IEEE P802.15
Wireless Personal Area Networks

Project	IEEE P802.15 Working Group for Wireless Personal Area Networks (WPANs)	
Title	**Mandatory ECC Security Algorithm Suite**	
Date Submitted	April 5, 2002	
Source	[Rene Struik, Certicom Corp.] 5520 Explorer Drive, 4th Floor Mississauga, ON Canada L4W 5L1 [Gregg Rasor, Motorola] 1500 Gateway Blvd. Boynton Beach, Florida 33426	Voice: (905) 501-6083 Fax: (905) 507-4230 E-mail: rstruik@certicom.com Voice: (561) 739-2952 Fax: (561) 739-3715 E-mail: Gregg.Rasor@motorola.com
Re:	802.15.3 TG3 Security call for proposals	
Abstract	An elliptic curve cryptography (ECC) security suite is defined including a full public key cryptosystem with digital signatures and implicit certificates. Binding between a device's identity and public key using digital certificates may be established manually through direct user intervention, at device provisioning by a distributor or manufacturer, or by a trusted third party.	
Purpose	This document defines the security elements necessary to implement the mandatory elliptic curve cryptography (ECC) security suite.	
Notice	This document has been prepared to assist the IEEE P802.15. It is offered as a basis for discussion and is not binding on the contributing individual(s) or organization(s). The material in this document is subject to change in form and content after further study. The contributor(s) reserve(s) the right to add, amend or withdraw material contained herein.	
Release	The contributor acknowledges and accepts that this contribution becomes the property of IEEE and may be made publicly available by P802.15.	

This document contains the following information:
1. Informal Specification of Cryptographic Primitives and Protocols;
2. Formal Specification of Cryptographic Primitives and Protocols;
3. Data Types and Data Formats.

Table of Contents

1. Introduction

02200r0P802-15_TG3-Mandatory-ECC-Security-Algorithm-Suite defines a security architecture designed to accommodate a selected highly secure elliptic curve public key cryptosystem including state of the art authentication methods using digital signatures and implicit digital certificates. Optional support for X.509 digital certificates is also provided. The implementation specified herein is intended to provide security services for the 802.15.3 WPAN, conforming with well established security standards as cited herein.

Annex B of the 802.15.3 specification contains implementation notes for designer that should be carefully read and understood. The drafters of this document warn anyone implementing this subject matter that any variations from the specified algorithms, security architecture, and recommended hardware protections may result in an insecure implementation and the possible compromise of secured information channels, as well as the unwanted exposure of private information.

1.1 Scope

This document covers text related to the selected mandatory ECC security suite (described herein as an algorithm suite) for inclusion in the 802.15.3 standard.

1.2 Purpose

This document is intended as an algorithm suite submission, including required architectural protocol elements, to the 802.15 TG3 for inclusion in the 802.15.3 standard. The text from this submission is suitable for direct incorporation in the standard.

1.3 Document Organization

This document contains text for the selected mandatory ECC algorithm suite for the 802.15.3 standard. In addition, this submission includes informative text that may or may not be included in the draft standard to support the architecture.

The document is organized into the following categories:

- References (Normative)
- Instantiation of the algorithm suite (Normative)
- Security considerations for an informative annex (Informative)

The reference section describes the external documents that are required in order to implement the cryptographic algorithms proposed in this document.

The instantiation section provides text and information for the aspects of the algorithm suite that must be defined in order to fully implement the security architecture. This text may be added to the main section of the document or to an annex.

The security considerations provide security analysis and rationale for the algorithm suite. For the security architecture, the reader is referred to document 02/130r1 NTRU Security Architecture Proposal.

2. Definitions, Abbreviations, and References

2.1 Definitions and Abbreviations

Our definitions follow those as defined in [ANSI X9.63-2001, §2.1].

2.2 Symbols and Notation

Our notation follows [ANSI X9.63-2001, §2.2].

2.2.1 Normative References

1. ANSI X9.30-1997, The Digital Signature Algorithm (DSA) (Revised), Annex F: Parameter Settings and Security, American Bankers Association, July 1, 1999.
2. ANSI X9.63-2001, Public Key Cryptography for the Financial Services Industry – Key Agreement and Key Transport Using Elliptic Curve Cryptography, American Bankers Association, November 20, 2001.
3. FIPS Pub 180-1, Secure Hash Standard, Federal Information Processing Standards Publication 180-1, US Department of Commerce/N.I.S.T., Springfield, Virginia, April 17, 1995 (supersedes FIPS Pub 180). Available from http://csrc.nist.gov/.
4. FIPS Pub 180-2, Draft Specification for the Secure Hash Standard, Federal Information Processing Standards Publication 180-2, US Department of Commerce/N.I.S.T., draft, 2001. Available from http://csrc.nist.gov/.
5. FIPS Pub 186-2, Digital Signature Standard (DSS), Federal Information Processing Standards Publication 186-2, US Department of Commerce/N.I.S.T., Springfield, Virginia, January 27, 2000. Available from http://csrc.nist.gov/.
6. FIPS Pub 197, Advanced Encryption Standard (AES), Federal Information Processing Standards Publication 197, US Department of Commerce/N.I.S.T, November 26, 2001. Available from http://csrc.nist.gov/.
7. FIPS Pub #HMAC, Keyed-Hash Message Authentication Code (HMAC), Federal Information Processing Standards Publication #HMAC, US Department of Commerce/N.I.S.T., draft, May 30, 2001. Available from http://csrc.nist.gov/.
8. ISO/IEC 9798-1, Information Technology - Security Techniques - Entity Authentication Mechanisms – Part 1: General Model, International Standardization Organization, Geneva, Switzerland, 1991 (first edition).
9. ISO/IEC 9798-2, Information Technology - Security Techniques - Entity Authentication Mechanisms – Part 2: Mechanisms Using Symmetric Encipherment Algorithms, International Standardization Organization, Geneva, Switzerland, 1994 (first edition).

10. ISO/IEC 9798-3, Information Technology - Security Techniques - Entity Authentication Mechanisms – Part 3: Entity Authentication using a Public Key Algorithm, International Standardization Organization, Geneva, Switzerland, 1993 (first edition).
11. NIST Pub 800-38A 2001 ED, Recommendation for Block Cipher Modes of Operation – Methods and Techniques, NIST Special Publication 800-38A, 2001 Edition, US Department of Commerce/N.I.S.T., December 2001. Available from http://csrc.nist.gov/.
12. PKIX, L. Bassham, R. Housley, W. Polk, Algorithms and Identifiers for the Internet X.509 Public Key Infrastructure Certificate and CRL Profile, Internet Draft, PKIX Working Group, October 2001.
13. RFC 2104, HMAC: Keyed-Hashing for Message Authentication, Internet Request for Comments 2104, H. Krawczyk, M. Bellare, R. Canetti, February 1997.
14. RFC 2119, Key Words for Use in RFCs to Indicate Requirement Levels, Internet Request for Comments 2119, S. Bradner, March 1997.
15. Standards for Efficient Cryptography, SEC 1: Elliptic Curve Cryptography, Version 1.0, Certicom Research, September 20, 2000. Available from http://www.secg.org/.
16. Standards for Efficient Cryptography, SEC 2: Recommended Elliptic Curve Domain Parameters, Version 1.0, Certicom Research, September 20, 2000. Available from http://www.secg.org/.
17. Standards for Efficient Cryptography, SEC 4: Implicit Certificates, working draft, Version 0.2, Certicom Research, November 14, 2000.

2.2.2 Informative References

1. D.R.L. Brown, R. Gallant, S.A. Vanstone, Provably Secure Implicit Certificate Schemes, in *Proceedings of Financial Cryptography 2001*, to appear.
2. IEEE Draft P802.15.3/D09, Draft Standard for Information Technology – Telecommunications and Information Exchange Between Systems – Local and Metropolitan Are Networks Specific Requirements – Part 15.3: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for High Rate Wireless Personal Area Networks (WPAN), Draft P802.15.3/09, December 7, 2001.
3. L. Law, A.J. Menezes, M. Qu, J. Solinas, S.A. Vanstone, An Efficient Protocol for Authenticated Key Agreement, Technical Report CORR 1998-05, CACR, University of Waterloo, 1998. Available from http://www.cacr.math.uwaterloo.ca.
4. A.K. Lenstra, Unbelievable Security: Matching AES Security using Public Key Systems, in *Proceedings of Advances in Cryptology – ASIACRYPT 2001,* December 10-13, 2001.
5. A.J. Menezes, P.C. van Oorschot, S.A. Vanstone, Handbook of Applied Cryptography, Boca Raton: CRC Press, 1997.
6. L.A. Pintsov, S.A. Vanstone, Postal Revenue Collection in the Digital Age, Technical Report CORR 2000-43, CACR, University of Waterloo, 2000. Available from http://www.cacr.math.uwaterloo.ca.

3. Algorithm Suite Specification

3.1 Object Identifier

The object identifiers for the mandatory ECC security suites will be built off of the following OID root. This root is written in ASN.1 format.

```
id-ecc-security-suites OBJECT IDENTIFIER ::= {
     iso(1) identified-organization() dod() internet()
     private() enterprises() certicom() ieee802-15-3()
     securitySuites()}
```

The object identifier for the security suite proposed in this clause is:

```
id-ecc-security-suite1 OBJECT IDENTIFIER ::= {
     id-ecc-security-suites 1}
```

The DER encoding of this element, which shall be included in the OID field when this OID is used is the hex value: {13 bytes of coded hexadecimal}.

3.2 Security Functionality Provided

This security suite provides the following security services.

Security Service	Provided
Mutual Authentication	✓
Authenticated Key Agreement	✓
Verification of Public-Key	✓
Key Establishment	✓
Key Transport	✓
Beacon Integrity Protection	✓
Freshness Protection	✓
Command Integrity Protection	✓
ACK Integrity Protection	✓
Data Integrity Protection	✓
Data Encryption	✓
Digital Signatures	✓
Digital Certificates	✓

3.3 Data Formats

This table specifies the length and meaning of the undefined data elements from clause 7.

Notation	Length	Value	Description
PublicKeyObjectType	2	See →.	The public key object type is selected from one of the following elements: (1) public key only, (2) implicit certificate, or (3) X.509 certificate. Operations are performed for 128 bit strength using the elliptic curve ansit283k1 as specified in ANSI X9.63.
PublicKeyObjectLength	2	37 + length of add. info	The length of the additional information concatenated with the implicit certificate.
PublicKeyObject	object length	Variable	The public key object is defined as one of the following elements: (1) public key only, (2) implicit certificate, or (3) X.509 certificate. Operations are performed for 128 bit strength using the elliptic curve ansit283k1 as specified in ANSI X9.63.
AuthResponseType	2	See 02/130rx sec 6.4.1.2.	The auth response type specifies the result of an HMAC computation as defined in FIPS #HMAC using a 16-byte key and the SHA-256 hash algorithm as defined in FIPS 180-2.
AuthResponseLength	2	16	The length of an HMAC computation using a 16-byte key as defined in FIPS #HMAC.
AuthResponse	16	Variable	The result of the HMAC computation using a 16-byte key as defined in FIPS #HMAC.
OIDLength	1	4	The length of the encoded IEEE security suite OID.
OID	4	OID Value	The encoded IEEE security suite OID, which is the hex value 00 00 00 00 for the mandatory security suite.
ChallengeType	2	1	The challenge type is an elliptic curve point. Operations are performed for 128 bit strength using the elliptic curve ansit283k1 as specified in ANSI X9.63. All challenges are randomly and unpredictably generated at the time of the challenge.
ChallengeLength	2	37	The length of an ECC MQV ephemeral key, which is a point on the curve ansit283k1 as specified in ANSI X9.63.

Notation	Length	Value	Description
Challenge	37	Variable	The ECC MQV ephemeral key, which is a point on the curve ansit283k1 as specified in ANSI X9.63.
ChallengeResponseType	2	4	The challenge response type specifies the result of an HMAC computation as defined in FIPS #HMAC using a 16-byte key and the SHA-256 hash algorithm as defined in FIPS 180-2.
ChallengeResponseLength	2	16	The length of an HMAC computation as defined in FIPS #HMAC using a 16-byte key and the SHA-256 hash algorithm as defined in FIPS 180-2.
ChallengeResponse	16	Variable	The result of the HMAC computation as defined in FIPS #HMAC using a 16-byte key and the SHA-256 hash algorithm as defined in FIPS 180-2.
KeyPurpose	1	0	The type of key requested in key request protocols. Only seeds are transmitted in this security suite.
EncryptedKeyType	2	2	The encrypted key type specifies the result of AES-128 CBC encryption of the 128-bit seed with random IV as specified in FIPS Pub 197 and NIST Special Publication 800-38A.
EncryptedKeyLength	2	16	The length of an encrypted 128-bit seed encrypted using AES-128 CBC encryption with random IV as specified in FIPS Pub 197 and NIST Special Publication 800-38A.
EncryptedKey	16	Variable	The result of the encryption of the 128-bit seed using AES-128 CBC encryption with random IV as specified in FIPS Pub 197 and NIST Special Publication 800-38A.

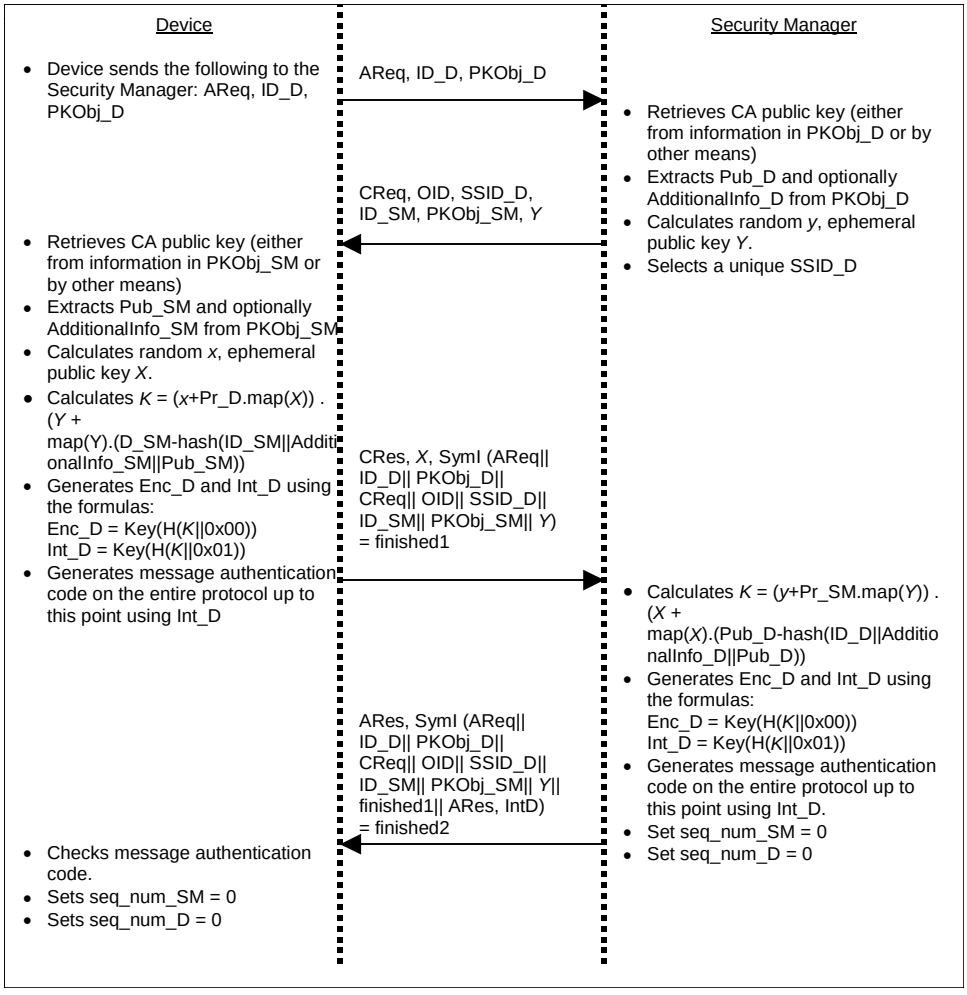
Table XX – Frame object formats

3.4 Cryptographic Operation Selections

The security architecture is instantiated by the following algorithms.

- Three and four-pass versions of MQV based on that specified in ANSI X9.63 [X9.63] for public-key key agreement and authentication.
- 128-bit AES CBC-EDE mode as specified in [AES] for all symmetric encryption.

- HMAC with SHA-256 message authentication codes as specified in [HMAC] for symmetric integrity protection.
- SHA-256 cryptographic hash as specified in [FIPS180-2] for all cryptographic hashing.
- Implicit certification as specified herein for establishing authenticity of public keys.
- Elliptic curve point arithmetic on the elliptic curve ansit283k1 defined in appendix J.4.5 of ANSI X9.63. Note that all points are to be transmitted in compressed form.
- SHA-2 cryptographic hash as specified in FIPS 180-2 [FIPS180-2] for all cryptographic hashing.



The use of the cryptographic algorithms for each of the security operations performed in the piconet are specified in the following table:

Use	Operation
Verification of Public-Key	The verification of the public key is assured by the certification by the appropriate CA and is implicitly verified if the MQV authentication and key exchange protocol succeeds.
Challenge generation	The challenges generated during the authentication protocol consist of an ephemeral public/private key pair, where the private key is an integer and the public key is a point on the curve.
Seed generation (for authentication protocol)	The seed *K* for the authentication protocol consists of the result of the each party's MQV computation on the challenges, the other party's certification information, the CA's public key, and the party's own private key.
Integrity Key Derivation	All integrity keys are generated from the seed *K* by calculating the SHA-256 hash of the seed concatenated with the byte 0x00 and then setting the key to be the first 128-bits of the result.
Encryption Key Derivation	All encryption keys are generated from the seed *K* by calculating the SHA-256 hash of the seed concatenated with the byte 0x01 and then setting the key to be the first 128-bits of the result.
Challenge response generation	The challenge response is computed by computing the HMAC message authentication code on the entire authentication protocol up to that point using the integrity key.
Authentication response generation	The authentication response is computed by computing the HMAC message authentication code on the entire authentication protocol up to that point using the integrity key.
Beacon message authentication code generation (Integrity Code information element)	The message authentication code included in the beacon is computed as the HMAC message authentication code on the entire beacon up to the integrity code information element using the integrity key.
Command message authentication code generation	The message authentication code included in command frames is computed as the MHAC message authentication code on the entire command up to the message authentication code using the integrity key.
ACK message authentication code generation	The message authentication code included in ACK frames is computed as the HMAC message authentication code on the entire ACK up to the message authentication code using the integrity key.
Data message authentication code generation	The message authentication code included in data frames is computed as the HMAC message authentication code on the entire data frame up to the message authentication code after encryption has been performed using the integrity key.
Seed encryption operation (for request key and distribute key)	The seed for key transport is encrypted using AES with a random IV using the encryption key.
Data encryption generation	Data in a data frame is encrypted using AES with a random IV using the encryption key.

Table XX – Security Related Operations

The following table specifies the instantiation of the protocols in the notation from clause XX.

Notation	Definition
ID_D	The 48-bit IEEE MAC address uniquely identifying the device.
ID_SM	The 48-bit IEEE MAC address uniquely identifying the device.
PKObj_D	The device's implicit certificate object. This contains the device's implicit certificate Pub_D. It may additionally contain information such as issuer, issue date, expiry date, serial number. This is considered to be the string AdditionalInfo, and may or may not be present.
PKObj_SM	The security manager's implicit certificate object. This contains the security manager's implicit certificate Pub_SM. It may additionally contain information such as issuer, issue date, expiry date, serial number. This is considered to be the string AdditionalInfo, and may or may not be present.
Pub_D	The device's implicit certificate, which is a point on the elliptic curve ansit283k1 defined in appendix J.4.5 of ANSI X9.63.
Pr_D	The device's private key, which is an integer less than the order of the elliptic curve ansit283k1 defined in appendix J.4.5 of ANSI X9.63.
Pub_SM	The security manager's implicit certificate, which is a point on the elliptic curve ansit283k1 defined in appendix J.4.5 of ANSI X9.63.
Pr_SM	The security manager's private key, which is an integer less than the order of the elliptic curve ansit283k1 defined in appendix J.4.5 of ANSI X9.63.
P	The base point of the elliptic curve ansit283k1 defined in appendix J.4.5 of ANSI X9.63.
x	A random positive integer, generated by the device, greater than 2 and less than the order of the elliptic curve ansit283k1 defined in appendix J.4.5 of ANSI X9.63.
X	The elliptic curve point *x*P, calculated according to the techniques of ANSI X9.63
y	A random positive integer, generated by the security manager, greater than 2 and less than the order of the elliptic curve ansit283k1 defined in appendix J.4.5 of ANSI X9.63.
Y	The elliptic curve point *y*P, calculated according to the techniques of ANSI X9.63

Notation	Definition
OID	Uniquely identifies the security suite. The object identifier is the ASN.1 DER encoding of the OID as defined by ISO/ITU 8824. For this security suite, this is the hex value 0x060A2B06010401C116020102.
SSID_D	8-octet random value chosen by the security manager to uniquely identify the keys used to communicate with device D.
SSID_G	8-octet random value chosen by the security manager to uniquely identify the keys used to communicate in the piconet.
Seed_G	256-bit random value associated with a particular SSID_G used to generate the encryption key Enc_G and integrity key Int_G.
K	256-bit seed value agreed on during the MQV authentication process associated with a particular SSID_D used to generate the encryption key Enc_D and integrity key Int_D.
Enc_D	128-bit AES key associated with a particular SSID_D, to be used in AES-CBC mode as defined in [AES] and [MODES].
Enc_G	128-bit AES key associated with a particular SSID_G, to be used in AES-CBC mode as defined in [AES] and [MODES].
Int_D	128-bit HMAC-with-SHA-256 key associated with a particular SSID_D
Int_G	128-bit HMAC-with-SHA-256 key associated with a particular SSID_G
seq_num_SM	4-octet integer in network byte order associated with a particular SSID_D, used to count commands sent by the SM using that key. The sequence number shall begin counting with 0.
seq_sum_D	4-octet integer in network byte order associated with a particular SSID_D, used to count commands sent by the device using that key. The sequence number shall begin counting with 0.
SymE(m, Enc, IV)	The result of AES encryption of the message *m* with the AES key *Enc* using CBC mode with initialization vector *IV* as defined in [AES] and [MODES].
SymI(m, Int)	The result of calculating the HMAC-with-SHA-256 message authentication code on the message *m* with the 128-bit HMAC key *Int*. If *m* is "…", the message authentication code is computed over all preceding fields in the frame.
PP(h, m, K1, K2)	The result of the header *h* concatenated with the encryption *e* of message *m*, using the key *K1*, concatenated with the integrity code on *h* concatenated with *e* using the key *K2*.
H(m)	The 32-octet result of SHA-256 hash on the message *m* as defined in [FIPS180-2].

Notation	Definition
m||n	The concatenation of two messages *m* and *n.*
Key(m)	The 128-bit result of truncating the message *m* to be used as a 128-bit AES key.
AReq	Authentication Request command header
CReq	Challenge Request command header
CRes	Challenge Response command header
ARes	Authentication Response command header
KUReq	Key Update Request command header
KURes	Key Update Response command header
KRReq	Key Request command header
KRRes	Key Request Response command header
finished1	SymI(m, Int) where *m* is the entire set of data in order in the preceding protocol up to the point of the message authentication code and *Int* is Int_D.
finished2	SymI(m, Int) where *m* is the entire set of data in order in the preceding protocol up to the point of the message authentication code and *Int* is Int_D.

4. Informal Specification of Cryptographic Primitives and Protocols

4.1 Elliptic-Curves and Points

Elliptic-curve cryptography (ECC in short) is cryptography carried out in a mathematical object known as an elliptic curve over a binary field (EC in short). The elements of an elliptic curve are called points. Each point can be represented as a 2-tuple of entries in a mathematical object known as a binary field. There are three operations that may be carried out on points of an elliptic curve: addition, subtraction, and scalar multiplication. All these operations are defined in terms of simple formulas combining the entries of the points.

In elliptic curve cryptography, most computations are carried out relative to a fixed pre-specified point G of the curve, called the generating point of the curve. The order m of the generator is the size of the set of all multiples of the generating point.

4.1.1 Public Key Pairs

In a fixed elliptic curve E, a public key pair is a tuple (Q,q), where q is an integer and where Q is the point on the curve determined by multiplying the generating point G of the curve E by the private key q, i.e., $Q=qG$. Here, Q is called the public key, whereas q is called the private key.

In this specification, we use the following notation for public key pairs. A short-term (ephemeral) key pair generated by Party A is denoted by (Q_A, q_A), whereas a long-term (static) key pair generated by A is denoted by (W_A, w_A).

4.2 Messages

A message is a string of symbols (e.g., binary digits). Typically, we will use the words 'message', 'data', resp. 'information' interchangeably. By $(x|| y)$, we denote the concatenation of x and y (i.e., the result of writing the string y right behind the string x). If the order between these two strings is irrelevant, we write x, y instead.

4.3 Random Numbers

Random numbers are strings of binary digits that closely resemble the output generated by a sequence of fair coin tosses. Random numbers should not be computationally predictable, i.e., knowledge of part of the sequence describing a random number should not give any insight on any subsequent part of the describing sequence.

4.4 Key-Pair Generation

An ECC key-pair comprises a private key and its corresponding public key. As explained in Section XX, a private key is a number and its corresponding public key is an EC point, calculated by multiplying the generating point by the private key. A key-pair is generated by first generating a private key. Finding the private key from the corresponding public key is infeasible, in general, but poor choices of private keys may decrease the effort involved. The best

assurance of having good private keys is to use a good random-number generator to generate the private keys and select the output appropriately.

Even a good source of random numbers must be used appropriately. One good method to generate private keys from a random-number generator is to discard inappropriate outputs. This method is fully specified in Section XX.

4.5 Block Ciphers

A block cipher is an invertible symmetric-key encryption function that maps message blocks of a fixed length to message blocks of the same length (say n bits). The input to the block cipher is called a plaintext block, its output a ciphertext block.

We denote a symmetric-key encryption function by the symbol E_K and its inverse mapping by D_K, where K is the secret key of the block cipher. Messages and their encrypted counterparts can be related using the terminology introduced so far: if a plaintext block m is encrypted to the ciphertext block c by applying encryption function E_K hereto, i.e., $c= E_K(m)$, then message m can be recovered by applying the decryption function D_K to the ciphertext c, i.e., $m=D_K(c)$.

Message padding

One can extend the functionality of the block cipher E, by stipulating the effect of feeding incomplete inputs hereto. If x is a binary t-tuple with $0<t<n$, we define $E(x):=E(x||0)$, where $(x||0)$ denotes the right-concatenation of the string x with the string of $n-t$ zeros.

With this extended definition, the inverse block cipher D is not a 'true' inverse mapping any more, since an incomplete input string x and the padded string $(x||0)$ yield the same ciphertext. This deficiency can be remedied if the existence of padding can be reliably inferred from side information, such as a message length indicator (thus preventing ambiguities in the decryption procedure). Note that the extended encryption function defined above may result in message expansion, depending upon whether padding of the plaintext is required.

Decryption of an incomplete ciphertext block is not defined.

4.6 Cryptographic Hash Functions

A hash function is a mechanism for mapping messages of arbitrary length to messages of small fixed length. The output of the hash function, the so-called hash-value of the message, is intended to serve as a compact representative image of the message itself. For this reason, a hash-value is sometimes called a digital fingerprint or message digest. We only consider cryptographic hash functions. A basic requirement on cryptographic hash functions is that these must be difficult to forge, i.e., knowledge of a hash-value alone should not allow the effective computation of a pre-image hereof, i.e., a message that maps to this hash-value.

One distinguishes between un-keyed hash functions (or Modification Detection Codes (MDCs)) and keyed hash functions (or Message Authentication Codes (MACs)). With un-keyed hash functions, any party is able to compute the hash-value of a message, whereas, with keyed hash

functions, only those parties that have access to the secret key are capable of computing the hash-value of a message.

We denote a hash function by the symbol H (or MAC_K, if it is a keyed hash function with key K). If one uses this notation, the hash value provided over the message x is denoted by $H(x)$ (resp. $MAC_K(x)$).

4.7 Message authentication

Message authentication is the process whereby one party is assured (via corroborative evidence) of the original source of specified data created at some (typically unspecified) time in the past. Thus, message authentication provides the following assurances:

1. *Data integrity.* No undetectable modifications, including deletions and injections, of messages by unauthorized parties since the time it was created, transmitted, or stored by an authorized source;
2. *Source authentication.* No confusion about who originated the message.

Message authentication does not necessarily provide for evidence regarding uniqueness or timeliness of data, although it can be augmented to do so.

In our context, message authentication is realized via a protocol based on symmetric-key techniques, where both parties participating in the protocol share secret keying material. The protocol provides for source authentication, since only parties that share the secret key can produce properly formatted messages (assuming there is no confusion about who has access to this key).

Message authentication may be augmented to provide for freshness guarantees, by including time-variant parameters, such as random numbers and sequence numbers, within the protocol's message data. The inclusion hereof ensures that protocol messages are never repeated; hence, recorded messages from previous protocol runs that originated from a particular source cannot be re-used to impersonate this source in a future protocol run (thus preventing message replay). (Note, that this assumes each party to store previously used time-variant parameters.)

4.7.1 Message Authentication Protocol

Message authentication is realized via a 1-pass protocol based on symmetric-key techniques. The protocol assumes the prior existence of a shared secret key between all parties involved in the protocol. If no other party aside from the specifically identified group of key-sharing parties has access to this secret key, the protocol provides evidence to the party that initiated the protocol that only parties of the key-sharing group can verify the authenticity of the transferred data and, similarly, it provides evidence to each key-sharing party that the transferred data originated from one of the key-sharing parties (i.e., source authentication). For a summary of the message authentication protocol, see Table 1.

Initial set-up	- Publication of system-wide parameters - Publication of message authentication code (MAC) used - Establishment of a shared secret key K between Party A and Party B
Protocol	- Message 1 (A $\rightarrow$ B): x, $tag_A = MAC_K(x)$.

messages	(Note: part of message *x* may be communicated to B via a separate channel)
Constraints	- Access to the key *K* shall be restricted to identified group G that includes A and B
Security services	- Source authentication (to B) of string *x* relative to key-sharing group G

Table 1: Summary of the message authentication protocol.

We broadly sketch the workings of the above protocol.

1. *Initial set-up*. Each party has access to an authentic copy of the secret key shared with the other party.
2. *Communicator's actions*. The communicating party A computes a message authentication check value over a data string and communicates both to the other party B.
3. *Recipient's actions*. The recipient party B computes a message authentication check value over the data string received from the other party and compares this with the check value received from the other party, to confirm possession of the shared key (and hereby the identity of the originator of the communicated data, since the shared key is authentic).

4.8 Encryption

Encryption is the process whereby one party transforms a message (the plaintext) into an unintelligible form hereof (the ciphertext). A legitimate recipient is able to recover the original message from the ciphertext by applying the inverse process of encryption, called decryption. For other parties, this is computationally infeasible. Message encryption is realized by logically separating messages such as to ensure that only authorized parties may learn their contents. Thus, encryption provides the following assurance:

1. *Confidentiality.* No confusion about who may gain access to the plaintext.

Encryption does not necessarily provide for evidence regarding who originated the plaintext message, although it can be augmented to do so.

The encryption process depends on the underlying encryption function and the mode of operation hereof. Since the specifications hereof are public, the security of the encryption process critically depends, as a minimum, on the secrecy of the underlying decryption function (and hereby on the secrecy of the decryption key). It is important to note here, that the key required for encrypting a message and the key required for decrypting a message need not be the same. If knowledge of one of the keys allows one to effectively compute the other key, one speaks of a symmetric-key encryption scheme; otherwise, one speaks of a public-key encryption scheme. We restrict ourselves to symmetric-key encryption schemes based on block ciphers.

In our context, symmetric-key encryption and decryption is realized via a protocol based on symmetric-key techniques, where only those parties that have access to the secret key are capable of correctly decrypting ciphertext (or of encrypting pre-determined plaintext).

Encryption may be augmented to provide for authenticity of the communicated message, by including some form of redundancy, such as a message authentication code, within the message data before encrypting it. The inclusion hereof ensures that only parties that share the secret key can encrypt properly formatted plaintext data (Note, that this does assume each party to have the means to automatically verify this redundancy.) Encryption may be augmented further to provide for freshness guarantees as well, using techniques similar to those discussed for message authentication.

We denote the encryption process by $Encr_{K,IV}$ and its inverse mapping by $Decr_{K,IV}$, where K is the secret key of the underlying block cipher E_K and where IV is an additional data block needed for encryption in the applicable mode of operation (if required). Messages and their encrypted counterparts can be related using the terminology introduced so far: if a plaintext m is encrypted to the ciphertext c by applying encryption function $Encr_{K,IV}$, i.e., $c= Encr_{K,IV}(m)$, then message m can be recovered by applying the decryption function $Decr_{K,IV}$ to the ciphertext c, i.e., $m=Decr_{K,IV}(c)$. If necessary, the decryption process uses side information on the purported length of the plaintext message, to ensure non-ambiguity in the decryption process.

4.8.1 Symmetric-Key Encryption: the Cipher-Block Chaining Mode

The Cipher-Block Chaining (CBC) mode is a mode of operation of an underlying block cipher in which the encryption of a plaintext block depends on the value of the previous ciphertext block. The CBC mode requires an additional data block IV (the so-called initialization vector), which is used in the encryption and decryption of the first message block. A description follows.

By E_K, we denote the underlying block cipher, with key K. Let IV be the initialization vector, which is a publicly known, yet unpredictable value. The encryption process $Encr_{K,IV}$ transforms the ordered sequence of plaintext blocks $x_1, x_2, \ldots$ into the ordered sequence of ciphertext blocks $c_1, c_2, \ldots$, where

$$c_j := E_K(x_j \oplus c_{j-1}) \text{ for all } j>0 \text{ and where } c_0 := IV.$$

The decryption process $Decr_{K,IV}$ transforms the ordered sequence of ciphertext blocks $c_1, c_2, \ldots$into the ordered sequence of plaintext blocks $x_1, x_2, \ldots$, where

$$x_j = D_K(c_j) \oplus c_{j-1} \text{ for all } j>0 \text{ and where } c_0 := IV.$$

Encryption of a plaintext that has a length that is not a multiple of the block length of E_K is defined as the result of executing the following two steps in order: (1) right-concatenating the plaintext with as few zeros as possible such as to make its resulting length is a multiple of the block length (the so-called padding); (2) encrypting the padded plaintext. Any ambiguities in the decryption process are prevented if the existence of padding can be reliably inferred from side information, such as a message length indicator. Note that the encryption process defined above may result in message expansion, depending upon whether padding of the plaintext is required.

Decryption of a ciphertext that has a length that is not a multiple of the block length of E_K (or D_K) is not defined.

4.8.2 Symmetric-Key Encryption Protocol

Encryption is realized via a 1-pass protocol based on symmetric-key techniques. The protocol assumes the prior existence of a shared secret key between all parties involved in the protocol. If no other party aside from the specifically identified group of key-sharing parties has access to this secret key, the protocol provides evidence to the party that initiated the protocol that only parties of the key-sharing group may gain access to the plaintext (known as data confidentiality).

(We assume access to the plaintext to be restricted to the key-sharing parties only.) For a summary of the symmetric-key encryption protocol, see Table 2.

Initial set-up	- Publication of system-wide parameters - Publication of encryption function $Encr_{K,IV}$ used (and its inverse $Decr_{K,IV}$) - Establishment of a shared secret key K between Party A and Party B - Establishment of a shared IV between Party A and Party B (if required)
Protocol messages	- Message 1 (A → B): $c=Encr_{K,IV}(x)$. (Note: c is called the ciphertext corresponding to plaintext x)
Constraints	- Access to the key K shall be restricted to identified group G that includes A and B - The string IV shall satisfy the requirements for the specific mode of operation - The string x itself shall not be communicated outside the identified group G - It shall be possible to unambiguously recover plaintext from properly decrypted ciphertext (possibly using side information, such as a message length indicator)
Security services	- Confidentiality of string x, i.e., restriction of access hereto to key-sharing group G

Table 2: Summary of the symmetric-key encryption protocol.

We broadly sketch the workings of the above protocol.

1. *Initial set-up*. Each party has access to an authentic copy of the secret key shared with the other party.
2. *Communicator's actions*. The communicating party A encrypts a data string and communicates it to the other party B.
3. *Recipient's actions*. The recipient party B decrypts the received data string.

4.8.3 Combined Symmetric-Key Encryption and Message Authentication Protocol

Combined encryption and authentication is realized via a 1-pass protocol based on symmetric-key techniques. The protocol assumes the prior existence of two shared secret keys between all parties involved in the protocol: an encryption key and an integrity key. If no other party aside from the specifically identified group of key-sharing parties has access to this secret keying material, the protocol provides evidence to the party that initiated the protocol that only parties of the key-sharing group may gain access to the plaintext (known as data confidentiality) and can verify the authenticity of the transferred data; similarly, it provides evidence to each key-sharing party that the transferred data originated from one of the key-sharing parties and was encrypted hereby. (We assume access to the plaintext to be restricted to the key-sharing parties only.) For a summary of the combined symmetric-key encryption and authentication protocol, see Table 3.

Initial set-up	- Publication of system-wide parameters - Publication of encryption function $Encr_{K,IV}$ used (and its inverse $Decr_{K,IV}$) - Publication of Message Authentication Code (MAC) used - Establishment of a shared secret encryption key K_1 between Party A and Party B - Establishment of shared IV between Party A and Party B (if required) - Establishment of a shared secret integrity key K_2 between Party A and Party B
Protocol messages	- Message 1 (A → B): x, $Encr_{K1,IV}(y \| MAC)$, where $MAC=MAC_{K2}(x \| y)$. (Note: part of the string x may be communicated to B via a separate channel)
Constraints	- Access to the keys K_1 and K_2 shall be restricted to identified group G that includes A and B

	- The string *y* itself shall not be communicated outside the identified group G - It shall be possible to unambiguously determine *y* and the MAC value from properly decrypted ciphertext (possibly using side information, such as a message length indicator)
Security services	- Confidentiality of string *y*, i.e., restriction of access hereto to key-sharing group G - Source authentication of strings *x* and *y* relative to key-sharing group G

Table 3: Summary of the combined symmetric-key encryption and message authentication protocol.

We broadly sketch the workings of the above protocol. For details we refer to …

1. *Initial set-up*. Each party has access to an authentic copy of the secret keys shared with the other party.
2. *Communicator's actions*. The communicating party A computes a message authentication check value over two data strings and subsequently encrypts the check value and one of the data strings. It communicates the resulting ciphertext and the non-encrypted string to the other party B.
3. *Recipient's actions*. The recipient party B decrypts the received ciphertext and extracts the check value and one of the data strings from this. Subsequently, it computes the message authentication check value over the (now recovered) data strings received from the other party and compares this with the check value received from the other party, to confirm possession of the shared integrity key (and hereby the identity of the originator of the communicated data, since the shared integrity key is authentic).

4.9 Key Establishment

Key establishment is the process whereby a shared secret key becomes available to two or more parties, for subsequent cryptographic use. Secure key establishment protocols provide assurances that no party aside from some specifically identified party (or parties) may gain access to a particular secret key. In addition, these protocols may provide assurances to a party as to the actual possession of secret keying material by some other party. Thus, key establishment protocols may provide assurances, including the following:

1. *Key authentication*. No confusion about whom an entity might share a secret value with;
2. *Key confirmation*. Evidence that some entity actually possesses or has access to a secret key.

Explicit key authentication refers to providing both these assurances at the same time.

One distinguishes between key agreement and key transport protocols. Key transport is a key establishment technique where one party creates or otherwise obtains a secret value and securely transfers it to other parties. Key agreement is a key establishment technique in which two or more parties derive a shared secret key based upon information contributed by, or associated with, each of these, ideally such that no party can predetermine the resulting value. Thus, with key transport, one party may control the value of the secret key, whereas, with key agreement, such unilateral key control is generally not possible.

In our context, key agreement is realized via a protocol based on public-key techniques, where each party participating in the protocol has access to an authentic copy of the other party's public key, and on unpredictable contributions by both parties involved in the protocol. The protocol

provides for mutual key authentication, since only parties that possess the private key corresponding to their purported public key can correctly determine the shared secret key (assuming there is no confusion about who has access to this private key, i.e., assuming a properly certified binding between the entity's identity and its public key). Moreover, it provides for key confirmation, since the key is derived from unpredictable inputs provided by both parties that are never repeated; hence, recorded messages from previous protocol runs do not leak information that could be used to impersonate a party involved in a future protocol run.

In our context, key transport is realized via a protocol based on symmetric-key techniques, where all parties participating in the protocol share secret keying material, and on non-repeating contributions by this key-sharing group. The protocol provides for source authentication relative to the key-sharing group, since only parties that possess the secret key can produce properly formatted protocol messages (assuming there is no confusion about who has access to this key). Moreover, freshness guarantees are provided, since protocol messages are never repeated; hence, recorded messages from previous protocol runs do not leak information that could be used to impersonate a party involved in a future protocol run.

4.9.1 Key Agreement Protocol

Key agreement is realized via a 3-pass authenticated key agreement with key confirmation protocol based on public-key techniques, as well as on unpredictable and non-repeating contributions by both parties involved in the protocol. The protocol assumes each of the two parties involved in the protocol to have access to an authentic copy of the other party's public key. If the binding between each party's identity and its public key is properly certified[1], the protocol provides evidence to the party that initiated the protocol on the true identity of the responding party with whom it establishes a shared secret key (i.e., key authentication) and on the active participation hereof in the protocol communications (via key confirmation), and vice versa. Optionally, the protocol may provide source authentication of a data string communicated by the responding party (as part of the protocol messages) to the initiating party and confirmation of proper receipt by the responding party of a data string communicated hereto by the initiating party, and vice versa. For a summary of the authenticated key agreement with key confirmation protocol, see Table 1.

Initial set-up	- Publication of system-wide parameters - Publication of public-key parameters (shared elliptic curve domain parameters) - Publication of message authentication function (MAC) used - Exchange of authentic static public keys W_A and W_B between Parties A and B
Protocol messages	Message 1 (A $\rightarrow$ B): Q_A, x_2. Message 2 (A $\leftarrow$ B): $Q_B, tag_B = MAC_K(Id_B \| Id_A \| Q_B \| Q_A \| y_1 \| x_2), y_1, y_2$. Message 3 (A $\rightarrow$ B): $tag_A = MAC_K(Id_A \| Id_B \| Q_A \| Q_B \| x_1 \| y_2), x_1$. (Note: the strings x_1, x_2, y_1, y_2 are optional)
Constraints	- Q_A and Q_B shall be generated at random (ephemeral public keys)

[1] Proper certification of a public key depends on appropriately checking the credentials of a party A with claimed public key P_A and involves the following two steps: (1) Checking, by cryptographic means, that the entity A has access to the private key S_A corresponding to P_A (the so-called 'proof of possession'); (2) Checking, by non-cryptographic means, the purported identity Id_A of A (the so-called 'proof of identity').

	- w_A and w_B shall be privately held by A, resp. B - w_A and w_B shall be valid during the execution of the protocol (Note: (W_A, w_A) and (W_B, w_B) are public key pairs of A, resp. B)
Security services	- Key agreement between A and B on the shared key K - Mutual entity authentication - Mutual explicit key authentication - Perfect forward secrecy; unknown key-share resilience; known-key security - (Optional) Authentication of string x_1 by A, resp. of string y_1 by B - (Optional) Confirmation of proper receipt of string x_2 by B, resp. of string y_2 by A

Table 4: Summary of the Authenticated Key Agreement with Key Confirmation Protocol.

The actual workings of the protocol depend on the public key system at hand. We broadly sketch Full MQV with Key Confirmation, an instantiation of the above protocol based on elliptic curve techniques. Instantiations of the above protocol based on ordinary public key techniques are, however, also possible, using the same formats.

1. *Initial set-up*. Each party has access to the system-wide elliptic curve parameters of some elliptic curve. Each party randomly generates a long-term (static) public key pair and publishes the public key (but not the private key). We assume each party to have access to an authentic copy of the other party's static public key.
2. *Key contributions*. Each party randomly generates a short-term (ephemeral) public key pair and communicates this ephemeral public key to the other party (but not the private key).
3. *Key establishment*. Let (W_A, w_A) be Party A's static public key pair and let (Q_A, q_A) be its ephemeral public key pair. Let (W_B, w_B) and (Q_A, q_A) be the corresponding public key pairs of Party B. Each party computes the shared key based on the static and ephemeral elliptic curve points it received from the other party and based on the static and ephemeral private keys it generated itself. Here, Party A computes $K_A=(q_A + map(Q_A)\, w_A)\, (Q_B + map(Q_B)W_B)$, whereas Party B computes $K_B=(q_B + map(Q_B)\, w_B)\, (Q_A + map(Q_A)W_A)$. Here, *map* is a fixed function that maps elliptic curve points to integers. Due to the properties of elliptic curves, both keys are the same, i.e., $K_A=K_B$, and can, indeed, be computed by either party.
4. *Key authentication.* Each party verifies the authenticity of the long-term static key of the other party, to obtain evidence that the only party that may be capable of computing the shared key is, indeed, its perceived communicating party.
5. *Key confirmation*. Each party computes a message authentication check value over the strings communicated with the other party, to prove possession of the shared key to the other party. This confirms to each party the true identity of the other party and proofs that that party successfully computed the shared key.

4.9.2 Key Transport Protocol

Key transport is realized via a 1-pass authenticated key transport protocol based on symmetric-key techniques, as well as on non-repeating contributions by all parties involved in the protocol (to prevent message replay). The protocol assumes the prior existence of two shared secret keys between all parties involved in the protocol: an encryption key and an integrity key. If no other party aside from the specifically identified group of key-sharing parties has access to this secret key, the protocol provides evidence to the party that initiated the protocol that only parties of the key-sharing group may gain access to the transported key (i.e., key authentication) and can

verify the authenticity of the transported key; similarly, it provides evidence to each key-sharing party that the transported key originated from one of the key-sharing parties (i.e., source authentication). Optionally, the protocol may provide source authentication of a data string communicated by the initiating party (as part of the protocol message) to each key-sharing party. For a summary of the authenticated key transport protocol, see Table 2.

4.10 Entity Authentication

Entity authentication is the process whereby one party is assured (through the acquisition of corroborative evidence) of the true identity of a second party involved in a protocol, and that this second party has actually participated in the protocol (i.e., is active at, or immediately prior to, the time the evidence was acquired). Thus, entity authentication provides the following assurances:

1. *Authenticity.* No confusion about whom an entity is really communicating with;
2. *'Aliveness'.* Evidence that this entity is actively participating in the communications.

In our context, entity authentication is realized via a challenge-response protocol based on symmetric-key techniques, where both parties participating in the protocol share secret keying material. The protocol provides for source authentication, since only parties that share the secret key can produce proper responses to unpredictable challenges (assuming there is no confusion about who has access to this key). Moreover, it provides for aliveness guarantees, since challenge messages are unpredictable and never repeated; hence, recorded messages from previous protocol runs do not leak information that could be used to impersonate a party involved in a future protocol run.

4.10.1 Unilateral Entity Authentication Protocol

Unilateral entity authentication is realized via a 2-pass challenge-response protocol based on symmetric-key techniques. The protocol assumes the prior existence of a shared secret key between the two parties involved in the protocol. If no other party has access to this secret key, the protocol provides evidence to the party that initiated the protocol on the true identity of the responding party and on the active participation hereof in the protocol communications. Optionally, the protocol may provide source authentication of a data string communicated by the responding party (as part of the protocol messages) to the initiating party and confirmation of proper receipt by the responding party of a data string communicated hereto by the initiating party. For a summary of the unilateral entity authentication protocol, see Table 1.

Initial set-up	- Publication of system-wide parameters - Publication of message authentication code (MAC) used - Establishment of a shared secret key K between Party A and Party B
Protocol messages	Message 1 (A $\rightarrow$ B): RND_A, x_2. Message 2 (A $\leftarrow$ B): RND_B, $tag_B = MAC_K(Id_B \| Id_A \| RND_B \| RND_A \| y_1 \| x_2)$, y_1. (Note: the strings x_2 and y_1 are optional)
Constraints	- RND_A and RND_B shall be generated at random - Access to the key K shall be restricted to A and B
Security services	- Unilateral entity authentication from B to A - (Optional) Authentication of string y_1 by B

	- (Optional) Confirmation of proper receipt of string x_2 by B

Table 5: Summary of the unilateral entity authentication protocol.

We broadly sketch the workings of the above protocol.

1. *Initial set-up*. Each party has access to an authentic copy of the secret key shared with the other party.
2. *Challenge contributions*. The challenging party A randomly generates a random number and communicates this challenge to the other party.
3. *Key confirmation*. The responding party B computes the message authentication check value over the strings communicated with the other party, to prove possession of the shared key to the other party (and hereby its identity, since the shared key is authentic). The challenging party verifies the check value communicated by the other party, to confirm possession of the shared secret key hereby.
4. *Aliveness guarantees*. The initiating party confirms aliveness of the other party by ensuring that its challenge contribution is random and 'fresh'.

4.10.2 Mutual Entity Authentication Protocol

Mutual entity authentication is realized via a 3-pass challenge-response protocol based on symmetric-key techniques. The protocol assumes the prior existence of a shared secret key between the two parties involved in the protocol. If no other party has access to this secret key, the protocol provides evidence to the party that initiated the protocol on the true identity of the responding party and on the active participation hereof in the protocol communications, and vice versa. Optionally, the protocol may provide source authentication of a data string communicated by the responding party (as part of the protocol messages) to the initiating party and confirmation of proper receipt by the responding party of a data string communicated hereto by the initiating party, and vice versa. For a summary of the mutual entity authentication protocol, see Table 3.

Initial set-up	- Publication of system-wide parameters - Publication of message authentication code (MAC) used - Establishment of a shared secret key K between Party A and Party B
Protocol messages	Message 1 (A $\rightarrow$ B): RND_A, x_2. Message 2 (A $\leftarrow$ B): $RND_B, tag_B = MAC_K (Id_B \| Id_A \| RND_B \| RND_A \| y_1 \| x_2), y_1, y_2$. Message 3 (A $\rightarrow$ B): $tag_A = MAC_K (Id_A \| Id_B \| RND_A \| RND_B \| x_1 \| y_2), x_1$. (Note: the strings x_1, x_2, y_1, y_2 are optional)
Constraints	- RND_A and RND_B shall be generated at random - Access to the key K shall be restricted to A and B
Security services	- Mutual entity authentication - (Optional) Authentication of string x_1 by A, resp. of string y_1 by B - (Optional) Confirmation of proper receipt of string x_2 by B, resp. of string y_2 by A

Table 6: Summary of the mutual entity authentication protocol.

We broadly sketch the workings of the above protocol.

1. *Initial set-up*. Each party has access to an authentic copy of the secret key shared with the other party.
2. *Challenge contributions*. Each party randomly generates a random number and communicates this challenge to the other party.
3. *Key confirmation*. Each party computes the message authentication check value over the strings communicated with the other party, to prove possession of the shared key to the other party (and hereby its identity, since the shared key is authentic). Each party verifies the check value communicated by the other party to confirm possession of the shared key hereby.
4. *Aliveness guarantees*. Each party confirms aliveness of the other party by ensuring that its challenge contribution is random and 'fresh'.

4.11 Public-Key Certificates

Proper certification of a public key depends on appropriately checking the credentials of a party A with claimed public key P_A and involves the following two steps: (1) Checking, by cryptographic means, that the entity A has access to the private key S_A corresponding to P_A (the so-called 'proof of possession'); (2) Checking, by non-cryptographic means, the purported identity Id_A of A (the so-called 'proof of identity'). We distinguish the following types of certificates:

1. *Public key certificates.* The external trusted party T provides his signature over A's public key and A's identifying information and includes these data in a public key certificate. Generation of the public key certificate requires interaction between Party A and the external trusted party, to corroborate evidence as to A's true identity and as to its possession of the corresponding private key. Verification of the authenticity of A's public key requires the signature of Party T, as contained in the public key certificate, to be verified. Thus, it only necessitates the verifying party to have access to an authentic copy of the (public) signature verification key of Party T, rather than to the public keys of all its potential communicating parties.
2. *Implicit certificates.* This provides an alternative for public key certificates. The main idea here is that it is not necessary to store public keys as such; storage of related data instead, plus an efficient procedure for reconstructing public keys from these, suffices. With implicitly certified public keys, the public key is reconstructed based upon the identity of the device the public key is associated with and the previously mentioned related data. In addition, evidence regarding the authenticity of the reconstructed public key string is provided. As with public key certificates, the generation of implicit certificates requires interaction with the external trusted party.
3. *Manual certificates.* This provides a mechanism to form a compliant certificate based on a predetermined action of a user that establishes a binding of the public key with that action to "certify" that the public key belongs to an entity. The result is useable with cryptographic authentication means to automatically identify the entity.

The suitability of either implementation method depends on the choice of public key primitive.

The above description assumes that each entity generates its own public key pair(s). As an alternative, the trusted party might generate these on behalf of the entity, provide his signature hereover, and have the public key pair securely stored on the device, during the manufacturing hereof (see §10.3.1.1).

In all cases, an authentic copy of the trusted party's public signature verification key must be stored in each device, prior to its operational deployment (i.e., either at the time of manufacturing of the device or at personalization of the device).

In our context, public keying material is created once and for all and is never revoked, nor is the public signature verification key of the external trusted party. The reason for this design choice is that, with short-range communications technology, one cannot rely on on-line centralized key management (although off-line centralized key management would have been possible). For details, see §10.1.2.

Notes
It is important to realize that the different methods for generating authentic public key pairs, as discussed above, require different levels of trust in the trusted party. If the trusted party both generates and authenticates an entity's public key pair, it must be trusted not to disclose the private key and not to create false credentials. If the trusted party merely authenticates a public key that was generated by the entity itself, it can still create false credentials if certificates are used; with public key certificates, it can completely control the public key value, whereas with implicit certificates key control this is not possible. For details, we refer to [9, Remark 13.7]. Last but not least, a party that generates its own keying material must have sufficient assurances as to the quality of its random number generator.

A more detailed discussion of this topic is beyond the scope of this document.

4.11.1 Authenticity of digital signatures

We assume the *external* presence of a distinguished party T who vouches for the authenticity of the binding between an entity A and its (public) signature verification string P_A.

The actual implementation of this binding, and a verification mechanism therefore, depends on context.
Usually, this binding is implemented by provision of a signature by T over the pair (A, P_A) and by inclusion of these data in the certificate $Cert_T(A, P_A)$. In its basic form, verification of this binding now only requires the signature of Party T over the pair (A, P_A), as contained in the string $Cert_T(A, P_A)$, to be verified. Thus, it only necessitates the verifying party to have access to an (authentic copy of) the public verification string $VALSign_T$ of Party T (rather than to the public key strings of all its (potential) communicating parties). In closed systems, this authentic binding could also be provided differently, e.g., by storage of bindings in an authentic hardware module. Depending upon the actual implementation environment, we therefore make the following assumption:

1. In the presence of a (rudimentary) external certificate structure, we assume the authentic transfer of T's public verification string P_T to Party X to be realized via out-of-band means, at configuration of the card of Party X[2].
2. In the absence of a certificate structure, we assume the authentic bindings between entities and their signature verification strings to be realized once and for all via out-of-band means, by incorporation of these data, during the configuration of each card.

Lifecycle issues
Signature verification requires verification of the authenticity and validity of signature verification strings. Here we consider the effect of changes to these. We assume a setting with certificates.

1. *Introduction of a new entity*. This requires off-line involvement of the trusted party of that entity, who should create and publish the corresponding certificate, for future use by other entities. We assume the new entity to be already endowed with a private signature key that is securely stored on its card, maintaining integrity.
2. *Removal of an entity*. This requires off-line involvement of the trusted party of that entity, who should revoke the corresponding certificate.
3. *Update of long-term keying material of a trusted party*. This requires the transfer of an update of the authentic signature verification string P_T of trusted party T. Hence, updates of long-term keys require the presence of an authentic channel with all entities in the system.

In the absence of a certificate structure, authentic updates of signature verification strings might be realized by out-of-band (i.e., non-cryptographic) means or, alternatively, via the execution of an authentic data transfer protocol involving another trusted party that is already present. A detailed discussion of this topic is beyond the scope of this document.

4.11.2 Certificate Issuance

Certificate issuance is the process whereby a device obtains a certificate from a certificate-issuing authority. The output of the process is a certificate and is beyond the scope of this standard.

4.11.3 Implicit Certificates

Implicit certificates are an efficient alternative to explicit certificates, especially X.509 certificates. One piece of data yields both the device's public key and the implicit signature of its issuer. The protocol described herein allows for more than one issuer by including an optional field containing information that specifies the issuer, thus allowing the recipient to retrieve the appropriate public key. The process of computing the device's public key from the implicit certificate is known as reconstructing the public key.
For a summary of implicit certificate use, see the following Table.

Initial set-up	- Publication of system-wide parameters - Publication of issuer's public key W_T. - Publication of cryptographic hash H.

[2] If communication requires verification of signatures provided by Parties A and B that resort under different trusted parties, then cross-certification is required.

Protocol	Step 1 (Receive Party A's implicit certificate): $ImpCert_T(Id_A) = (Id_A,D,IssuerInfo)$. Step 2 (Retrieve issuer's public key): W_T. Step 2 (Compute Party A's public key): $W_A := H(Id_A\|\|D).D + W_T$.
Constraints	- Party B has trusted copy of issuer's public key
Security services	-

Table 7 Summary of implicit certificates.

4.11.4 X.509 Certificates

Certificates defined by ISO X.509 (as part of The Directory) are explicit certificates. A description and use of these certificates may be found in RFC2459 or X.509.

5. Formal Specification of Cryptographic Primitives and Protocols

5.1 Elliptic-Curves and Points

This standard uses the specification of elliptic curves as described in ANSI X9.63-2001. The elliptic curves are defined over a binary field using a polynomial representation. Points are represented in a compressed format.

Elliptic curve points are stored as octet strings. Each point is represented as a tuple with entries in a binary field. In our curve, each element of the binary field requires 46 bytes of storage, thus requiring 92 bytes of storage for the point. (A further classification byte is required by most standards.) A space optimization, called point compression, allows a point to be stored in 47 bytes, at negligible computational overhead. Descriptions of point compression and decompression may be found in Sec. 4.2.2 of ANSI X9.63-2001.

5.2 Challenge Validation Primitive

Challenge validation refers to the process of checking the length properties of a challenge. It is used to check whether the challenge to be used by the schemes in the standard have a length that is sufficient for the scheme at hand.

Input: The input of the validation transformation is a valid challenge domain parameter D=[*minchallengelen, maxchallengelen*] together with the bit string *Challenge*.

Actions: The following checks are made:

1. Calculate the length *challengelen* of the bit string *Challenge*.

Output: If *challengelen* $\in D$, then output ‘valid’; otherwise, output ‘invalid’.

5.3 Unilateral Entity Authentication Scheme

This section specifies the unilateral entity authentication scheme. A MAC scheme is used to provide key confirmation.

Figure 12 illustrates the messaging involved in the use of the unilateral entity authentication scheme.

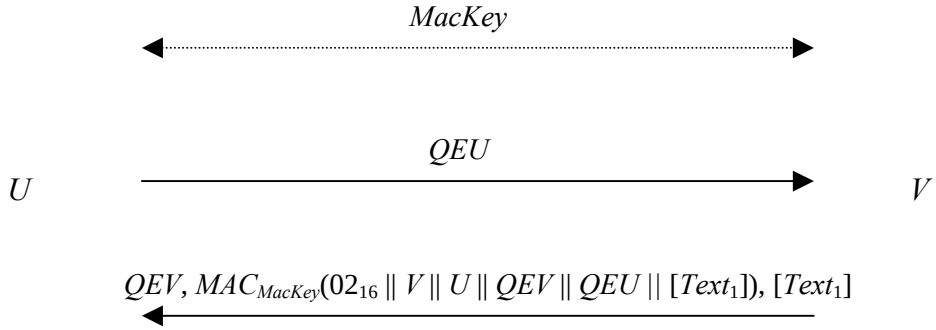


Figure 12 – Unilateral Entity Authentication Scheme

The scheme is 'asymmetric', so two transformations are specified. *U* uses the transformation specified in Section 6.11.1 to obtain evidence from *V* as to the actual involvement hereof in a real-time communication with *U* if *U* is the protocol's initiator. *V* uses the transformation specified in Section 6.11.2 to provide this evidence to *U* if *V* is the protocol's responder.

If *U* executes the initiator transformation and *V* executes the responder transformation and if access to the shared key is restricted to *U* and *V*, then *U* will obtain unilateral entity authentication of *V*.

Prerequisites: The following are the prerequisites for the use of the scheme:

1. Each entity has access to a bit string *MacKey* of length *mackeylen* bits to be used as the key. Each party shall have evidence that access to this key is restricted to the entity itself and the other entity involved in the entity authentication scheme.

2. Each entity has an authentic copy of the system's challenge domain parameters *D*=[*minchallengelen*, *maxchallengelen*].

3. Each entity shall be bound to a unique identifier (e.g. distinguished names). All identifiers shall be bit strings of the same length *entlen* bits. Entity *U*'s identifier will be denoted by the bit string *U*. Entity *V*'s identifier will be denoted by the bit string *V*.

4. Each entity shall have decided which ANSI-approved MAC scheme to use as specified in Section 5.7. The length in bits of the keys used by the MAC scheme is denoted by *mackeylen*.

5.3.1 Initiator Transformation

U shall execute the following transformation to obtain evidence from *V* as to the actual involvement hereof in a real-time communication with *U* if *U* is the protocol's initiator. *U* shall obtain an authentic copy of *V*'s identifier and an authentic copy of the key *MacKey* shared with *V*.

Input: This routine does not take any input.

Ingredients: The initiator transformation employs the challenge generation primitive in Section 5.3 of ANSI X9.63 - 2001, challenge validation in Section 5.2, and uses the truncated HMAC function described in FIPS Pub #HMAC using as the hash function SHA-256 described in FIPS Pub 180-2.

Actions: Entity authentication shall be established as follows:

1. Use the challenge generation primitive in Section 5.3 of ANSI X9.63 – 2001 to generate a challenge *QEU* for the parameters *D*. Send *QEU* to *V*.

2. Then receive from *V* a challenge *QEV*' purportedly owned by *V*, an optional bit string $Text_1$, and a purported tag $MacTag_1$'. If these values are not received, output 'invalid' and stop.

3. Verify that *QEV*' is a valid challenge for the parameters *D* as specified in Section 5.2. If the validation primitive rejects the challenge, output 'invalid' and stop.

4. Form the bit string consisting of the octet 02_{16}, *V*'s identifier, *U*'s identifier, the bit string *QEV*', the bit string *QEU*, and if present $Text_1$:

$$MacData_1 = 02_{16} \| V \| U \| QEV' \| QEU \| [Text_1].$$

5. Verify that $MacTag_1$' is the tag for $MacData_1$ under the key *MacKey* using the tag checking transformation of the appropriate HMAC scheme previously specified. If the tag checking transformation outputs 'invalid', output 'invalid' and stop.

Output: If any of the above verifications has failed, then output 'invalid' and reject the authentication of *V*. Otherwise, output 'valid', accept the authentication of *V* and accept *V* as the source of the bit string $Text_1$ (if present).

5.3.2 Responder Transformation

V shall execute the following transformation to provide evidence to *U* as to its actual involvement in a real-time communication with *U* if *V* is the protocol's responder. *V* shall obtain an authentic copy of *U*'s identifier and an authentic copy of the key *MacKey* shared with *U*.

Input: The input to the responder transformation is:

1. A challenge *QEU*' purportedly owned by *U*.

Ingredients: The initiator transformation employs the challenge generation primitive in Section 5.3 of ANSI X9.63 - 2001, challenge validation in Section 5.2, and uses the truncated HMAC function described in FIPS Pub #HMAC using as the hash function SHA-256 described in FIPS Pub 180-2.

Actions: Entity authentication shall be established as follows:

1. Verify that *QEU*' is a valid challenge for the parameters *D* as specified in Section 5.2. If the validation primitive rejects the challenge, output 'invalid' and stop.

2. Use the challenge generation primitive in Section 5.3 of ANSI X9.63 – 2001 to generate a challenge *QEU* for the parameters *D*. Send *QEU* to *V*.

3. Form the bit string consisting of the octet 02_{16}, *V*'s identifier, *U*'s identifier, the bit string *QEV*, the bit string *QEU*', and, optionally, a bit string $Text_1$:

   $$MacData_1 = 02_{16} \| V \| U \| QEV \| QEU' \| [Text_1].$$

4. Calculate the tag $MacTag_1$ for $MacData_1$ under the key *MacKey* using the tagging transformation of the appropriate HMAC scheme previously specified.

   $$MacTag_1 = MAC_{MacKey}(MacData_1).$$

   If the tagging transformation outputs 'invalid', output 'invalid' and stop. Send to *U* the challenge *QEV*, if present the bit string $Text_1$, and $MacTag_1$.

Output: If any of the above verifications has failed, then output 'invalid' and stop; otherwise, output 'valid'.

5.4 Mutual Entity Authentication Scheme

This section specifies the mutual entity authentication scheme. A MAC scheme is used to provide key confirmation.

Figure 12 illustrates the messaging involved in the use of the full mutual entity authentication scheme.

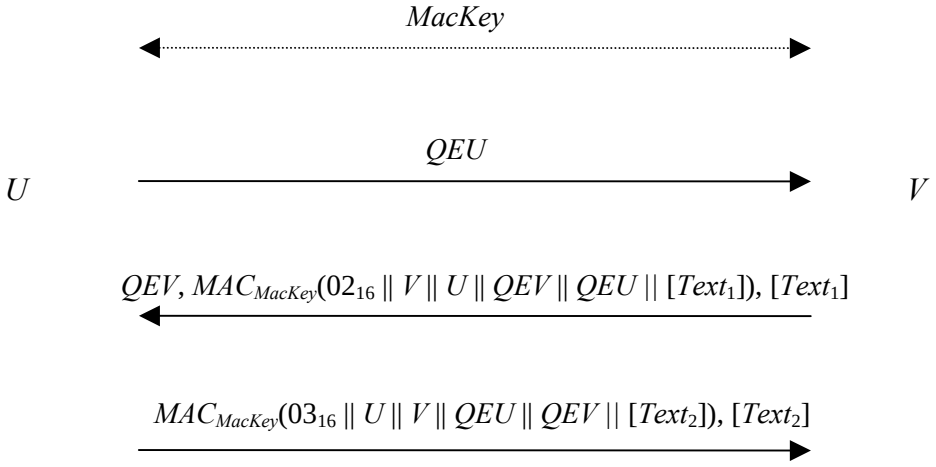


Figure 12 – Mutual Entity Authentication Scheme

The scheme is 'asymmetric', so two transformations are specified. *U* uses the transformation specified in Section 6.12.1 of ANSI X9.63 – 2001 to obtain evidence from *V* as to the actual involvement hereof in a real-time communication with *U* and to provide similar evidence to *V* if *U* is the protocol's initiator. *V* uses the transformation specified in Section 6.12.2 of ANSI X9.63 – 2001 to provide evidence to *U* as to its actual involvement in a real-time communication with *U* and to obtain similar evidence from *U* if *V* is the protocol's responder.

The essential difference between the role of the initiator and the role of the responder is merely that the initiator sends the first pass of the exchange.

If *U* executes the initiator transformation and *V* executes the responder transformation and if access to the shared key is restricted to *U* and *V*, then *U* and *V* will obtain mutual entity authentication.

Prerequisites: The following are the prerequisites for the use of the scheme:

1. Each entity shall have access to a bit string *MacKey* of length *mackeylen* bits to be used as the key. Each party shall have evidence that access to this key is restricted to the entity itself and the other entity involved in the entity authentication scheme.

2. Each entity has an authentic copy of the system's challenge domain parameters *D*=[*minchallengelen*, *maxchallengelen*].

3. Each entity shall be bound to a unique identifier (e.g. distinguished names). All identifiers shall be bit strings of the same length *entlen* bits. Entity *U*'s identifier will be denoted by the bit string *U*. Entity *V*'s identifier will be denoted by the bit string *V*.

4. Each entity shall use the truncated HMAC scheme. The length in bits of the keys used by the MAC scheme is denoted by *mackeylen*.

5.4.1 Initiator Transformation

U shall execute the following transformation to obtain evidence from *V* as to the actual involvement hereof in a real-time communication with *U* and to provide similar evidence to *V* if *U* is the protocol's initiator. *U* shall obtain an authentic copy of *V*'s identifier and an authentic copy of the key *MacKey* shared with *V*.

Input: This routine does not take any input.

Ingredients: The initiator transformation employs the challenge generation primitive in Section 5.3 of ANSI X9.63 - 2001, challenge validation in Section 5.2, and uses the truncated HMAC function described in FIPS Pub #HMAC using as the hash function SHA-256 described in FIPS Pub 180-2.

Actions: Entity authentication shall be established as follows:

1. Use the challenge generation primitive in Section 5.3 of ANSI X9.63 - 2001to generate a challenge *QEU* for the parameters *D*. Send *QEU* to *V*.

2. Then receive from *V* a challenge *QEV*' purportedly owned by *V*, an optional bit string $Text_1$, and a purported tag $MacTag_1$'. If these values are not received, output 'invalid' and stop.

3. Verify that *QEV*' is a valid challenge for the parameters *D* as specified in Section 5.2. If the validation primitive rejects the challenge, output 'invalid' and stop.

5. Form the bit string consisting of the octet 02_{16}, *V*'s identifier, *U*'s identifier, the bit string *QEV*', the bit string *QEU*, and if present $Text_1$:

$$MacData_1 = 02_{16} \| V \| U \| QEV' \| QEU \| [Text_1].$$

5. Verify that $MacTag_1$' is the tag for $MacData_1$ under the key *MacKey* using the tag checking transformation of the appropriate MAC scheme specified in Section 5.7. If the tag checking transformation outputs 'invalid', output 'invalid' and stop.

6. Form the bit string consisting of the octet 03_{16}, *U*'s identifier, *V*'s identifier, the bit string *QEU*, the bit string *QEV*', and optionally a bit string $Text_2$:

$$MacData_2 = 03_{16} \| U \| V \| QEU \| QEV' \| [Text_2].$$

7. Calculate the tag $MacTag_2$ on $MacData_2$ under the key *MacKey* using the tagging transformation of the appropriate HMAC scheme previously specified:

$$MacTag_2 = MAC_{MacKey}(MacData_2).$$

If the tagging transformation outputs 'invalid', output 'invalid' and stop. Send $MacTag_2$ and if present $Text_2$ to *V*.

Output: If any of the above verifications has failed, then output 'invalid' and reject the authentication of *V*. Otherwise, output 'valid', accept the authentication of *V* and accept *V* as the source of the bit string $Text_1$ (if present).

5.4.2 Responder Transformation

V shall execute the following transformation to provide evidence to *U* as to its actual involvement in a real-time communication with *U* and to obtain similar evidence from *U* if *V* is the protocol's responder. *V* shall obtain an authentic copy of *U*'s identifier and an authentic copy of the key *MacKey* shared with *U*.

Input: The input to the responder transformation is:

1. A challenge *QEU*' purportedly owned by *U*.

Ingredients: The responder transformation employs the challenge generation primitive in Section 5.3 of ANSI X9.63 - 2001, challenge validation in Section 5.2, and uses the truncated HMAC function described in FIPS Pub #HMAC using as the hash function SHA-256 described in FIPS Pub 180-2.

Actions: Entity authentication shall be established as follows:

1. Verify that *QEU*' is a valid challenge for the parameters *D* as specified in Section 5.2. If the validation primitive rejects the challenge, output 'invalid' and stop.

2. Use the challenge generation primitive in Section 5.3 of ANSI X9.63 - 2001 to generate a challenge *QEU* for the parameters *D*. Send *QEU* to *V*.

3. Form the bit string consisting of the octet 02_{16}, *V*'s identifier, *U*'s identifier, the bit string *QEV*, the bit string *QEU*', and, optionally, a bit string $Text_1$:

$$MacData_1 = 02_{16} \| V \| U \| QEV \| QEU' \| [Text_1].$$

4. Calculate the tag $MacTag_1$ for $MacData_1$ under the key *MacKey* using the tagging transformation of the appropriate HMAC scheme previously specified.

   $MacTag_1 = MAC_{MacKey}(MacData_1)$.

   If the tagging transformation outputs 'invalid', output 'invalid' and stop. Send to *U* the challenge *QEV*, if present the bit string $Text_1$, and $MacTag_1$.

5. Then receive from *U* an optional bit string $Text_2$ and a purported tag $MacTag_2$'. If this data is not received, output 'invalid' and stop.

6. Form the bit string consisting of the octet 03_{16}, *U*'s identifier, *V*'s identifier, the bit string *QEU*', the bit string *QEV*, and the bit string $Text_2$ (if present):

   $MacData_2 = 03_{16} \| U \| V \| QEU' \| QEV \| [Text_2]$.

7. Verify that $MacTag_2$' is the valid tag on $MacData_2$ under the key *MacKey* using the tag checking transformation of the appropriate HMAC scheme previously specified. If the tag checking transformation outputs 'invalid', output 'invalid' and stop.

Output: If any of the above verifications has failed, then output 'invalid' and reject the authentication of *U*. Otherwise, output 'valid', accept the authentication of *U* and accept *U* as the source of the bit string $Text_2$ (if present).

5.5 Reconstructing the Public Key from an Implicit Certificate

Recipient device *U* shall execute the following steps to reconstruct device *V*'s public key from the implicit certificate *I*.

Input: Implicit certificate $I = (Id_V, D, IssuerInfo)$.

Ingredients: The reconstruction employs as the hash function *H* the hash function SHA-2 of FIPS Pub 180-2.

Actions: The public key shall be reconstructed as follows:

1. Retrieve the public key W_T corresponding to *IssuerInfo*. Output `invalid' if there is no public key corresponding to *IssuerInfo*.

2. Extract the components Id_V and *D* from *I* and calculate the hash of their concatenation. $h = H(Id_V || D)$.

3. Convert the octet string *h* of the previous step into an integer *e* following the conversion routine of Step 5 in Section 4.1.3 of SEC1.

4. Convert the octet string D into an elliptic-curve point W_D using the conversion routine specified in Section 2.3.4 of SEC1.

5. Perform the following elliptic-curve point calculations to obtain the public key.

$$W_V = e\, W_D + W_T.$$

Output: Device V's public key W_V.

6. Notation

Cryptographic Building Blocks		
Block cipher	E_k (with inverse function D_k)	AES-128
Block size	B	128
Symmetric key	K	Generated
Cryptographic bit-strength	*keysize*	128
Initialization vector	IV	Generated
Forward cipher function	$Encr_{K,IV}$ (with inverse $Decr_{K,IV}$)	E_K in CBC mode
		E_K in CTR mode
Un-keyed Hash function	H	SHA-256
Message Authentication Code	MAC_K	*HMAC* using *H*, truncated to *maclen* bits
-MAC length	*maclen*	*keysize*
Challenge	*RND*	Generated
-Challenge length	*keysize*	
Binary elliptic curve	$E=(m,f(x),n,h,G,a,b)$	Koblitz curve *K-283*
-Underlying binary field	$F_2^m=F_2[x]/(f(x))$	
-Generating point	G	
-Order of generating point	n	
-Cofactor of curve	h	4
-Elliptic curve equation	$Y^2+XY=X^3 + aX +b$	a=0, b=1
Elliptic curve point	$P=(x_P, y_P)$	
	$P=(x_P, \hat{y}_P)$	Point compression
Global device identifier	Id_A	48-bit IEEE MAC Address
Local device identifier	Id_A^*	8-bit piconet device address
Static public key (of entity A)	W_A	Static elliptic curve point
Static private key (of entity A)	S_A such that $W_A=S_A\ G$	Integer in range [1,n-1]
Ephemeral public key (of A)	Q_A	Ephemeral elliptic curve point
Ephemeral private key (of A)	q_A such that $Q_A=q_A\ G$	Integer in range [1,n-1]

Remarks (key management for the WPAN security architecture)

1. The key agreement protocol does not provide for evidence as to the intended usage of the shared secret key. In our context, the shared secret key *K* is a link key between Parties A and B, which serves as to protect the authenticity and confidentiality of key transport between A and B. Moreover, at any given instance of time, there is at most one link key between any pair of parties. Hence, the key usage is implied by the operational usage of the key agreement protocol and does not need to be explicitly bound to the key itself.
2. Key transport in NTRU's security architecture is always of one of the following types: broadcast key. Moreover, the key is always generated by the current PNC. Hence, group

membership, key usage attributes, key-life cycle attributes, associated cryptographic algorithms, and key identifiers are all implied by the (limited) security architecture.

3. Data protection in NTRU's security architecture is always of the following type: encryption + integrity check hereover.
4. Command protection in NTRU's security architecture is always of the following type: integrity check value hereover.