Certificate Format

David Stainton

Table of Contents

Abstract

This document proposes a certificate format that Katzenpost mix server, directory authority server and clients will use.

1. Introduction

Mixes and Directory Authority servers need to have key agility in the sense of operational abilities such as key rotation and key revocation. That is, we wish for mixes and authorities to periodically utilize a long-term signing key for generating certificates for new short-term signing keys.

Yet another use-case for these certificate is to replace the use of JOSE RFC7515 in the voting Directory Authority system KATZMIXPKI for the multi-signature documents exchanged for voting and consensus.

1.1 Conventions Used in This Document

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in RFC2119.

1.2 Terminology

Tbw…

2. Document Format

The CBOR RFC7049 serialization format is used to serialize certificates:

Signature is a cryptographic signature which has an associated signer ID.

```
type Signature struct {
        // Identity is the identity of the signer.
        Identity []byte
        // Signature is the actual signature value.
        Signature []byte
}
```

Certificate structure for serializing certificates.

```
type certificate struct {
    // Version is the certificate format version.
    Version uint32

    // Expiration is seconds since Unix epoch.
    Expiration int64

    // KeyType indicates the type of key
    // that is certified by this certificate.
    KeyType string

    // Certified is the data that is certified by
    // this certificate.
    Certified []byte

    // Signatures are the signature of the certificate.
    Signatures []Signature
}
```

That is, one or more signatures sign the certificate. However the `Certified` field is not the only information that is signed. The `Certified` field along with the other non-signature fields are all concatenated together and signed. Before serialization the signatures are sorted by their identity so that the output is binary deterministic.

2.1 Certificate Types

The certificate `type` field indicates the type of certificate. So far we have only two types:

- identity key certificate
- directory authority certificate

Both mixes and directory authority servers have a secret, long-term identity key. This key is ideally stored encrypted and offline, it's used to sign key certificate documents. Key certificates contain a medium-term signing key that is used to sign other documents. In the case of an "authority signing key", it is used to sign vote and consensus documents whereas the "mix singing key" is used to sign mix descriptors which are uploaded to the directory authority servers.

2.2. Certificate Key Types

It's more practical to continue using Ed25519 ED25519 keys but it's also possible that in the future we could upgrade to a stateless hash based post quantum cryptographic signature scheme such as SPHINCS-256 or SPHINCS+. SPHINCS256

3. Golang API

- https://godoc.org/github.com/katzenpost/katzenpost/core/crypto/cert

Our golang implementation is agnostic to the specific cryptographic signature scheme which is used. Cert can handle single and multiple signatures per document and has a variety of helper functions that ease use for multi signature use cases.

4. Acknowledgments

This specification was inspired by Tor Project's certificate format specification document:

- https://gitweb.torproject.org/torspec.git/tree/cert-spec.txt

Appendix A. References

Appendix A.1 Normative References

Appendix A.2 Informative References

Appendix B. Citing This Document

Appendix B.1 Bibtex Entry

Note that the following bibtex entry is in the IEEEtran bibtex style as described in a document called "How to Use the IEEEtran BIBTEX Style".

```
@online{KatzenCert,
title = {Certificate Format Specification},
author = {David Stainton},
url = {https://github.com/katzenpost/katzenpost/blob/master/docs/specs/certificate
year = {2018}
}
```

ED25519

- RFC8032 [https://tools.ietf.org/html/rfc8032]

KATZMIXPKI

```
Angel, Y., Piotrowska, A., Stainton, D.,
"Katzenpost Mix Network Public Key Infrastructure Specification",
December 2017,
https://github.com/katzenpost/katzenpost/blob/master/docs/specs/pki.md
```

RFC2119

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

RFC7049

C. Bormannm, P. Hoffman,
"Concise Binary Object Representation (CBOR)",
Internet Engineering Task Force (IETF),
October 2013,
https://tools.ietf.org/html/rfc7049

RFC7515

Jones, M., Bradley, J., Sakimura, N.,
"JSON Web Signature (JWS)",
May 2015,
https://tools.ietf.org/html/rfc7515

RFC7693

Saarinen, M-J., Ed., and J-P. Aumasson,
"The BLAKE2 Cryptographic Hash and Message Authentication Code (MAC)",
RFC 7693, DOI 10.17487/RFC7693,
November 2015,
http://www.rfc-editor.org/info/rfc7693

SPHINCS256

Bernstein, D., Hopwood, D., Hulsing, A., Lange, T., Niederhagen, R., Papachristodo
"SPHINCS: practical stateless hash-based signatures",
http://sphincs.cr.yp.to/sphincs-20141001.pdf