LAMPS
Internet-Draft
Intended status: Standards Track
Expires: September 9, 2019

M. Ounsworth
S. Mister
J. Gray
Entrust Datacard
S. Fluhrer
P. Kampanakis
Cisco Systems

March 08, 2019

Composite Keys and Signatures For Use In Internet PKI
draft-ounsworth-pq-composite-sigs-00

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, CompositeSignatureAlgorithmParams, and CompositeSignatureValue 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 September 9, 2019.

Table of Contents

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. Even after the transition period, a composite approach may be advantageous as a single entity may have multiple public keys on different algorithms or strengths to address different use-cases, and a single signature may want to compase multiple of them together.

The deployment of composite public keys and 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. 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, but where specific design decisions needed to be made, the authors chose the variant that caused the least disruption to existing X.509 certificates, as defined in [RFC5280].

EDNOTE: While the scope of this document is restricted to signatures, we note that the same 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 than; a naive implementer might be tempted to follow the same pattern as bellow and encrypt the message with each public key separately and then concatenate the ciphertexts, which is wrong. 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."CompositePublicKey"

2. 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.

3. Definitions and notation

3.1. Definitions

EDNOTE: A glossary of terms we define for this document, or terms that we borrow from other RFCs.

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.

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 {{sec-composite-structs}}.

3.2. Notation

No special notation is used in this document.

4. Composite Structures

In order for public keys and signatures to be composed of multiple algorithms, the respective structures defined in [RFC2986], [RFC5280] (AND OTHERS??) need to be extended. We define encodings of sequences of public keys and signature data which consist of a sequence of public keys and signatures from more basic signature algorithms (aka "component algorithms") such that these structures can be places into any existing public key or signature structure.

This section defines

- Composite public key: A general structure for holding multiple public keys within a single public key data structure.

- Composite signature: Data structures needed to make use of the Composite Signature signature algorithm (defined in Section 5), which encapsulates signatures made with multiple public keys.

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

4.1. Composite Public Key

A composite public key is a sequence of component public keys that are used together. A composite public key is identified by the object identifier

id-ce-compositePublicKey OBJECT IDENTIFIER ::= { OID }

The parameters field for this public key type MUST be absent. The composite public key data is represented by the following structure:

CompositePublicKey ::= SEQUENCE OF SubjectPublicKeyInfo

where each element of the sequence is a "SubjectPublicKeyInfo" of a public key that MAY be used in conjunction with the other keys in the sequence. When the public key must be provided in octet string or bit string format, the data structure is converted as specified in Section 4.3.

EDNOTE: This document does not define the corresponding private key formats because the authors deemed it to be out of scope. We do note that the draft the Max Pala on a similar topic, does define private key formats, so there could be scope to merge these submissions.[I-D.pala-composite-crypto]

4.2. Composite Signature Algorithm

The Composite Signature signature algorithm defined in Section 5 is identified by the following object identifier:

id-ce-compositeSignature OBJECT IDENTIFIER ::= { OID }

The following algorithm parameters MUST be included when this identifier is used:

CompositeSignatureAlgorithmParams ::= SEQUENCE OF AlgorithmIdentifier

When a composite signature is generated by a key with a CompositePublicKey, the signature's CompositeSignatureAlgorithmParams sequence MUST contain the same component algorithms listed in the same order as in the associated CompositePublicKey.

The Composite Signature algorithm output is the DER encoding of the following structure:

id-ce-CompositeSignatureValue OBJECT IDENTIFIER ::= { OID }

CompositeSignatureValue ::= SEQUENCE OF BIT STRING

Where each bit string within "CompositeSignatureValue" is a signature by one of the component signature algorithms.

The choice of "SEQUENCE OF BIT STRING" rather than "BIT STRING" is so the type-length-value encoding can solve the problem of variable-length signature values. The signature's "CompositeSignatureValue" sequence MUST contain the same component algorithms listed in the same order as in the associated "CompositeSignatureAlgorithmParams".

4.3. Encoding Composite Structures As Octet Strings and Bit Strings

Many specifications require that the composite public 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.

5. Composite Signature Algorithm

The Composite Signature signature algorithm generates a single composite signature by using multiple private keys to apply multiple signature algorithms to the input message, with the resulting signature effectively being the concatenation of the individual signature values.

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

5.1. Composite Signature Generation

The following algorithm is used to generate composite signature values.

Input:

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

Output:

S	Signature, an octet string

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. DER encode the component signatures into an ASN.1 value of type Signature, where the type Signature has the syntax
   Signature ::= Sequence { S1, S2, ..., Sn }
   Let S be the DER encoding of the Signature
3. Output S

5.2. 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.

Implementations MAY include policy mechanisms for determining which and how many component algorithms must be valid in order for the composite signature to be considered valid. See Section 6 for further discussion of possible standardization of such mechanisms. This section assumes the existence of such a policy mechanism, specifically one that allows revocation of individual component algorithms.

This section provides a sample algorithm for validating composite signatures. Compliant implementations MUST return "Invalid signature" whenever the sample algorithm does, but MAY require more than one signature to be valid.

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, 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 each signature individually

```
V1, V2 , ..., Vn := BOOLEAN
for i := 1 to n
   Check if Ai is a recognized algorithm,  if so then,
     Check if algorithm Ai has been revoked, if not then,
       Verify the component signature according to the component
       algorithm's specification
         Vi = verify( Pi, M, Si )
```

3. Check policy to see whether V1, V2, ..., Vn constitutes a valid signature

```
if isValid(V1, V2, ..., Vn), then
  output "Valid signature"
else
  output "Invalid signature"
```

If "V1 = V2 = ... = Vn = false" for all "Vi", then "isValid(V1, V2, ..., Vn)" MUST return false. Implementations MAY include additional policy mechanisms as discussed in Section 6.

6. Mechanisms to distribute verification policy to clients

In the traditional world of single-key public keys and signatures, the semantics of a signature and a verification are straight-forward: if the key is trusted (via public key pinning, a PKIX revocation check, etc) and the signature is valid, then the signed content can be trusted. However the semantics are less obvious in a world where public keys and signatures are composed of two or more algorithms; it is conceivable that even though one component algorithm fails verification, for example because the algorithm is revoked, a multi-algorithm signature may contain enough other trustworthy component algorithms to still be considered valid.

This section addresses how a verifier can obtain policy information for which and/or how many component algorithms must be valid in order for the signature as a whole to be valid. The authors ask for community feedback about whether this needs to be specified, and if so, how best to do it.

This section lists rough outlines for several such mechanisms that have come up in discussion during the drafting of this document. They are mainly focused around X.509 PKIs, and provided here merely for the purposes of sparking debate. The authors believe that by specifying such a mechanism, the world will be able to more quickly react to news of algorithm compromise with a lower service disruption compared to the need to revoke and re-issue all certificates using that algorithm. However, we are not sure if the gains justify the added complexity.

6.1. Local verifier policy

Much as we do today, this is left up to domain administrators and software vendors to implement the guidance of governing bodies on a system-by-system basis.

6.2. Extra metadata in the public key or signature

This policy information could be specified by the signer at signing time. Depending on the structure of the databeing signed, this metadata could go into the public key, or an extension to the signature, or some other field provided that it is inside the signed data blob.

6.3. Extra metadata in the certificate

This policy information could be included in a certificate via an X.509 v3 extension. This gives the Certificate Authority control, but has the drawback that updating the policy requires revoking and reissuing certificates.

6.4. Policy certificate issued by the Certificate Authority

Certificate Authorities have the ability to issue policy certificates that specify the behaviour when verifying signatures performed by keys in certificates within the scope of the policy certificate.

This method has the advantage that policy is centrally-managed, and can be updated without needing to reissue any certificates, but has the drawback that not all PKI implementations support policy certificates.

6.5. Policy constraints in a cross-certificate

This method behaves similarly to the policy certificate method above, but has better support across PKI implementations.

6.6. Revoked Algorithms CRL Extension

Add an extension to CRLs so that in addition to revoking certificates, they can also revoke algorithms for all certificates within the scope of that CRL. Implemented with care, this could allow a single PKI to do a staged algorithm migration by only revoking the algorithm for one CRL group at a time.

```
id-ce-RevokedAlgorithms OBJECT IDENTIFIER ::= { OID }

RevokedAlgorithms ::= SEQUENCE OF SEQUENCE  {
     algorithms              AlgorithmIdentifier,
     revocationDate          Time,
     crlEntryExtensions      Extensions OPTIONAL
                              -- if present, version MUST be v2
     }
```

EDNOTE: do we need the crlEntryExtensions field? If so, which ones from https://tools.ietf.org/html/rfc5280#section-5.3 are allowed here?

There may only be one "RevokedAlgorithms" extension in a CRL. This extension is OPTIONAL. If a CRL contains only composite certificates, then this extension SHOULD be designated as critical.

If a CRL contains a mixture of composite and traditional certificates then it SHOULD be designated as non-critical.

If the Revoked Algorithms extension is present in a CRL, then a client performing a certificate validation on an otherwise non-revoked certificate within the scope of that CRL MUST skip any signatures corresponding to a revoked algorithm; thus a certificate is valid only if it would have been valid had those Algorithm IDs and Signature Values been omitted from the certificate.

Once a algorithm has been marked as revoked on a given CRL, it MUST remain revoked on subsequent CRLs.

EDNOTE: Is there corresponding wording about cert serial numbers on CRLs from RFC5280? Or is this unnecessary implied?

Note that a similar mechanism could be used on a per-certificate basis via CRL Entry Extensions, however the authors believe that giving operators the ability to perform partial revocation of a certificate (ie revoking some keys or signatures but leaving the certificate as a whole valid) will greatly increase the complexity of certificate validation routines, thus increasing the chance of both human error, and implementation bugs leading to vulnerabilities, without providing a commensurate amount of increased functionality. By not defining a new CRL Entry Extension, the following requirement is implied: if any key within a certificate warrant revocation, the entire certificate MUST be revoked using the existing revocation mechanisms (this does not apply when the algorithm is globally revoked for the entire scope of this CRL).

6.6.1. Implicit Revocation

A Composite Signature Algorithm is considered to be "implicitly revoked" if the certificate is otherwise valid but one of the following conditions are met.

- A certificate using a single-key algorithm which is revoked within the scope of its CRL. In this case, signature verification SHOULD fail when performed by a compliant client, but of course will succeed when performed by a legacy client which is not aware of this CRL extension.

- All of the component algorithms are revoked within the scope if its CRL. In this case, signature verification MUST fail when performed by a compliant client, regardless of which verification algorithm is used.

At the time of an algorithm revocation, a certificate authority MAY revoke certificates meeting one ofd the above criteria (by placing them in the traditional "revokedCertificates" list) with a revocation reason of "keyCompromise". OCSP responders SHOULD designate a certificate as revoked if it meets the above condition.

7. New Algorithm Identifiers

EDNOTE: This subsection will define the OIDs for the initial composite algorithm combinations we want to define. These are the OID that Section 10 will ask for IANA to assign.

8. In Practice

EDNOTE: This section will talk about practical issue of how these certificates will be used. For example it will talk about the size of these certs and cert chains. It will explain that if a cert in the chain is a Composite cert then the whole chain needs to be of Composite Certs. It will 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. It will talk about overhead (size and processing). It will also discuss backwards compatibility. It could include a subsection about implementation considerations.

9. Implications for existing standards

9.1. RFC 2986

EDNOTE: summarize the updates to RFC 2986 (CSR / PKCS#10).

9.2. RFC 5280

EDNOTE: summarize the updates to RFC 5280 (X.509).

9.3. 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.

- 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.

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

- 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.

10. IANA Considerations

EDNOTE: This section will include content only if new OIDs or IANA codepoints are assigned for it.

11. Security Considerations

EDNOTE: This section includes the Security Considerations.

- CA implementations need to be careful when checking for compromised key reuse, for example as required by WebTrust regulations; unpack the CompositePublicKey structure and compare individual keys.

- The corresponding multi-key encryption routine is considerably more prone to implementation errors that will result in a catestrophic loss of security, as compared with the signature algorithms specified in this document.

12. Appendices

12.1. Intellection Property Considerations

The authors claim no IPR associated with any of the content of this draft.

12.2. Comparison with draft-truskovsky-lamps-pq-hybrid-x509

EDNOTE: This section will explain the differences from . IPR Claims should be mentioned here if necessary. Other things to consider are the things like simplicity and format, inadvertnt implementation errors, algorithm agility.[I-D.truskovsky-lamps-pq-hybrid-x509]

13. Contributors

This document incorporates contributions and comments from a large group of experts. The Editor would especially like to acknowledge the tireless dedication of the following people, who attended many long meetings and generated millions of bytes of electronic mail over the past 3 years months in pursuit of this document:

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

14. Acknowledgenents

EDNOTE: this section include all those that need to be acknowledged in the draft

15. References

15.1. Normative References

[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>.

[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>.

15.2. Informative References

[I-D.pala-composite-crypto] Pala, M., "Composite Public Keys and Signatures", draft-pala-composite-crypto-00 (work in progress), February 2019.

[I-D.truskovsky-lamps-pq-hybrid-x509]
Truskovsky, A., Geest, D., Fluhrer, S., Kampanakis, P., Ounsworth, M., and S. Mister, "Multiple Public-Key Algorithm X.509 Certificates", draft-truskovsky-lamps-pq-hybrid-x509-01 (work in progress), August 2018.

Authors' Addresses

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

Email: mike.ounsworth@entrustdatacard.com

Serge Mister
Entrust Datacard Limited

Email: serge.mister@entrustdatacard.com

John Gray
Entrust Datacard Limited

Email: john.gray@entrustdatacard.com

Scott Fluhrer
Cisco Systems

Email: sfluhrer@cisco.com

Panos Kampanakis
Cisco Systems

Email: pkampana@cisco.com