Hybrid Certificates in OpenSSL

Luca Gladiator

March 20, 2019

Introduction

Hybrid certificates [1] are a concept for using two different signature schemes to sign one certificate, thereby enabling the parallel usage of two different schemes for authentication and key exchange purposes. This has the advantage of having the security of both signatures for the certificate. They are meant to be used in a transition to post-quantum cryptography, where a classical signature scheme is combined with a new post-quantum scheme. This allows using the new scheme while still retaining at least the security of the old scheme, if the new scheme turns out to be unsafe. This is the documentation of the implementation of hybrid certificates for OpenSSL [2].

In this approach hybrid certificates are realized using the standard signature and public key fields of the X.509 [3] certificate for one of the signature schemes, which allows compatibility with clients not supporting hybrid signatures. The second signature scheme is integrated using two non critical X.509 extensions. One of the extensions contains the public key of the second scheme, while the other contains the second signature on the certified data. Figure 1 shows the resulting certificate structure. The standard signature is also called the outer signature in the rest of this document, while the one in the extensions is called the inner signature.

Two other approaches have been considered. The first one would have been two concatenate the keys and signatures, like it is already implemented in the Open Quantum Safe project [4]. This has the disadvantage of not providing backwards compatibility for clients not understanding the concatenated keys. The second one would have been to include a second standard certificate signed with a different scheme in an extension. This has the disadvantage of duplicating information in the inner and outer certificate and unnecessarily increasing the size of the resulting certificate.

The next two sections describe the two extensions that were implemented. Afterwards the signing and verification process is described and finally the security of this approach is discussed.

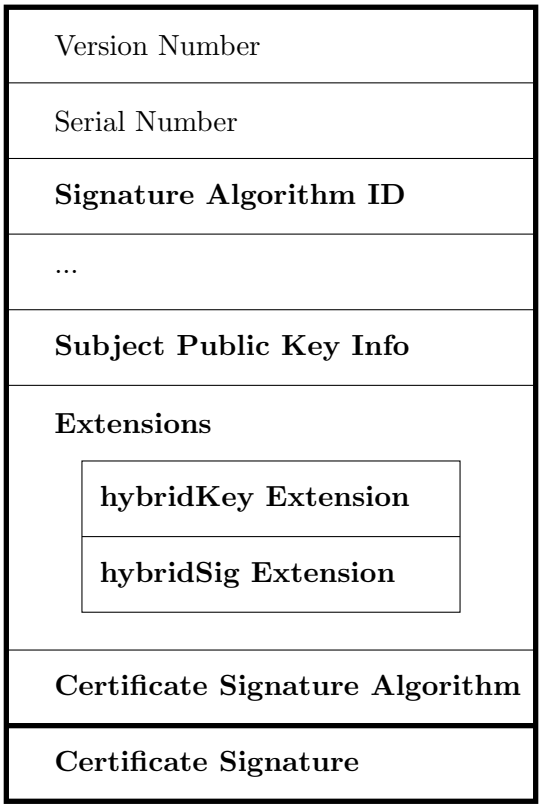

Figure 1: X.509 v3 structure

The 'hybridKey' extension

The 'hybridKey' extension is defined as a non critical extension which fulfills the job of storing the public key info of the inner signature scheme. It has the following ASN1 structure:

```
id-ce-hybridKey OBJECT IDENTIFIER ::= { id-ce 211 }

HybridKey ::= SubjectPublicKeyInfo
```

The object id was chosen to be in the certificate extension id group, but has not been registered yet. `SubjectPublicKeyInfo` is the structure defined in RFC5280 [3, p. 16]. This structure was chosen because it contains all relevant information for a public key, which is exactly what is required by this extension.

The implementation can be found in the file `crypto/x509v3/v3_hybridkey.c`. Apart from the extension structure definitions it contains the following three functions:

The function `v2i_HybridKey` is used when adding this extension to a certificate via the command line option `-addext "hybridKey=file:path/to/pubkey"`, where `path/to/pubkey` is the path to the public key that should be added.

The function `i2r_HybridKey` is used to output a textual representation of the extension.

The function `X509_get_hybrid_key` is used to retrieve the public key of the inner signature scheme from a certificate, if it contains a 'hybridKey' extension. Otherwise it returns a null pointer.

The 'hybridSig' extension

The 'hybridSig' extension is defined as a non critical extension which contains the signature of the inner signature scheme. It has the following ASN1 structure:

```
id-ce-hybridKey OBJECT IDENTIFIER ::= { id-ce 212 }

HybridSig ::= SEQUENCE {
    algorithm    AlgorithmIdentifier,
    signature    BIT STRING }
```

The object id was chosen to be in the certificate extension id group, but has not been registered yet. `AlgorithmIdentifier` is the structure defined in RFC5280 [3, p. 17]. This structure was chosen because a signature needs information on which algorithm is used, which is provided by the `algorithm` field, and the actual bit string of the signature, which is provided by the `signature` field.

In this structure the field `algorithm` shows which algorithm is used for the signature and `signature` is a bit string containing the actual signature. The signature is generated by signing the entire TBS [3, p. 17] part of the certificate with the only difference, that the signature bit string is replaced by a bit string of the same length with all bits set to zero. This approach was chosen, because the length of the bit string has effects on different parts of the ASN1 encoding, which would have to be undone for the verification of the inner signature if this extension was left out when creating the inner signature.

The implementation can be found in the file `crypto/x509v3/v3_hybridsig.c`. This file contains the structure definition for this extension and a few helper functions.

The function `v2i_HybridSig` is used when adding this extension to a certificate via the command line option `-addext "hybridSig=file:path/to/key"`, where `path/to/key` is the path to the key that should be used when signing with the second signature scheme. It does not perform the signing itself, but rather adds a dummy signature extension, containing just zeroes for the signature bit string, so the structure can be used to calculate the actual signature. It also stores the key, so it can be used to create the signature later on, when the entire certificate is created.

The function `i2r_HybridKey` is used to output a textual representation of the extension.

The function `HYBRID_SIGNATURE_sign` uses the key saved in `v2i_HybridSig` to create the actual signature of the inner scheme and replace the bit string of zeroes with this signature.

The function `HYBRID_SIGNATURE_verify` checks whether a certificate contains a valid inner signature for a given public key.

The function X509v3_hybrid_sig_validate_path checks whether the inner signatures of the chain form a valid signature chain. It uses the function hybrid_sig_validate_path_internal, which in turn calls HYBRID_SIGNATURE_verify for each certificate in the chain.

The function create_dummy_extension is used to create a 'hybridSig' extension with zeroes for the signature bit string. This is used when generating and verifying the signature of the inner scheme (Because the signature extension has to contain a bit string of just zeroes for the signature calculations).

Certificate Signing

For signing certificates the function X509_sign_ctx in the file crypto/x509/x_all.c was adapted to call HYBRID_SIGNATURE_sign prior to actually signing the certificate. This ensures that the inner signature is performed first and the outer signature also authenticates this inner signature.

The signing process is illustrated in Figure 2. In step 1, the certificate is constructed as normal with the addition of adding the hybrid extensions. Because the HYBRID_SIGNATURE_sign function requires all fields of the certificate, except for the outer signature, to be set to their final value, ASN1_item_sign_ctx is actually called once before it, because it sets the Signature Algorithm ID and Certificate Signature Algorithm fields while signing (step 2). This was an implementation choice to avoid duplicating code of that function and possibly should be optimized to only set these fields and not perform the signing.

Next the inner signature is generated using the function HYBRID_SIGNATURE_sign (step 3) and assigned to the bit string of the **hybridSig** extension (step 4). The signature generated in the first call to ASN1_item_sign_ctx from step 2 can not be used for the final outer signature, because the TBS [3, p. 17] part of the certificate changed when the generated inner signature was written into the extension in step 4. So HYBRID_SIGNATURE_sign is called again in order to get the correct outer signature (step 5) and assign it to the certificate (step 6).

Hybrid Path Validation

The path verification process is shown in Figure 3. In step 1 the standard chain verification is performed, which checks the outer signatures. To verify the inner signatures in a chain of certificates, a call to X509v3_hybrid_sig_validate_path was added to the function verify_chain in the file crypto/x509/x509_vfy.c. This results in a second walk of the chain after the standard verification walk which verifies the inner signatures (step 2). During this second walk the function HYBRID_SIGNATURE_verify is used to check each single inner signature. After the verification, 1 is returned if the verification succeeded and 0 is returned otherwise (step 3). For a chain to be considered correct, the following conditions MUST be fulfilled:

1. standard path validation MUST succeed

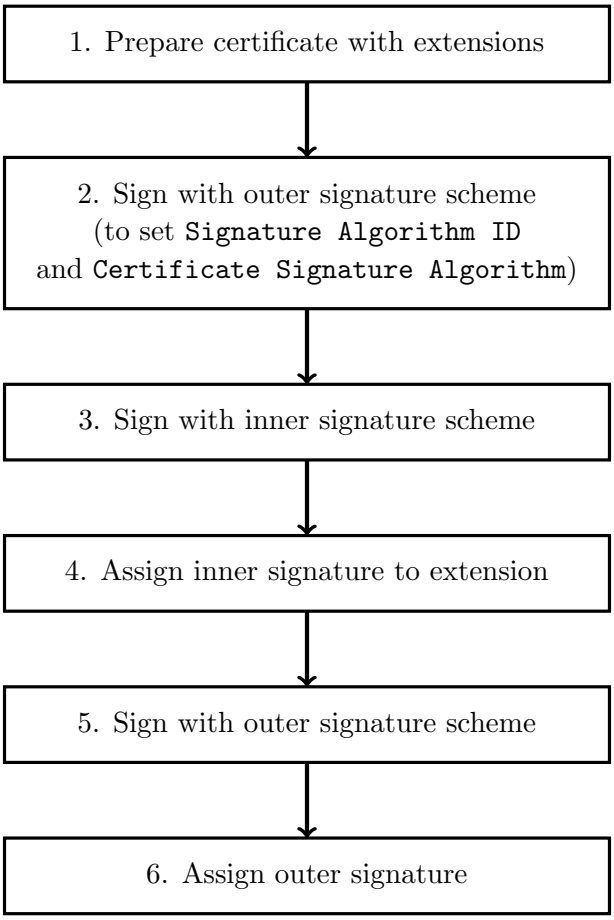

Figure 2: Certificate signing process

2. each child of a certificate that contains a **hybridKey** extension MUST contain a correct **hybridSig** extension
3. if a certificate contains a **hybridKey** extension, the parent certificate MUST contain a **hybridKey** extension and a **hybridSig** extension
4. each certificate with a **hybridKey** extension MUST contain a **hybridSig** extension as well

With these restrictions the following scenarios are legal:

- A chain that contains no **hybridKey** and **hybridSig** extension.

  This means legacy chains are legal.

- A chain that contains a **hybridKey** and **hybridSig** extension in each certificate.

  This is the desired case, having a complete hybrid chain.

- A chain that starts with certificates containing **hybridKey** and **hybridSig** extensions until there is one certificate with only a **hybridSig** extension. All children from there on contain neither a **hybridKey** nor **hybridSig** extension.

  This scenario might be used, where a hybrid CA issues a classical certificate for a client that can not handle hybrid signatures. Since the client does not know about the hybrid extensions, it will not check the inner signatures, but it can still verify the classical outer signatures in the chain and a client understanding hybrid certificates will also accept this chain.

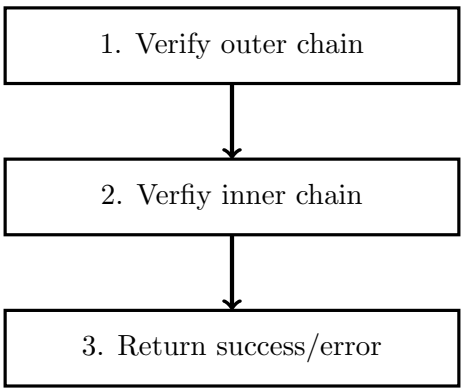

Figure 3: Hybrid path validation process

Security

Because the extensions are non critical, a client that does not support them achieves the standard security properties of the outer signature chain without the benefits of the inner signatures. For clients that are aware of the extensions, condition 1 ensures they achieve at least the standard security properties of the chain. This means if only the inner signature scheme is broken, the outer scheme still protects the chain. Condition 2 ensures, that an adversary who can forge only the outer signatures can not forge a chain starting from a hybrid CA, because he would also need to forge the inner signature. Condition 3 ensures, that if a leaf certificate contains a hybrid key, the entire chain must be hybrid. This prohibits a leaf certificate from pretending to have hybrid security, when the chain does not give this security. Condition 4 ensures, that the root CA must self sign its certificate not only with the outer but also the inner signature scheme.

Client Usage

Clients that wish to use the hybrid key not only for the certificate verification but for further usage can extract the inner key with the function `X509_get_hybrid_key` and use

it together with the standard public key of the certificate in order form a hybrid scheme. Alternatively the function `X509_extract_pubkey` can be used to retrieve a concatenated key consisting of the outer and inner key.

Further Work

Currently the OpenSSL TLS implementation only uses the key returned by `X509_extract_pubkey` which is not backwards compatible with legacy clients. The TLS server could be extended to choose between the legacy or hybrid scheme based on the cipher suites submitted by the client.

The implementation supports certificate signing requests containing a `hybridKey` extension to request a certificate with this extension. However, currently there is Proof-of-Possession implemented for this inner key. This could be solved by using a self signed `hybridSig` extension for CSRs as a Proof-of-Possession that is checked by the CA but not copied into the issued certificate.

References

[1] Nina Bindel, Udyani Herath, Matthew McKague, and Douglas Stebila. Transitioning to a quantum-resistant public key infrastructure. Cryptology ePrint Archive, Report 2017/460, 2017. https://eprint.iacr.org/2017/460.

[2] https://www.github.com/CROSSINGTUD/openssl-hybrid-certificates.

[3] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk. Internet x.509 public key infrastructure certificate and certificate revocation list (crl) profile. RFC 5280, RFC Editor, May 2008. http://www.rfc-editor.org/rfc/rfc5280.txt.

[4] https://www.github.com/open-quantum-safe.