB4a
d9QQvYRVzPlILUQ8NvZ4W0BKLviXTXWIggjtp/Y1pKRHKz8n35J6OmFWz4TKGNth
n87D28kmdwQYH5NLsDePHbfdw3AyLrPvQLlQw/hRPz/9Txf7yi9Djg9HtJ88ES6+
ZbfE1ZHxLYLSDt25tSL8A2pMuGMD3P81nYWO+gJ0vYV2WcRpXHRkjmliGqiCg4eB
mC4//tm0J4r9Ll8b/pp6xyOMI7jppVUCAwEAAQ==
-----END PUBLIC KEY-----
```

which decodes as:

```
algorithm: AlgorithmIdentifier{id-composite-key}

subjectPublicKey: CompositePublicKey {
  SubjectPublicKeyInfo {
    algorithm: AlgorithmIdentifier {
      algorithm: ecPublicKey
      parameters: prime256v1
      }
    subjectPublicKey: <ec key octet string>
    },
    SubjectPublicKeyInfo {
    algorithm: AlgorithmIdentifier {
      algorithm: rsaEncryption
      parameters: NULL
      }
    subjectPublicKey: <rsa key octet string>
    }
  }
```

The corresponding generic private key is:

```
-----BEGIN PRIVATE KEY-----
MIIFHgIBADAMBgpghkgBhvprUAQBBIIFCTCCBQUwQQIBADATBgcqhkjOPQIBBggq
hkjOPQMBBwQnMCUCAQEEICN0ihCcgg5n8ALtk9tkQZqg/WLEm5NefMi/kdN06Z9u
MIIEvgIBADANBgkqhkiG9w0BAQEFAASCBKgwggSkAgEAAoIBAQDbFUCtX3C0Dmy+
GWLXM1mzmxAgBUrJ4NDi6UUz6/VRld0rEv1aDEWalxfrLM8+iNlHBU2jBfQsDUW1
i7iAXM87oWx0o5+uTrqDfjptsLD6PqJMOOkU3pAeGnfUEL2EVcz5SC1EPDb2eFtA
Si74l011iIII7af2NaSkRys/J9+SejphVs+EyhjbYZ/Ow9vJJncEGB+TS7A3jx23
3cNwMi6z70C5UMP4UT8//U8X+8ovQ44PR7SfPBEuvmW3xNWR8S2C0g7dubUi/ANq
TLhjA9z/NZ2FjvoCdL2FdlnEaVx0ZI5pYhqogoOHgZguP/7ZtCeK/S5fG/6aescj
jCO46aVVAgMBAAECggEAFtT6LpdZuYofTxh6Mo9Jc+xfG9cxWiSx4FQLQEQBBwWl
TQ3nlXDd+CRy+7Fpz8yXSE2HL8w5DDY945OyIL6LYl2KXgWHaLUPvxByqmfVqd7J
L0RnFiOzxU9g2Zr9BUOj3v7kqM3VtI4KhIK2rnWmPu+BDckmzgP9Kpm4KhbPuAYP
iqUZSkxpSUsd5ALLsk9b0xjR7UEYkEpV2/vORwieEhOmPLzuXh+Px0yavkazT/vU
+h/rDSoLQn7v4fVsQgNdOaaOG/gHemGuuiLPJJlX5ZZ6mmsIaEjz+MNk0aJDH2po
KbAr4B709dTsnYgv7YtkEfSyOeMEdhMiswI1c9FpwQKBgQD6kdHmHCoeWNNvlqxU
v57e7ZDAXDA6WcfrypcsF0l72rI3J8oOPmFaNaCmwIH/Icz+Zy7fr2IYxVjyDjCa
zi8qTnj2ZNds71hUYOcq60u0TcSVrtocA4HW7NoWJqK5thNlNaa1M358cYBopGoN
ocS9yf10q2MBZtpF0fc5PbFf+QKBgQDf1L4cezoebbNTaN4KoapycHXxKozP2GwI
r15YRYjt0ZpHstdUPABQuwlL9CuL+5Q17VRiM81cUVNfFsBzKIXYb/PBC5UD+DmR
qGlT6v6uUWY6jifUgEjfyPxO0oJ3M6cChHR/TvpkT5SyaEwHpIH7IeXbMFcS5m4G
mSNBECO/PQKBgCD0CoHT1Go3Tl9PloxywwcYgT/7H9CcvCEzfJws19o1EdkVH4qu
A4mkoeMsUCxompgeo9iBLUqKsb7rxNKnKSbMOTZWXsqR07ENKXnIhiVJUQBKhZ7H
i0zjy268WAxKeNSHsMwF4K2nE7cvYE84pjI7nVy5qYSmrTAfg/8AMRKpAoGBAN/G
wN6WsE9Vm5BLapo0cMUC/FdFFAyEMdYpBei4dCJXiKgf+7miVypfI/dEwPitZ8rW
YKPhaHHgeLq7c2JuZAo0Ov2IR831MBEYz1zvtvmuNcda8iU4sCLTvLRNL9Re1pzk
sdfJrPn2uhH3xfNqG+1oQXZ3CMbDi8Ka/a0Bpst9AoGBAPR4p6WN0aoZlosyT6NI
4mqzNvLE4KBasmfoMmTJih7qCP3X4pqdgiI0SjsQQG/+utHLoJARwzhWHOZf1JKk
D8lSJH02cp/Znrjn5wPpfYKLphJBiKSPwyIjuFwcR1ck84ONeYq421NDqf7lXbvx
oMqjTPagXUpzHvwluDjtSi8+
-----END PRIVATE KEY-----
```

   which decodes as:

```
algorithm: AlgorithmIdentifier{id-composite-key}

SEQUENCE {
  OneAsymmetricKey {
      version: 0,
      privateKeyAlgorithm: PrivateKeyAlgorithmIdentifier{
        algorithm: ecPublicKey
        parameters: prime256v1
      }
      privateKey: <ec key octet string>
    },
  OneAsymmetricKey {
      version: 0,
      privateKeyAlgorithm: PrivateKeyAlgorithmIdentifier{
        algorithm: rseEncryption
        parameters: NULL
      }
      privateKey: <rsa key octet string>
    }
  }
```

B.2. Explicit Composite Public Key Examples

Assume that the following is a defined explicit pair:

```
id-pk-example-ECandRSA OBJECT IDENTIFIER ::= { 1 2 3 4 }

pk-example-ECandRSA PUBLIC-KEY ::= pk-explicitComposite{
    id-pk-example-ECandRSA,
    ecPublicKey,
    pk-ec,
    rsaEncryption,
    pk-rsa,
}
```

Then the same key as above could be encoded as an explicit composite public key as:

```
-----BEGIN PUBLIC KEY-----
MIIBkTAFBgMqAwQDggGGADCCAYEwWTATBgcqhkjOPQIBBggqhkjOPQMBBwNCAATE
Y+Gue5Ib98bLDUU36XmHingAZFAJLQsFecE752OFwjqqcXZO+tARohgYalBPnAdL
7vLmIkn50TmqPFy/yWhEMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA
2xVArV9wtA5svhli1zNZs5sQIAVKyeDQ4ulFM+v1UZXdKxL9WgxFmpcX6yzPPojZ
RwVNowX0LA1FtYu4gFzPO6FsdKOfrk66g346bbCw+j6iTDjpFN6QHhp31BC9hFXM
+UgtRDw29nhbQEou+JdNdYiCCO2n9jWkpEcrPyffkno6YVbPhMoY22GfzsPbySZ3
BBgfk0uwN48dt93DcDIus+9AuVDD+FE/P/1PF/vKL0OOD0e0nzwRLr5lt8TVkfEt
gtIO3bm1IvwDaky4YwPc/zWdhY76AnS9hXZZxGlcdGSOaWIaqIKDh4GYLj/+2bQn
iv0uXxv+mnrHI4wjuOmlVQIDAQAB
-----END PUBLIC KEY-----
```

which decodes as:

```
algorithm: AlgorithmIdentifier{id-pk-example-ECandRSA}

subjectPublicKey: CompositePublicKey {
  SubjectPublicKeyInfo {
    algorithm: AlgorithmIdentifier {
      algorithm: ecPublicKey
      parameters: prime256v1
      }
    subjectPublicKey: <ec key octet string>
    },
    SubjectPublicKeyInfo {
    algorithm: AlgorithmIdentifier {
      algorithm: rsaEncryption
      parameters: NULL
      }
    subjectPublicKey: <rsa key octet string>
    }
  }
```

The corresponding explicit private key is:

```
-----BEGIN PRIVATE KEY-----
MIIFFwIBADAFBgMqAwQEggUJMIIFBTBBAgEAMBMGByqGSM49AgEGCCqGSM49AwEH
BCcwJQIBAQQgI3SKEJyCDmfwAu2T22RBmqD9YsSbk158yL+R03Tpn24wggS+AgEA
MA0GCSqGSIb3DQEBAQUABIIEqDCCBKQCAQACggEBANsVQK1fcLQObL4ZYtczWbOb
ECAFSsng0OLpRTPr9VGV3SsS/VoMRZqXF+sszz6I2UcFTaMF9CwNRbWLuIBczzuh
bHSjn65OuoN+Om2wsPo+okw46RTekB4ad9QQvYRVzPlILUQ8NvZ4W0BKLviXTXWI
ggjtp/Y1pKRHKz8n35J6OmFWz4TKGNthn87D28kmdwQYH5NLsDePHbfdw3AyLrPv
QLlQw/hRPz/9Txf7yi9Djg9HtJ88ES6+ZbfE1ZHxLYLSDt25tSL8A2pMuGMD3P81
nYWO+gJ0vYV2WcRpXHRkjmliGqiCg4eBmC4//tm0J4r9Ll8b/pp6xyOMI7jppVUC
AwEAAQKCAQAW1Poul1m5ih9PGHoyj0lz7F8b1zFaJLHgVAtARAEHBaVNDeeVcN34
JHL7sWnPzJdITYcvzDkMNj3jk7IgvotiXYpeBYdotQ+/EHKqZ9Wp3skvRGcWI7PF
T2DZmv0FQ6Pe/uSozdW0jgqEgraudaY+74ENySbOA/0qmbgqFs+4Bg+KpRlKTGlJ
Sx3kAsuyT1vTGNHtQRiQSlXb+85HCJ4SE6Y8vO5eH4/HTJq+RrNP+9T6H+sNKgtC
fu/h9WxCA105po4b+Ad6Ya66Is8kmVfllnqaawhoSPP4w2TRokMfamgpsCvgHvT1
1OydiC/ti2QR9LI54wR2EyKzAjVz0WnBAoGBAPqR0eYcKh5Y02+WrFS/nt7tkMBc
MDpZx+vKlywXSXvasjcnyg4+YVo1oKbAgf8hzP5nLt+vYhjFWPIOMJrOLypOePZk
12zvWFRg5yrrS7RNxJWu2hwDgdbs2hYmorm2E2U1prUzfnxxgGikag2hxL3J/XSr
YwFm2kXR9zk9sV/5AoGBAN/Uvhx7Oh5ts1No3gqhqnJwdfEqjM/YbAivXlhFiO3R
mkey11Q8AFC7CUv0K4v7lDXtVGIzzVxRU18WwHMohdhv88ELlQP4OZGoaVPq/q5R
ZjqOJ9SASN/I/E7SgnczpwKEdH9O+mRPlLJoTAekgfsh5dswVxLmbgaZI0EQI789
AoGAIPQKgdPUajdOX0+WjHLDBxiBP/sf0Jy8ITN8nCzX2jUR2RUfiq4DiaSh4yxQ
LGiamB6j2IEtSoqxvuvE0qcpJsw5NlZeypHTsQ0peciGJUlRAEqFnseLTOPLbrxY
DEp41IewzAXgracTty9gTzimMjudXLmphKatMB+D/wAxEqkCgYEA38bA3pawT1Wb
kEtqmjRwxQL8V0UUDIQx1ikF6Lh0IleIqB/7uaJXKl8j90TA+K1nytZgo+FoceB4
urtzYm5kCjQ6/YhHzfUwERjPXO+2+a41x1ryJTiwItO8tE0v1F7WnOSx18ms+fa6
EffF82ob7WhBdncIxsOLwpr9rQGmy30CgYEA9HinpY3RqhmWizJPo0jiarM28sTg
oFqyZ+gyZMmKHuoI/dfimp2CIjRKOxBAb/660cugkBHDOFYc5l/UkqQPyVIkfTZy
n9meuOfnA+l9goumEkGIpI/DIiO4XBxHVyTzg415irjbU0Op/uVdu/GgyqNM9qBd
SnMe/CW4OO1KLz4=
-----END PRIVATE KEY-----
```

which decodes as:

```
algorithm: AlgorithmIdentifier{id-pk-example-ECandRSA}

SEQUENCE {
  OneAsymmetricKey {
      version: 0,
      privateKeyAlgorithm: PrivateKeyAlgorithmIdentifier{
        algorithm: ecPublicKey
        parameters: prime256v1
      }
      privateKey: <ec key octet string>
    },
  OneAsymmetricKey {
      version: 0,
      privateKeyAlgorithm: PrivateKeyAlgorithmIdentifier{
        algorithm: rseEncryption
        parameters: NULL
      }
      privateKey: <rsa key octet string>
    }
  }
```

Appendix C. ASN.1 Module

```
<CODE STARTS>

Composite-Keys-2022

DEFINITIONS IMPLICIT TAGS ::= BEGIN

EXPORTS ALL;

IMPORTS
  PUBLIC-KEY, SIGNATURE-ALGORITHM, ParamOptions, AlgorithmIdentifier{}
    FROM AlgorithmInformation-2009  -- RFC 5912 [X509ASN1]
      { iso(1) identified-organization(3) dod(6) internet(1)
        security(5) mechanisms(5) pkix(7) id-mod(0)
        id-mod-algorithmInformation-02(58) }

  SubjectPublicKeyInfo
    FROM PKIX1Explicit-2009
      { iso(1) identified-organization(3) dod(6) internet(1)
        security(5) mechanisms(5) pkix(7) id-mod(0)
        id-mod-pkix1-explicit-02(51) }

  OneAsymmetricKey
    FROM AsymmetricKeyPackageModuleV1
      { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1)
        pkcs-9(9) smime(16) modules(0)
        id-mod-asymmetricKeyPkgV1(50) } ;

--
-- Object Identifiers
--

der OBJECT IDENTIFIER ::=
  {joint-iso-itu-t asn1(1) ber-derived(2) distinguished-encoding(1)}

-- To be replaced by IANA
id-composite-key OBJECT IDENTIFIER ::= {
    joint-iso-itu-t(2) country(16) us(840) organization(1) entrust(11402
    Algorithm(80) Composite(4) CompositeKey(1)



--  COMPOSITE-KEY-ALGORITHM
--
--  Describes the basic properties of a composite key algorithm
--
--  &id - contains the OID identifying the composite algorithm
--  &Params - if present, contains the type for the algorithm
--               parameters; if absent, implies no parameters
--  &paramPresence - parameter presence requirement
--
```

```
-- }

COMPOSITE-KEY-ALGORITHM ::= CLASS {
    &id             OBJECT IDENTIFIER UNIQUE,
    &Params         OPTIONAL,
    &paramPresence  ParamOptions DEFAULT absent
} WITH SYNTAX {
    IDENTIFIER &id
    [PARAMS [TYPE &Params] ARE &paramPresence ]
}


CompositeAlgorithmIdentifier ::= AlgorithmIdentifier{COMPOSITE-KEY-ALGOR

CompositeAlgorithmSet COMPOSITE-KEY-ALGORITHM ::= {
  CompositeAlgorithms, ...
}

--
-- Public Key
--

pk-Composite PUBLIC-KEY ::= {
    IDENTIFIER id-composite-key
    KEY CompositePublicKey
    PARAMS TYPE CompositeAlgorithmIdentifier ARE optional
    PRIVATE-KEY CompositePrivateKey
}

CompositePublicKey ::= SEQUENCE SIZE (2..MAX) OF SubjectPublicKeyInfo

CompositePublicKeyOs ::= OCTET STRING (CONTAINING CompositePublicKey ENC

CompositePublicKeyBs ::= BIT STRING (CONTAINING CompositePublicKey ENCOD

CompositePrivateKey ::= SEQUENCE SIZE (2..MAX) OF OneAsymmetricKey


-- pk-explicitComposite - Composite public key information object

pk-explicitComposite{OBJECT IDENTIFIER:id, PUBLIC-KEY:firstPublicKey,
  FirstPublicKeyType, PUBLIC-KEY:secondPublicKey, SecondPublicKeyType}
  PUBLIC-KEY ::= {
    IDENTIFIER id
    KEY ExplicitCompositePublicKey{firstPublicKey, FirstPublicKeyType,
      secondPublicKey, SecondPublicKeyType}
    PARAMS ARE absent
}

-- The following ASN.1 object class then automatically generates the
```

```
-- public key structure from the types defined in pk-explicitComposite.

-- ExplicitCompositePublicKey - The data structure for a composite
-- public key sec-composite-pub-keys and SecondPublicKeyType are needed
-- because PUBLIC-KEY contains a set of public key types, not a single
-- type.
-- TODO The parameters should be optional only if they are marked
-- optional in the PUBLIC-KEY


ExplicitCompositePublicKey{PUBLIC-KEY:firstPublicKey, FirstPublicKeyType
  PUBLIC-KEY:secondPublicKey, SecondPublicKeyType} ::= SEQUENCE {
    firstPublicKey SEQUENCE {
        params firstPublicKey.&Params OPTIONAL,
        publicKey FirstPublicKeyType
    },
    secondPublicKey SEQUENCE {
        params secondPublicKey.&Params OPTIONAL,
        publicKey SecondPublicKeyType
    }
}

END

<CODE ENDS>
```

Appendix D. Intellectual Property Considerations

The following IPR Disclosure relates to this draft:

https://datatracker.ietf.org/ipr/3588/

Appendix E. Contributors and Acknowledgements

This document incorporates contributions and comments from a large group of experts. The Editors would especially like to acknowledge the expertise and tireless dedication of the following people, who attended many long meetings and generated millions of bytes of electronic mail and VOIP traffic over the past year in pursuit of this document:

John Gray (Entrust), Serge Mister (Entrust), Scott Fluhrer (Cisco Systems), Panos Kampanakis (Cisco Systems), Daniel Van Geest (ISARA), Tim Hollebeek (Digicert), Klaus-Dieter Wirth (D-Trust), and Francois Rousseau.

We are grateful to all, including any contributors who may have been inadvertently omitted from this list.

This document borrows text from similar documents, including those referenced below. Thanks go to the authors of those documents. "Copying always makes things easier and less error prone" - [RFC8411].

E.1. Making contributions

Additional contributions to this draft are welcome. Please see the working copy of this draft at, as well as open issues at:

https://github.com/EntrustCorporation/draft-ounsworth-pq-composite-keys

Authors' Addresses

Mike Ounsworth
Entrust Limited
2500 Solandt Road -- Suite 100
Ottawa, Ontario K2K 3G5
Canada

Email: mike.ounsworth@entrust.com

Massimiliano Pala
CableLabs

Email: director@openca.org

Jan Klaussner
D-Trust GmbH
Kommandantenstr. 15
10969 Berlin
Germany

Email: jan.klaussner@d-trust.net