Intelligent Composed Algorithms

Frank Byszio [1]
Dr. Klaus-Dieter Wirth [2]
Dr. Kim Nguyen [3]
(D-Trust GmbH, Berlin, Germany)

[1] Frank.Byszio@bdr.de [2] K.Wirth@d-trust.net [3] Kim.Nguyen@bdr.de

March 05, 2021

Summary

Intelligent Composed Algorithms (ICA) have been developed as a mechanism for introducing new cryptographic algorithms into applications and PKIs. Using ICAs, known cryptographic algorithms (Component-Algorithms) can be combined in order to obtain a stronger mix of cryptographic algorithms or primitives. Using ICAs it is also possible to use known Component-Algorithms as mutual alternatives. Furthermore, the combined and alternative use of Component-Algorithms as ICAs shall enable agile use of cryptographic algorithms without having to change standards as X.509 or CMS.

An Intelligent Composed Algorithm is a flexible group of cryptographic algorithms together with the corresponding rules for their combination. The rules for the combination of Component-Algorithms are defined as algorithms (Controlling-Algorithms) themselves. In applications, ICAs are used as conventional algorithms, described by an algorithm identifier (an OID) and matching parameters. The chosen Component-Algorithms are defined by parameters of the Controlling-Algorithm.

The use of ICAs impose no need to modify higher-order standards for applications and protocols, as X.509, RFC 5280, RFC 6960, RFC 2986, RFC 4210, and RFC 5652.

Introduction

D-Trust is a member of the Bundesdruckerei Group. As an independent and certified trust service provider, D-Trust operates several PKIs. These PKIs include the PKIs for the German identity card, the PKIs for German passports, state-regulated PKIs according to eIDAS [1], publicly verifiable PKIs according to the rules of CAB/F [2] [3], as well as large PKIs for companies. In the past, PKIs had to be migrated regularly due to new or changed algorithms. Among other things, the development of computing power has forced the introduction of improved algorithms and larger keys in PKIs or algorithms such as RIPEMD-160 hade to be phased out. Examples are the change from SHA-1 to SHA-2, the change from RSA PKCS-1 V1.5 to RSA-PSS, or the increase of RSA key lengths from 1024 over 2048 to 3072 and 4096 bits today. New applications also require new algorithms. The introduction of ECDSA was largely motivated and partially enforced by small and embedded devices.

Changes to a PKI represent a high effort for the operators. Many PKIs have a lifespan of 10, 20 or more years, depending on their purpose and area of application. During this time, business continuity is the most important requirement. The operators must ensure the safety and availability of the PKIs during this time. Therefore, changes to a PKI must be done during ongoing operation or, in the worst case, need a seamless migration to a new PKI.

Challenging when migrating a PKI is that changes in algorithms and keys do not only affect their own certificates, CA management systems, OCSP responders, CRLs, and HSMs. The systems of the customers (end entities) and their partners (relying parties) are also affected. A PKI operator usually does not necessarily have a direct relationship with Relying Parties. Thus, a change in a PKI is usually introduced through communication with software manufacturers. Such a process usually takes many years. This means that short-term threats due to identified weaknesses in algorithms are difficult to respond to adequately and the topic of backward compatibility is always discussed.

In the future, it is expected that migrations due to changes in algorithms or key lengths will occur more frequently and for larger PKIs. Cryptoagility and agility of PKIs are essential for future processes of PKI operation. A major driver for this expectation is the development in the field of quantum computing. Future quantum computers have the potential to make the asymmetric signature and encryption algorithms used today insecure.

Various organizations have started processes to select successors of the currently used asymmetric algorithms that can withstand attacks using quantum computers.

In 2016, NIST has started a process for the development and selection of new cryptographic standards for algorithms that are resistant to attacks by quantum computers [4]. Currently, as planned, the 3rd round [5] in the selection process of the competition is ongoing. Four candidates for encryption / KEM as well as three candidates for digital signatures were selected for round 3. Further five alternative candidates for encryption / KEM and three alternative candidates for digital signatures suggest that NIST expects a greater variety of algorithms in PKIs and applications in the future.

One prominent project in the scope of the EU research programme Horizon 2020 has been the project pqcrypto [6]. Various algorithms for small devices (WP1), the Internet (WP2) and cloud applications (WP3) have been examined in three work packages.

The Chinese Association for Cryptologic Research has launched its post-quantum cryptography competition in 2018 and has completed the competition end of 2019 with the selection of three first, three second, and five third prizes [7].

In summary, all current projects or investigations avoid the statement that a single algorithm will be the new successor to the existing algorithms. None of the current candidates meets the requirements of "performance", "small keys", "small signatures", "flexible use" and "proven mathematical security". Therefore, we assume that there will be several successors and mixed forms of known and new algorithms. Furthermore, we assume that algorithm changes in PKIs will occur more frequently in the future. Most candidates under consideration are relatively young and require even more time for thorough cryptographic analysis. Nevertheless, the new algorithms will have to be introduced, as the development of the technology requires high-performance, quantum computer-resistant algorithms, with - depending on the application - small keys or flexible use. Agile PKIs are likely to become much more important in the future.

Since the advantages and disadvantages of classical algorithms on the one hand and new algorithms on the other hand are very different, it is interesting to consider if hybrid solutions that arise from combination of these algorithms might be robust against all possible attacks, see e.g.[8]. Such solutions are currently discussed and implemented in prototypes. Two of the most noted implementations for key negotiation in TLS has been done by Google with CECPQ1 [9] and CECPQ2 [10]. The well-known ECDH was combined with some new lattice-based key exchange algorithms. An attempt is currently being made to standardize a hybrid key exchange in TLS [11] .

For PKIs, however, only few approaches are known how to deal with combinations of algorithms. For certificates and CRLs, ITU-T X.509 (10/2019) [12] defines additional extensions to introduce additional algorithms for the certificate holder's public key and for the signature. The additional algorithms do not use the known fields subjectPublicKeyInfo and signature, but new extensions subjectAltPublicKeyInfo, altSignatureAlgorithm and altSignatureValue. The verification of such certificates with alternative algorithms depends on the ability of the relying party to process the alternative algorithms. If they can process the alternative algorithms, only these should be used for verification, if they cannot process them, only the native algorithms should be used. This approach is intended to solve the problem of

backward compatibility. We see the advantages of such a PKI for the introduction of new algorithms in certificates and CRLs. From our point of view, the isolated introduction of alternative algorithms in certificates and revocation lists is not sufficient for operators of open PKIs. The alternative algorithms need also to be introduced for certificate management formats (e.g. in PKCS#10, CMP, OCSP) and various end user applications. We see the most critical point of this approach in the fact that the certificate validity shall depend on the capability of software to process new algorithms. This results in different verification results of a certificate depending on the software used. In this way it is not possible to use certificates with alternative algorithms and signatures as qualified certificates according to eIDAS [1].

Despite these promising approaches, it will be necessary to set up new PKIs (parallel to the existing ones) if new algorithms are to be introduced and business continuity for applications must be guaranteed. Setting up and operating parallel PKIs, which may be linked by cross- or link certificates, is very complex, but well described by existing standards and less susceptible to errors due to the long experience and existing standards.

Another way is to compose algorithms to groups of algorithms. This composition is done below the certificate level and therefore it does not affect the standards for structure and validation of certificates. For the specific application TLS, this method is described in the above mentioned Draft RFC [11]. The way of building groups for TLS lacks flexibility. Introducing or removing algorithms in a group depends on standardization.

Intelligent Composed Algorithms

An Intelligent Composed Algorithm (ICA) describes a flexible set of algorithms and the associated rules for their combination. ICAs can be used with the previously mentioned PKI models such as hybrid and parallel PKI. This is possible because the combination of algorithms affects only the structures defined by an algorithm, such as algorithm identifiers, keys, signatures, or ciphertexts. These structures can be described in ASN.1. Standards for applications and protocols, such as X.509, RFC 5280, RFC 6960, RFC 2986, RFC 4210 or RFC 5652, are not affected.

ICAs are composed of two parts, the component algorithms, i.e. the actual cryptographic algorithms and the controlling algorithms. The component algorithms of an ICA are known[1] cryptographic algorithms, generally of the same type, i.e. signature algorithms or encryption algorithms. Controlling algorithms control the use of the cryptographic component algorithms in an Intelligent Composed Algorithm.

ICA have been devised to obtain stronger algorithms by combining known algorithms or to use component algorithms alternatively. In addition, the combined and/or alternative use of component algorithms shall prevent changes in PKI standards such as X.509 or CMS for the only reason to achieve algorithm agility. Changing today's PKI standards would require enormous efforts.

Cryptographic infrastructures that support ICAs allow a high level of cryptoagility. They enable the introduction of new cryptographic algorithms into running PKIs and the phase-out of outdated algorithms without jeopardizing the continuous operation of the cryptographic services. ICAs are not intended to be a tool for migrating to new algorithms, but are a means of building cryptographically agile systems[2]. When ICAs are introduced into a PKI, new cryptographic algorithms can easily be introduced without having to change higher-level standards of applications or protocols.

Syntax of ICAs

As usual, an ICA can and should be clearly identified in ASN.1 notation. An algorithm according to RFC 5280 is defined as

```
AlgorithmIdentifier  ::=  SEQUENCE  {
       algorithm               OBJECT IDENTIFIER,
       parameters              ANY DEFINED BY algorithm OPTIONAL  }
```

[1] "known" is understood as "known at the time of use as a component of an ICA". New algorithms can be used in ICAs as soon as the are defined.

[2] The introduction of the ICA concept requires new software for all participants of a PKI. Some people might think that this is too expensive. It should be noted, however, that several new cryptographic algorithms will have to be introduced in the next few years with significantly greater implementation effort. In comparison, the introduction of controlling algorithms is simple and cheap.

Accordingly, an ICA is defined by an OBJECT IDENTIFIER and its parameters. The component `algorithm` of type OBJECT IDENTIFIER designates the controlling algorithm used and the component `parameters`, the type of which is determined by the controlling algorithm, designates the component algorithms used and, optionally, other possible parameters.

An ICA defined in this way fits into all cryptographic structures containing algorithms specified in ASN.1. The example of an X.509 certificate according to RFC 5280 is used to illustrate this approach. In RFC 5280, an X.509 certificate is described as follows:

```
Certificate ::= SEQUENCE {
  tbsCertificate           TBSCertificate,
  signatureAlgorithm       AlgorithmIdentifier,
  signatureValue           BIT STRING }
```

where

```
TBSCertificate ::= SEQUENCE {
  version             [0]   EXPLICIT Version DEFAULT v1,
  serialNumber              CertificateSerialNumber,
  signature                 AlgorithmIdentifier,
  issuer                    Name,
  validity                  Validity,
  subject                   Name,
  subjectPublicKeyInfo      SubjectPublicKeyInfo,
  issuerUniqueID      [1]   IMPLICIT UniqueIdentifier OPTIONAL,
                            -- If present, version MUST be v2 or v3
  subjectUniqueID     [2]   IMPLICIT UniqueIdentifier OPTIONAL,
                            -- If present, version MUST be v2 or v3
  extensions          [3]   EXPLICIT Extensions OPTIONAL
                            -- If present, version MUST be v3
}
```

The cryptographically interesting components are `signatureAlgorithm` and `signatureValue` in the certificate itself and `signature` and `subjectPublicKeyInfo` in the certificate content (TBSCertificate)[3], where `signatureAlgorithm` in the certificate and `signature` in the certificate content must contain the same AlgorithmIdentifier. This AlgorithmIdentifier can of course also be an ICA consisting of a controlling algorithm and named component algorithms. The value of `signatureValue` is a BIT STRING whose generation (signature generation) and verification

[3] We are not going into details of cryptographic contents in certificate extensions here. It is easy to see, that ICAs can be applied in extensions, too.

(signature verification) is precisely defined by the specification of the algorithm designated by the AlgorithmIdentifier, see section Controlling algorithms.

The content of the component `subjectPublicKeyInfo` has little to do with the other three components mentioned, which all concern the certificate signature. It contains the public key that is assigned to the certificate owner by the certificate. The type of `subjectPublicKeyInfo` is

```
SubjectPublicKeyInfo  ::=  SEQUENCE  {
     algorithm                  AlgorithmIdentifier,
     subjectPublicKey           BIT STRING  }
```

Again, the component `algorithm` of type AlgorithmIdentifier can denote an ICA. The component `subjectPublicKey` is a BIT STRING whose use (signature verification, encryption or key agreement) is precisely defined by the AlgorithmIdentifier, see section Controlling algorithms.

There are no changes for ICA in the definition of the X.509 certificate and its use compared to the known cryptographic algorithms.

The same consideration as for X.509 certificates can be made for other structures with cryptographic elements, such as for revocation lists (according to RFC 5280), for CMS (according to RFC 5652), for OCSP (according to 6960) or even for signatures in passports (according to ICAO Doc 9303, Part 11, 2015). Wherever cryptographic algorithms with the definition of AlgorithmIdentifier given in RFC 5280 are used, ICAs can be used.

Controlling-Algorithms

A controlling algorithm defines syntax and rules how to combine known algorithms. Like cryptographic algorithms, it has a unique name, an OBJECT IDENTIFIER. The OBJECT IDENTIFIER of the controlling algorithm of an ICA is also the OBJECT IDENTIFIER of the ICA as it appears in applications.

Parameters are defined for a controlling algorithm. The parameters are the component algorithms that are combined by the controlling algorithm[4] . Optionally, there may be additional parameters.

For an application to process an ICA, it must first be able to process the controlling algorithm. As for cryptographic algorithms, an area of application is defined for controlling algorithms. The most obvious application areas are

- digital signature
- encryption
- key agreement

Further, it is defined how the keys of the algorithm should look like. We are primarily dealing with asymmetric cryptography. Thus, it has to be defined what the private and public keys of the controlling algorithm should look like. The private and public keys of a controlling algorithm are composed by concatenating the private and public keys, respectively, of the component algorithms named in the parameters. The order of the (partial) keys of the component algorithms is exactly that of the component algorithms in the parameter. The key generation is just as simple: The keys of all component algorithms are generated and assembled by concatenation in the named order to the key of the ICA[5].

Depending on the application area it has to be specified for controlling algorithm

- in case of digital signature: format of signatures, process of signature generation, process of signature verification,
- in case of encryption: format of ciphertexts, process of encryption, process of decryption,
- in case of key agreement: format of shared secrets, process of shared secret generation.

Appendix A lists rudimentary specifications of controlling algorithms.

---

[4] The idea to parametrize an algorithm by further algorithms is not really new. RSA in the variants RSA PSS and RSA OAEP is parametized by mask functions and hash algorithms, see RFC 8017.

[5] The rules to generate and to use the keys is certainly associated to the controlling algorithm. The result of the application of the rules might be seen as associated to the ICA.

In Appendix B, an AlgorithmIdentifier of a signature-OR combination of RSA PSS and ECDSA is given as an example.

Use of ICAs

To use ICAs, only the controlling algorithms need to be implemented. The component algorithms, be they RSA, ECDSA, ECDH or new, quantum-safe algorithms, are already implemented or need to be implemented, regardless of whether ICAs should be used. Controlling algorithms are easy to implement, no deep cryptographic knowledge is required. Controlling algorithms are based on simple arithmetic and logical operations.

Roll out of ICAs is only a small additional effort if new server and client software has to be rolled out anyway due to the introduction of a new cryptographic component algorithm.

Once the most important ICAs[6] have been implemented and rolled out in a PKI, there are many advantages:

Even without the introduction of new cryptographic algorithms, more flexible solutions or more secure solutions can be found by combining known algorithms.

The introduction of new cryptographic algorithms into a PKI is substantially easier, because there is no need to perform PKI upgrades for all PKI participants simultaneously. The introduction of a new cryptographic algorithm can occur at different times at the CAs, the end entities (EEs - i.e. those participants of the PKI who have keys and certificates) and Relying Parties (RPs):

- CAs can create and distribute certificates with new algorithms encapsulated in ICAs, regardless of the special conditions of EEs and RPs.
- EEs can individually customize their application software for new algorithms independently of the CAs and other EEs.

[6] We think of ICAs with the controlling algorithms sketched in Appendix A.

- RPs can customize their application software for new algorithms independently of CAs, EEs and other RPs.

This asynchronous introduction of new algorithms is possible as long as controlling algorithms are used with OR designs, such as signature-OR or signature K of N.

As with the introduction of new algorithms, an asynchronous approach is also possible for the phase-out of outdated algorithms.

- As soon as a CA rates an algorithm for no longer trustworthy, new certificates are issued without this algorithm encapsulated in the ICAs. The other PKI participants will be affected by this action only if they have supported only this one specific algorithm and not two or more algorithms, as would have been possible with ICAs.

- If EEs or RPs no longer want to accept an outdated algorithm, they can customize their systems accordingly. Thereafter, only certificates or other data that contain only the outdated cryptographic algorithm encapsulated in the ICAs and not two or more algorithms, as would have been possible with ICAs, will not be processed.

ICAs allow for another option in usage: Usually there are several algorithms for signature, encryption, and key agreement that can be combined. On the other hand, individual policies can regulate the use or non-use of certain cryptographic algorithms. Combining algorithms with ICA allows different policies of PKI participants without destroying communication with partners. Thus, if an EE wants to sign some documents for a larger number of RPs and at the same time it is known that the RPs accept different algorithms in their verification policies without a common intersection, the EE can use an ICA signature OR to combine several signatures into an ICA signature that can be verified by each RP.

This method is also suitable for introducing algorithm catalogs in the operation of a PKI.

References

[1] 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

[2] Baseline Requirements for the Issuance and Management of Publicly-Trusted Certificates, CA/Browser Forum, https://cabforum.org/baseline-requirements-documents/

[3] Guidelines For The Issuance And Management Of Extended Validation Certificates, CA/Browser Forum, https://cabforum.org/extended-validation-2/

[4] Post-Quantum Cryptography, Call for Proposals, National Institute of Standards and Technology, 2017, https://csrc.nist.gov/Projects/post-quantum-cryptography/post-quantum-cryptography-standardization/Call-for-Proposals

[5] Post-Quantum Cryptography, Round 3 Submissions, National Institute of Standards and Technology, 2020, https://csrc.nist.gov/Projects/post-quantum-cryptography/round-3-submissions

[6] https://pqcrypto.eu.org/index.html

[7] https://player.vimeo.com/video/507939029

[8] Nina Bindel, Jacqueline Brendel, Marc Fischlin, Brian Goncalves, Douglas Stebila: Hybrid Key Encapsulation Mechanisms and Authenticated Key Exchange, 10th International Workshop on Post-Quantum Cryptography, 2019, https://link.springer.com/chapter/10.1007/978-3-030-25510-7_12

[9] https://security.googleblog.com/2016/07/experimenting-with-post-quantum.html

[10] https://sites.google.com/a/chromium.org/dev/cecpq2

[11] https://tools.ietf.org/html/draft-ietf-tls-hybrid-design-01

[12] International Telecommunication Union: DATA NETWORKS, OPEN SYSTEM COMMUNICATIONS AND SECURITY, Directory, Information technology – Open Systems Interconnection – The Directory: Public-key and attribute certificate frameworks, Recommendation ITU-T X.509 (10/2019)

[13] M. Nystrom, B. Kaliski: PKCS #10: Certification Request Syntax Specification, Version 1.7, Request for Comments: 2986, November 2000

[14] C. Adams, S. Farrell, T. Kause, T. Mononen: Internet X.509 Public Key Infrastructure - Certificate Management Protocol (CMP), Request for Comments: 4210, September 2005

[15] S. Santesson, M. Myers, R. Ankney, A. Malpani, S. Galperin, C. Adams: X.509 Internet Public Key Infrastructure - Online Certificate Status Protocol - OCSP, Request for Comments: 6960, June 2013

[16] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, W. Polk: Internet X.509 Public Key Infrastructure - Certificate and Certificate Revocation List (CRL) Profile, Request for Comments: 5280, May 2008

[17] R. Housley: Cryptographic Message Syntax (CMS), Request for Comments: 5652, September 2009

Appendix A

A.1 signature-OR

Name: **signature-OR**

OID: `{1 3 6 1 4 1 4788 6 1 1 1}`

Parameters: `ComponentAlgorithms ::= SEQUENCE SIZE (1..MAX) OF AlgorithmIdentifier`

Application area: Digital signatures

Keys:

private: SEQUENCE OF ANY

The number of elements of the private key (SEQUENCE OF ANY) is the number of elements of the parameters (`ComponentAlgorithms`) and the precise types of the private key elements (ANY) are given by the corresponding elements of the parameters (`AlgorithmIdentifier`).

(Concatenation of private keys of the component algorithms)

public: SEQUENCE OF ANY

The number of elements of the public key (SEQUENCE OF ANY) is the number of elements of the parameters (`ComponentAlgorithms`) and the precise types of the public key elements (ANY) are given by the corresponding elements of the parameters (`AlgorithmIdentifier`).

(Concatenation of public keys of the component algorithms)

Key generation: The keys are generated individually for all component algorithms. Subsequently, the generated keys are concatenated in the same succession as the are named in the parameters (`ComponentAlgorithms`).

Signature: SEQUENCE OF ANY

The number of elements of the signature (SEQUENCE OF ANY) is the number of elements of the parameters (`ComponentAlgorithms`) and the precise types of the signature elements (ANY) are given by the corresponding elements of the parameters (`AlgorithmIdentifier`).

(Concatenation of signatures of the component algorithms)

Preconditions: A key pair of the chosen algorithm {OID, Parameter} has been generated. The private and the public key, respectively, are available in the form of the concatenation of the private and public keys, respectively, of the component algorithms named in the parameters.

A hash algorithm has been agreed and an hash value H of a message to be signed M can be computed.

Signature process:

Input: Private key given as concatenation of the private keys of the component algorithms named in the parameters.

Hash H of the message to be signed M

Actions: The component signatures for all component algorithms named in the parameters are computed.

The signature value is formed as the concatenation of the component signatures in the order given in the parameters.

Output: Signature value as the concatenation of the component signatures.

Verification process:

Input: Public key given as concatenation of the public keys of the component algorithms named in the parameters.

Hash H of the message to be signed M

Value of the signature to be verified

Actions: For all component algorithms do:

- Check if the component algorithm can be handled, otherwise proceed to the next component algorithm.
- Extract the component signature from the signature to be verified.
- If the component signature can be verified successfully, complete verification process with the result OK, otherwise proceed to the next component algorithm.

If the processing of none of the component algorithms has provided the result OK, complete verification process with the result not-OK.

Output. Verification process OK or not-OK.

A.2 signature-AND

Name: **signature-AND**

OID: `{1 3 6 1 4 1 4788 6 1 1 2}`

Parameters: `ComponentAlgorithms ::= SEQUENCE SIZE (1..MAX) OF AlgorithmIdentifier`

Application area: Digital signatures

Keys:

private: SEQUENCE OF ANY

The number of elements of the private key (SEQUENCE OF ANY) is the number of elements of the parameters (`ComponentAlgorithms`) and the precise types of the private key elements (ANY) are given by the corresponding elements of the parameters (`AlgorithmIdentifier`).

(Concatenation of private keys of the component algorithms)

public: SEQUENCE OF ANY

The number of elements of the public key (SEQUENCE OF ANY) is the number of elements of the parameters (`ComponentAlgorithms`) and the precise types of the public key elements (ANY) are given by the corresponding elements of the parameters (`AlgorithmIdentifier`).

(Concatenation of public keys of the component algorithms)

Key generation: The keys are generated individually for all component algorithms. Subsequently, the generated keys are concatenated in the same succession as the are named in the parameters (`ComponentAlgorithms`).

Signature: SEQUENCE OF ANY

The number of elements of the signature (SEQUENCE OF ANY) is the number of elements of the parameters (`ComponentAlgorithms`) and the precise types of the signature elements (ANY) are given by the corresponding elements of the parameters (`AlgorithmIdentifier`).

(Concatenation of signatures of the component algorithms)

Preconditions: A key pair of the chosen algorithm {OID, Parameter} has been generated. The private and the public key, respectively, are available in the form of the concatenation of the private and public keys, respectively, of the component algorithms named in the parameters.

A hash algorithm has been agreed and an hash value H of a message to be signed M can be computed.

Signature process:

Input: Private key given as concatenation of the private keys of the component algorithms named in the parameters.

Hash H of the message to be signed M

Actions: The component signatures for all component algorithms named in the parameters are computed.

The signature value is formed as the concatenation of the component signatures in the order given in the parameters.

Output: Signature value as the concatenation of the component signatures.

Verification process:

Input: Public key given as concatenation of the public keys of the component algorithms named in the parameters.

Hash H of the message to be signed M

Value of the signature to be verified

Actions: For all component algorithms do:

- Check if the component algorithm can be handled, otherwise complete verification process with the result not-OK.
- Extract the component signature from the signature to be verified.
- If the component signature can be verified successfully, proceed to the next component algorithm, otherwise complete verification process with the result not-OK.

If the processing of all of the component algorithms has provided the result OK, complete verification process with the result OK.

Output. Verification process OK or not-OK.

A.3 signature-K-OF-N

Name: **signature-K-OF-N**

OID: `{1 3 6 1 4 1 4788 6 1 1 1}`

Parameters:

```
SEQUENCE {
    k             INTEGER,
    compAlgs      SEQUENCE SIZE (k..MAX) OF AlgorithmIdentifier}
```

Application area: Digital signatures

Keys:

private: SEQUENCE OF ANY

The number of elements of the private key (SEQUENCE OF ANY) is the number of elements of the parameters (`ComponentAlgorithms`) and the precise types of the private key elements (ANY) are given by the corresponding elements of the parameters (`AlgorithmIdentifier`).

(Concatenation of private keys of the component algorithms)

public: SEQUENCE OF ANY

The number of elements of the public key (SEQUENCE OF ANY) is the number of elements of the parameters (`ComponentAlgorithms`) and the precise types of the public key elements (ANY) are given by the corresponding elements of the parameters (`AlgorithmIdentifier`).

(Concatenation of public keys of the component algorithms)

Key generation: The keys are generated individually for all component algorithms. Subsequently, the generated keys are concatenated in the same succession as the are named in the parameters (`ComponentAlgorithms`).

Signature: SEQUENCE OF ANY

The number of elements of the signature (SEQUENCE OF ANY) is the number of elements of the parameters (`ComponentAlgorithms`) and the precise types of the signature elements (ANY) are given by the corresponding elements of the parameters (`AlgorithmIdentifier`).

(Concatenation of signatures of the component algorithms)

Preconditions: A key pair of the chosen algorithm {OID, Parameter} has been generated. The private and the public key, respectively, are available in the form of the concatenation of the private and public keys, respectively, of the component algorithms named in the parameters.

A hash algorithm has been agreed and an hash value H of a message to be signed M can be computed.

Signature process:

Input: Private key given as concatenation of the private keys of the component algorithms named in the parameters.

Hash H of the message to be signed M

Actions: The component signatures for all component algorithms named in the parameters are computed.

The signature value is formed as the concatenation of the component signatures in the order given in the parameters.

Output: Signature value as the concatenation of the component signatures.

Verification process:

Input: Public key given as concatenation of the public keys of the component algorithms named in the parameters

Value k - parameter of the control algorithm

Value n - the number of component algorithms given by the number of elements in parameter compAlgs

Hash H of the message to be signed M

Value of the signature to be verified

Actions: Set a the counter i of the number of verified component signatures to 0.

Set the counter j of the number of of still not checked component signatures to n.

For all component algorithms do:

- If i+j < k , complete verification process with the result not-OK.
- Check if component algorithm can be handled, otherwise set j := j-1 and proceed to the next component algorithm.
- Extract component signature from the signature to be verified.
- If component signature can be verified successfully, set i := i+1, otherwise set j := j-1.
- If i >= k, complete verification process with the result OK, otherwise proceed to the next component algorithm.

Output. Verification process OK or not-OK.

A.4 key-agreement-XOR

Name: **key-agreement-XOR**

OID: `{1 3 6 1 4 1 4788 6 1 2 1}`

Parameters: `ComponentAlgorithms ::= SEQUENCE SIZE (1..MAX) OF AlgorithmIdentifier`

Application area: Key agreement

Keys:

private: SEQUENCE OF ANY

The number of elements of the private key (SEQUENCE OF ANY) is the number of elements of the parameters (`ComponentAlgorithms`) and the precise types of the private key elements (ANY) are given by the corresponding elements of the parameters (`AlgorithmIdentifier`).

(Concatenation of private keys of the component algorithms)

public: SEQUENCE OF ANY

The number of elements of the public key (SEQUENCE OF ANY) is the number of elements of the parameters (`ComponentAlgorithms`) and the precise types of the public key elements (ANY) are given by the corresponding elements of the parameters (`AlgorithmIdentifier`).

(Concatenation of public keys of the component algorithms)

Key generation: The keys are generated individually for all component algorithms. Subsequently, the generated keys are concatenated in the same succession as the are named in the parameters (`ComponentAlgorithms`).

Shared Secret: OCTET STRING

Preconditions: Key pairs of the chosen algorithm {OID, Parameter} have been generated, each by the communication partners. The own private key and the public key of the communication partner, respectively, are available in the form of the concatenation of the private and public keys, respectively, of the component algorithms named in the parameters.

Process of key agreement:

Input: Own private key given as concatenation of the private keys of the component algorithms named in the parameters.

Public key of the communication partner given as concatenation of the public keys of the component algorithms named in the parameters.

Actions: Set value ComposedSharedSecret to 0.

For all component algorithms do:

- Compute value ComponentSharedSecret as the shared secret of the own private key and the public key of the communication partner associated to the selected component algorithm.
- Set ComposedSharedSecret := ComposedSharedSecret XOR ComponentSharedSecret

Output: Value of ComposedSharedSecret

A.5 key-agreement-CONCAT

Name: **key-agreement-CONCAT**

OID: `{1 3 6 1 4 1 4788 6 1 2 2}`

Parameters: `ComponentAlgorithms ::= SEQUENCE SIZE (1..MAX) OF AlgorithmIdentifier`

Application area: Key agreement

Keys:

private: SEQUENCE OF ANY

The number of elements of the private key (SEQUENCE OF ANY) is the number of elements of the parameters (`ComponentAlgorithms`) and the precise types of the private key elements (ANY) are given by the corresponding elements of the parameters (`AlgorithmIdentifier`).

(Concatenation of private keys of the component algorithms)

public: SEQUENCE OF ANY

The number of elements of the public key (SEQUENCE OF ANY) is the number of elements of the parameters (`ComponentAlgorithms`) and the precise types of the public key elements (ANY) are given by the corresponding elements of the parameters (`AlgorithmIdentifier`).

(Concatenation of public keys of the component algorithms)

Key generation: The keys are generated individually for all component algorithms. Subsequently, the generated keys are concatenated in the same succession as the are named in the parameters (`ComponentAlgorithms`).

Shared Secret: OCTET STRING

Preconditions: Key pairs of the chosen algorithm {OID, Parameter} have been generated, each by the communication partners. The own private key and the public key of the communication partner, respectively, are

available in the form of the concatenation of the private and public keys, respectively, of the component algorithms named in the parameters.

Process of key agreement:

Input: Own private key given as concatenation of the private keys of the component algorithms named in the parameters.

Public key of the communication partner given as concatenation of the public keys of the component algorithms named in the parameters.

Actions: Set value ComposedSharedSecret to 0.

For all component algorithms do:

- Compute value ComponentSharedSecret as the shared secret of the own private key and the public key of the communication partner associated to the selected component algorithm.
- Set ComposedSharedSecret := ComposedSharedSecret || ComponentSharedSecret

Output: Value of ComposedSharedSecret

A.6 encryption-ITERATION

Name: **encryption-ITERATION**

OID: `{1 3 6 1 4 1 4788 6 1 3 1}`

Parameters: `ComponentAlgorithms ::= SEQUENCE SIZE (1..MAX) OF AlgorithmIdentifier`

Application area: Asymmetric Encryption

Keys:

private: SEQUENCE OF ANY

The number of elements of the private key (SEQUENCE OF ANY) is the number of elements of the parameters (`ComponentAlgorithms`) and the precise types of the private key elements (ANY) are given by the corresponding elements of the parameters (`AlgorithmIdentifier`).

(Concatenation of private keys of the component algorithms)

public: SEQUENCE OF ANY

The number of elements of the public key (SEQUENCE OF ANY) is the number of elements of the parameters (`ComponentAlgorithms`) and the precise types of the public key elements (ANY) are given by the corresponding elements of the parameters (`AlgorithmIdentifier`).

(Concatenation of public keys of the component algorithms)

Key generation: The keys are generated individually for all component algorithms. Subsequently, the generated keys are concatenated in the same succession as the are named in the parameters (`ComponentAlgorithms`).

Ciphertext: OCTET STRING

Preconditions: A key pair of the chosen algorithm {OID, Parameter} has been generated. The private and the public key, respectively, are available in the form of the concatenation of the private and public keys, respectively, of the component algorithms named in the parameters.

Encryption process:

Input: Public key given as concatenation of the public keys of the component algorithms named in the parameters

Message to be encrypted

Actions: Set value Ciphertext := Message to be encrypted.

For all component algorithms do (in the order given by the parameters):

- Compute the new value of Ciphertext by application (encryption) of the component algorithm to the current value of Ciphertext.

Output: Encrypted message given by the value of Ciphertext

Decryption process:

Input: Private key given as concatenation of the private keys of the component algorithms named in the parameters

Encrypted message

Actions: Set value Plaintext := Encrypted message

For all component algorithms do (in the reverse order given by the parameters):

- Compute the new value of Plaintext by application (decryption) of the component algorithm to the current value of Plaintext.

Output: Decrypted message given by the value of Plaintext

Appendix B

Algorithm identifier for signature-OR with RSA-PSS and ECDSA

```
SEQUENCE {
   OBJECT IDENTIFIER ' 1 3 6 1 4 1 4788 6 1 1 1 ,
                      -- id-signature-OR
   SEQUENCE {                -- Parameter signature-OR
      SEQUENCE {                   -- algorithIdentifier RSA-PSS
         OBJECT IDENTIFIER '1 2 840 113549 1 1 10'
                                      -- id-RSASSA-PSS
         SEQUENCE {                   -- RSASSA-PSS-params
            [0] {
               SEQUENCE {
                  OBJECT IDENTIFIER '2 16 840 1 101 3 4 2 3'
               }
            }
            [1] {
               SEQUENCE {
                  OBJECT IDENTIFIER '1 2 840 113549 1 1 8'
                  SEQUENCE {
                     OBJECT IDENTIFIER '2 16 840 1 101 3 4 2 3’
                  }
               }
            }
            [2] {
               INTEGER 64
            }
         }
      }
      SEQUENCE {                   -- algorithmIdentifier ECDSA
         OBJECT IDENTIFIER '1 2 840 10045 4 3 3 '
                                   -- ecdsawith-SHA384 (RFC 5758)
      }                            -- no parameters
   }
}
```