Abstract

With the widespread adoption of post-quantum cryptography will come the need for an entity to possess multiple public keys on different cryptographic algorithms. Since the trustworthiness of individual post-quantum algorithms is at question, a multi-key cryptographic operation will need to be performed in such a way that breaking it requires breaking each of the component algorithms individually. This requires defining new structures for holding composite public keys and composite signature data.

This document defines the structures CompositePublicKey, CompositeSignatureValue, and CompositeParams, which are sequences of the respective structure for each component algorithm. This document also defines algorithms for generating and verifying composite signatures. This document makes no assumptions about what the component algorithms are, provided that their algorithm identifiers and signature generation and verification algorithms are defined.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on January 5, 2020.
1. 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 will also not fully trust their post-quantum replacements until they have had sufficient scrutiny. 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 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 also 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.

- Backwards compatibility: During the transition period, post-quantum algorithms will not be supported by all clients.

This document provides a mechanism to address algorithm strength uncertainty by providing formats for encoding multiple public keys and multiple signature values into existing public key and signature fields, as well as an algorithm for validating a composite signature. The issue of backwards compatibility is left open to be addressed in separate draft(s).

This document is intended for general applicability anywhere that public key structures or digital signatures are used within PKIX structures.

EDNOTE: While the scope of this document is restricted to signatures, we note that the same "CompositePublicKey" structure is equally applicable to asymmetric encryption keys. Though a word of warning that the corresponding "encrypt / decrypt with a composite public key" logic is somewhat less obvious; a naive implementer might be tempted to follow the same pattern as below and encrypt the message with each public key separately and then concatenate the ciphertexts, which is wrong, they need to be nested. Specifying the correct implementation of such an encryption scheme is out of scope for this
document, but would be good work for someone in the standards community to pick up.

1.1. 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:
An information object class for identifying the type of cryptographic operation to be performed. This document is primarily concerned with algorithms for producing digital signatures, though the public key structure could just as easily hold encryption keys.

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

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

COMPOSITE ALGORITHM:
An algorithm which is a sequence of one or more basic algorithm, as defined in Section 2.

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

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

PRIMITIVE PUBLIC KEY / SIGNATURE:
A public key or signature object of a non-composite algorithm type.

SIGNATURE:
A digital cryptographic signature, making no assumptions about which algorithm.
2. Composite Structures

In order for public keys and signatures to be composed of multiple algorithms, we define encodings consisting of a sequence of public key and signature primitives (aka "component algorithms") such that these structures can be used as a drop-in compatible way with existing public key or signature fields such as those found in PKCS#10 [RFC2986], CMP [RFC4210], X.509 [RFC5280], CMS [RFC5652].

This section defines the following structures:

- The id-alg-composite is an OID identifying a composite public key or signature object.
- The CompositePublicKey carries all the public keys associated with an identity within a single public key structure.
- The CompositePrivateKey carries all the private keys associated with an identity within a single private key structure.
- The CompositeSignatureValue, carries a sequence of signatures that are generated by a CompositePrivateKey, and can be verified with the corresponding compositePublicKey.

EDNOTE: the choice to define composite algorithm parameters as a sequence inside the existing fields avoids the exponential proliferation of OIDs that are needed for each pairwise combination of signature algorithms in other schemes for achieving multi-key certificates. This scheme also naturally extends from 2-keypair to n-keypair keys and certificates.

2.1. Algorithm Identifier

The same algorithm identifier is used for identifying a public key, a private key, and a signature. Additional encoding information is provided below for each of these objects.

```plaintext
id-alg-composite OBJECT IDENTIFIER ::= {
  iso(1)  identified-organization(3) dod(6) internet(1) private(4)
  enterprise(1) OpenCA(18227) Algorithms(2) id-alg-composite(1) }
```

EDNOTE: this is a temporary OID for the purposes of prototyping. We are requesting IANA to assign a permanent OID, see Section 5.
2.2. Composite Keys

A composite key is a single key object that performs an atomic signature or verification operation, using its encapsulated sequence of component keys.

The ASN.1 algorithm object for composite public and private keys is:

```asn1
pk-Composite PUBLIC-KEY ::= {
  IDENTIFIER id-alg-composite
  KEY CompositePublicKey
  PARAMS ARE absent
  CERT-KEY-USAGE
    { digitalSignature, nonRepudiation, keyCertSign, cRLSign }  
  PRIVATE-KEY CompositePrivateKey
}
```

EDNOTE1: the authors are currently unsure whether the params should be absent (ie this structure simply says "I am a composite algorithm"), or used to duplicate some amount of information about what the component algorithms are. See Section 2.3 for a longer ENDOTE on this.

EDNOTE2: In order to reduce complexity, we are intentionally limiting the scope of this draft to signature-type CERT-KEY-USAGES, but we note that it would be trivial to extend it to encryption-type keys.

2.2.1. Key Usage Bits

The intended application for the key is indicated in the keyUsage certificate extension and defined in the CERT-KEY-USAGE field of pk-Composite.

If the keyUsage extension is present in an end-entity certificate that indicates id-alg-composite, then the keyUsage extension MUST contain one or both of the following values:

```
  nonRepudiation; and
digitalSignature.
```

If the keyUsage extension is present in a certification authority certificate that indicates id-alg-composite, then the keyUsage extension MUST contain one or more of the following values:

```
  nonRepudiation;
digitalSignature;
keyCertSign; and
cRLSign.
```
As this draft only covers composite signatures, the key usage bits specified here apply to all component keys within a composite key.

2.3. Composite Public Key

Composite public key data is represented by the following structure:

\[
\text{CompositePublicKey} ::= \text{SEQUENCE SIZE (1..MAX) OF SubjectPublicKeyInfo}
\]

The corresponding AlgorithmIdentifier for a composite public key MUST use the id-alg-composite object identifier, defined in Section 2.1, and the parameters field MUST be absent.

A composite public key MUST contain at least one component public key.

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

Each element of a CompositePublicKey is a SubjectPublicKeyInfo object one of the component public keys. When the CompositePublicKey must be provided in octet string or bit string format, the data structure is encoded as specified in Section 2.6.

~~~ Begin EDNOTE ~~~

EDNOTE: there has been a fair amount of discussion among the authors about whether the component public key should contain a full SubjectPublicKeyInfo for each component algorithm, or whether the \( \text{algID, and algParams} \) should be move to the params of the PUBLIC-KEY or OID, and only the BIT STRINGs of the component public key values contained in the CompositePublicKey.

Using a wonky, simplified notation, the alternatives considered were:
Current composite:

    CompositeAlg: {
        algorithm={id-alg-composite, none}
        subjectPublicKey=SEQ SPKI[{{algID1, algParams1}, value1},
                       SPKI[algID2, algParams2], value2}, ..]
    }

Alternative 1:

    CompositeAlg: {
        algorithm={id-alg-composite, {{algID1, algParams1},
                                       {algID2, algParams2}, ..}}
        subjectPublicKey=SEQ BIT STRING[value1, value2, ..]
    }

Alternative 2:

    CompositeAlg: {
        algorithm={id-alg-composite, {algID1, algID2, ..}}
        subjectPublicKey=SEQ SPKI[{{algID1, algParams1}, value1},
                                 {algID2, algParams2}, value2], ..]
    }

The authors have decided, for the time being, to use the current approach since it A) promotes ease of modifying existing software whose APIs require SubjectPublicKeyInfos to be passed, and B) avoids bloating wire protocols with duplicated information.

We note that the chosen approach means that the algorithm field essentially carries no useful information about the key it’s describing. Analysis is required to see if there are any circumstances in which this opens up cryptographic attacks, such as algorithm substitution or stripping attacks.  ~~~ End EDNOTE ~~~

2.4. Composite Private Key

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

CompositePrivateKey ::= SEQUENCE SIZE (1..MAX) OF OneAsymmetricKey

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

The corresponding AlgorithmIdentifier for a composite private key MUST use the id-alg-composite object identifier, and the parameters field MUST be absent.
A CompositePrivateKey MUST contain at least one component private key, and they MUST be in the same order as in the corresponding CompositePublicKey.

2.5. Composite Signature

The ASN.1 algorithm object for a composite signature is:

```asn1
sa-CompositeSignature

SIGNATURE-ALGORITHM ::= {  
   IDENTIFIER id-alg-composite  
   VALUE CompositeSignatureValue  
   PARAMS TYPE CompositeParams ARE required  
   PUBLIC-KEYS { pk-Composite }  
   SMIME-CAPS { IDENTIFIED BY id-alg-composite }  
}
```

The id-alg-composite object identifier MUST be used to identify when a signature has been created by a composite private key, and the following algorithm parameters MUST be included:

```asn1
CompositeParams ::= SEQUENCE SIZE (1..MAX) OF AlgorithmIdentifier
```

The signature’s CompositeParams sequence MUST contain the same component algorithms listed in the same order as in the associated CompositePrivateKey and CompositePublicKey.

The output of the composite signature algorithm is the DER encoding of the following structure:

```asn1
CompositeSignatureValue ::= SEQUENCE SIZE (1..MAX) OF BIT STRING
```

Where each BIT STRING within the SEQUENCE is a signature value produced by one of the component keys. It MUST contain one signature value produced by each component key, and in the same order as in the associated "CompositeParams", CompositePublicKey, and CompositePrivateKey objects.

The choice of "SEQUENCE OF BIT STRING", rather than for example a single BIT STRING containing the concatenated signature values, is to gracefully handle variable-length signature values by taking advantage of ASN.1’s build-in length fields.

2.6. Encoding Rules

Many protocol specifications will require that the composite public key, composite private key, and composite signature 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.

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.

In the interests of simplicity and avoiding compatibility issues, implementations that parse these structures MAY accept both BER and DER.

3. Composite Signature Algorithm

This section specifies the algorithms for generating and verifying composite signatures.

This algorithm addresses algorithm strength uncertainty by providing the verifier with parallel signatures from all the component signature algorithms; thus breaking the composite signature would require breaking all of the component signatures.

3.1. Composite Signature Generation

Generation of a composite signature involves applying each component algorithm’s signature routine to the input message according to its specification, and then placing each component signature value into the "CompositeSignatureValue" structure defined in Section 2.5.

The following algorithm is used to generate composite signature values.
Internet-Draft  PQ Composite Certs  July 2019

Input:
K1, K2, .., Kn  Private keys for the n component signature
               algorithms
M               Message to be signed, an octet string

Output:
S               The signature, a CompositeSignatureValue

Signature Generation Procedure:
1. Generate the n component signatures independently,
   according to their algorithm specifications.
   for i := 1 to n
       Si := Sign( Ki, M )

2. Encode each component signature S1, S2, .., Sn into a BIT STRING
   according to its algorithm specification.
   S := Sequence { S1, S2, .., Sn }

3. Output S

Since recursive composite public keys are disallowed in Section 2.3,
no component signature may itself be composite; ie the signature
generation routine MUST fail if one of the private keys K1, K2, ..,
Kn is composite with the OID id-alg-composite.

A composite signature MUST produce and include in the output a
signature value for every component key in the corresponding
CompositePublicKey.

EDNOTE1: With NIST’s position that they will standardize use-case-
specific algorithm suites, the authors are aware of potential use-
cases where a PKI entity may want to have many public keys, but only
sign with a subset for each signature. At the present time, this
draft does not allow for this because the algorithm for verifying
"subset-signatures" in a way that is secure against algorithm
stripping attacks would be very complex and prone to implementation
errors (currently, the verifier can detect omitted signatures even if
it does not recognize all the algorithm OIDs because the count will
be wrong. In a subset-signature algorithm, additional mechanisms
would be needed to specify for each component key, whether it is
meant to produce a signature or not). The draft-compliant way to
achieve a "subset-signature" behaviour would be for each PKI entity
to have multiple public keys (and certificates) with overlapping
subsets of their component keys. We welcome public opinions on
whether this is sufficient, or whether this draft should specify a
subset-signature algorithm.
EDNOTE2: The authors are also aware of a potential use-case of combining signature and KEM keys inside a single public key / certificate. This would give us back the "dual-usage key" property that was so appealing about RSA. At the present time, this draft does not allow for this because, again, the algorithm for verifying "subset-signatures" in a secure way would be very complex. We also welcome public opinions on this.

3.2. Composite Signature Verification

Verification of a composite signature involves applying each component algorithm’s verification routine according to its specification, and then outputting "Valid signature" (true) if a sufficient number of component algorithms were valid, and "Invalid signature" (false) otherwise.

In order to future-proof implementations of verifiers against evolutions in cryptographic algorithms and attacks against them, implementations SHOULD include a field-updatable policy mechanism for determining which and/or how many component algorithms must be valid in order for the composite signature as a whole to be considered valid. This section assumes the existence of such a policy mechanism, denoted as "checkPolicy(A1, A2, ..., An)" in the algorithm below. The implementation of such a policy mechanism is largely the responsibility of the verifier / client and therefore is out of scope for this document, but at a minimum, one component signature MUST be recognized and validated for the composite signature to be considered valid.

Modifications of the provided verification algorithm are permitted, so long as they are strengthening, and not weakening, this algorithm. In other words, any modified versions of this algorithm MUST return "Invalid signature" whenever the sample algorithm does, with the one exception noted below.
Input:
P    Signer’s composite public key
M    Message whose signature is to be verified, an octet string
S    Composite Signature to be verified
A    Composite Algorithm identifier

Output:
Validity      "Valid signature" (true) if the composite signature
               is valid, "Invalid signature" (false) otherwise.

Signature Verification Procedure::
1. Parse P, S, A into the component public keys, signatures,
   and algorithm identifiers
   P1, P2, .., Pn := Desequence( P )
   S1, S2, .., Sn := Desequence( S )
   A1, A2, .., An := Desequence( A )

   If Error during Desequencing, or the three sequences have
different numbers of elements, then output "Invalid signature"
and stop.

2. Check client policy to see whether A1, A2, .., An constitutes an
   acceptable combination of algorithms.
   if not checkPolicy(A1, A2, .., An), then
   output "Invalid signature"

3. Check each component signature individually, according to its
   algorithm specification.
   If any fail, then the entire signature validation fails.
   for i := 1 to n
   if not verify( Pi, M, Si ), then
   output "Invalid signature"

   if all succeeded, then
   output "Valid signature"

Since recursive composite public keys are disallowed in Section 2.3,
no component signature may be composite; ie the signature
verification procedure MUST fail if any of the public keys P1, P2,
.., Pn or algorithm identifiers A1, A2, .., An are composite with the
OID id-alg-composite.

Exception to this algorithm: There will be circumstances in which the
verifier does not have cryptographic libraries for all of the
provided component algorithms, or where the performance gains from
omitting algorithms justifies the loss of security. In these cases, an acceptable modification to this algorithm is to produce in step 2 one or more subsets of the algorithms "A1, A2, ..., An" which constitute acceptable combinations, outputting "Invalid signature" if an acceptable subset can not be found, and then in step 3 only perform verification of the necessary component algorithms.

Implementations SHOULD verify all recognized and supported algorithms, and output "Invalid signature" if the verification of any component signature fails, but MAY choose to only verify a subset of the algorithms for the reasons stated above.

4. In Practice

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

~~~ BEGIN EDNOTE ~~~

EDNOTE: 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.
- We could talk about overhead (size and processing).
- We could also discuss backwards compatibility.
- We could include a subsection about implementation considerations.

~~~ END EDNOTE ~~~

4.1. PEM Storage of Composite Private Keys

CompositePrivateKeys can be encoded to the PEM 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.

EDNOTE: Do we really need this? Isn’t it obvious?

4.2. 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-alg-composite.

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.

4.3. Cryptographic protocols

This section talks about how protocols like (D)TLS and IKEv2 are affected by this specifications. It will not attempt to solve all these problems, but it will explain the rationale, how things will work and what open problems need to be solved. Obvious issues that need to be discussed.

- How does the protocol declare support for composite signatures? TLS has hooks for declaring support for specific signature algorithms, however it would need to be extended, because the client would need to declare support for both the composite infrastructure, as well as for the various component signature algorithms.
o How does the protocol use the multiple keys. The obvious way would be to have the server sign using its composite public key; is this sufficient.

o Overhead; including certificate size, signature processing time, and size of the signature.

o How to deal with crypto protocols that use public key encryption algorithms; this document only lists how to work with signature algorithms. Encoding composite public keys is straightforward; encoding composite ciphertexts is less so – we decided to put that off to another draft.

5. IANA Considerations

The ASN.1 module OID is TBD. The id-alg-composite OID is to be assigned by IANA. The authors suggest to use the id-pkix arc for this usage:

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

6. Security Considerations

6.1. 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 structures using that algorithm are implicitly revoked.

In the composite model this is less obvious since a single public key, certificate, or signature may contain a mixture of deprecated and non-deprecated algorithms. Moreover, implementers may decide that certain cryptographic algorithms have complementary security properties and are acceptable in combination even though neither algorithm is acceptable by itself.

In Section 3.2, we specify that the signature verification routine must include a step to check that the combination of algorithms is acceptable under local policy:

2. Check policy to see whether A1, A2, ..., An constitutes a valid combination of algorithms.
   if not checkPolicy(A1, A2, ..., An), then output "Invalid signature"
While intentionally not specified in this document, implementors should put careful thought into implementing a meaningful policy mechanism within the context of their signature verification engines.

6.2. Protection of Private Keys

This structures described in this document do not protect the private keys information 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 key information 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 as ‘strong’ as the key it is protecting.

6.3. Checking for Compromised Key Reuse

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.

6.4. Composite Encryption and KEMs

This document deals only with signature keys. While the CompositePublicKey and CompositePrivateKey structures could equally be used to hold encryption or KEM keys, the authors warn that there are non-trivial design decisions to be made when constructing a multi-key public key encryption or KEM algorithm. Some of these design and implementation decisions, if done incorrectly will result in a catastrophic loss of security. We leave it to the community to standardize analogous composite encryption and KEM schemes.

7. Appendices

7.1. ASN.1 Module

<CODE STARTS>

Composite-Signatures-2019

{ TBD }

DEFINITIONS IMPLICIT TAGS ::= BEGIN
EXPORTS ALL;

IMPORTS
  PUBLIC-KEY, SIGNATURE-ALGORITHM
  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
--

  id-alg-composite OBJECT IDENTIFIER ::= { TBD }

--
-- Public Key
--

  pk-Composite PUBLIC-KEY ::= { IDENTIFIER id-alg-composite
    KEY CompositePublicKey
    PARAMS ARE absent
    CERT-KEY-USAGE
      { digitalSignature, nonRepudiation, keyCertSign, cRLSign }
    PRIVATE-KEY CompositePrivateKey }

CompositePublicKey ::= SEQUENCE SIZE (1..MAX) OF SubjectPublicKeyInfo

CompositePrivateKey ::= SEQUENCE SIZE (1..MAX) OF OneAsymmetricKey

--
-- Signature Algorithm
--
7.2. Intellectual Property Considerations

The authors are aware that Massimiliano Pala and CableLabs have applied for Intellectual Property Rights around composite key, signatures, and certificates. We have a verbal agreement with Max that this IP will be made freely available to the community.

As of this version of the draft, the authors have reviewed and provided feedback on the March 24, 2019 version of the IPR disclosure, available at https://datatracker.ietf.org/ipr/3481/, and are awaiting the posting of an updated version that covers this draft.

EDNOTE: remove this section once the IPR disclosure is posted and tagged against this draft.

8. 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 Datacard), Serge Mister (Entrust Datacard), Scott Fluhrer (Cisco Systems), Panos Kampanakis (Cisco Systems), Daniel Van Geest (ISARA), and Tim Hollebeek (Digicert).

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].

9. References

9.1. Normative References


9.2. Informative References

[I-D.pala-composite-crypto]

[I-D.truskovsky-lamps-pq-hybrid-x509]


Authors’ Addresses

Mike Ounsworth
Entrust Datacard Limited
1000 Innovation Drive
Ottawa, Ontario K2K 1E3
Canada

Email: mike.ounsworth@entrustdatacard.com

Massimiliano Pala
CableLabs

Email: director@openca.org