Wireless Identity Module

Candidate Version 1.2 – 22 Mar 2005

Open Mobile Alliance

OMA-WAP-WIM-V1_2-20050322-C

Continues the Technical Activities
Originated in the WAP Forum

Contents

1. SCOPE 7

2. REFERENCES 8

2.1 NORMATIVE REFERENCES 8
2.2 INFORMATIVE REFERENCES 9

3. DEFINITIONS AND ABBREVIATIONS 10

3.1 DEFINITIONS 10
3.2 ABBREVIATIONS 11
3.3 DOCUMENT CONVENTIONS 12

4. ARCHITECTURAL OVERVIEW 13

5. WAP SECURITY OPERATIONS 14

5.1 ON-BOARD KEY GENERATION AND ENROLMENT 14
5.1.1 Generate Asymmetric Key pair 14
5.1.2 Generate Assurance Key 14
5.2 WTLS AND TLS OPERATIONS 14
5.3 WAP APPLICATION SECURITY OPERATIONS 15
5.3.1 Unwrapping a Key 15
5.3.2 Digital Signature 16
5.3.3 Signature Verification 16

6. SERVICE INTERFACE DEFINITION 17

6.1 NOTATIONS USED 17
6.1.1 Definition of Service Primitives and Parameters 17
6.1.2 Primitive Types 17
6.1.3 Service Parameter Tables 17
6.2 DESCRIPTION OF PRIMITIVES 18
6.2.1 Device Control Primitives 18
6.2.1.1 WIM-OpenService 18
6.2.1.2 WIM-CloseService 18
6.2.2 Verification Related Primitives 18
6.2.2.1 WIM-PerformVerification 19
6.2.2.2 WIM-DisableVerificationRequirement 19
6.2.2.3 WIM-EnableVerificationRequirement 19
6.2.2.4 WIM-ChangeReferenceData 20
6.2.2.5 WIM-UnblockReferenceData 20
6.2.3 Data Access Primitives 21
6.2.3.1 WIM-OpenFile 21
6.2.3.2 WIM-CloseFile 21
6.2.3.3 WIM-ReadBinary 22
6.2.3.4 WIM-UpdateBinary 22
6.2.4 Cryptography Primitives 23
6.2.4.1 WIM-ComputeDigitalSignature 23
6.2.4.2 WIM-VerifySignature 23
6.2.4.3 WIM-GetRandom 23
6.2.4.4 WIM-KeyTransport 24
6.2.4.5 WIM-KeyAgreement 24
6.2.4.6 WIM-DeriveMasterSecret 24
6.2.4.7 WIM-PHash 25
6.2.4.8 WIM-Decipher 25
6.2.4.9 WIM-Generate Asymmetric Key Pair 26
6.2.4.10 WIM-GenerateKeyAssurance 26
6.2.5 Exceptions 27
6.2.5.1 WIM-Exception 27

7. WIM OPERATIONS IN WTLS 28

7.1 RSA HANDSHAKE 28

7.2 ECDH_ECDSA HANDSHAKE 33
7.3 ABBREVIATED HANDSHAKE 35
7.4 OPTIMISED ECDH_ECDSA HANDSHAKE 37
8. WIM OPERATIONS IN TLS 38
8.1 RSA HANDSHAKE 38
8.2 ABBREVIATED HANDSHAKE 41
9. INFORMATION FORMAT 43
9.1 CONTENTS OF THE FILES 43
9.2 WTLS BITMASK TYPE 43
9.3 ISO OBJECT IDENTIFIERS 44
9.4 PKCS#15 APPLICATION DIRECTORY CONTENTS 44
9.4.1 EF(ODF) 44
9.4.2 Private Key Directory Files (PrKDFs) 45
9.4.3 Public Key Directory Files (PuKDFs) 46
9.4.4 Certificate Directory Files (CDFs) 46
9.4.5 Data Object Directory Files (DODFs) 47
9.4.6 Authentication Object Directory Files (AODFs) 47
9.4.7 EF(TokenInfo) 48
9.4.8 EF(UnusedSpace) 49
9.4.9 Other elementary files in the PKCS#15 directory 49
9.4.10 'Peers-wtls' Data Object 49
9.4.11 'Sessions-wtls' Data Object 50
9.4.12 'Peers-tls' Data Object 51
9.4.13 'Sessions-tls' Data Object 51
9.5 AN EXAMPLE WIM LAYOUT 52
10. SECURITY ENVIRONMENTS 53
10.1 SECURITY ENVIRONMENT DEFINITION 53
10.2 WTLS SECURITY ENVIRONMENTS 55
10.2.1 WTLS_RSA Security Environment 55
10.2.1.1 DST 56
10.2.1.2 CT 57
10.2.1.3 CCT 57
10.2.2 WTLS_ECDH SECURITY ENVIRONMENT 58
10.2.2.1 DST 59
10.2.2.2 CT 59
10.2.2.3 CCT 60
10.3 TLS SECURITY ENVIRONMENTS 60
10.3.1 TLS_RSA Security Environment 60
10.3.1.1 DST 61
10.3.1.2 CT 62
10.3.1.3 CCT 62
10.4 GENERIC SECURITY ENVIRONMENTS 64
10.4.1 WIM_GENERIC_RSA Security Environment 64
10.4.1.1 DST 64
10.4.1.2 CT 65
10.4.2 WIM_GENERIC_ECC Security Environment 66
11. SMART CARD IMPLEMENTATION 67
11.1 PHYSICAL CHARACTERISTICS 67
11.2 ELECTRONIC SIGNALS AND TRANSMISSION PROTOCOLS 67
11.2.1 Answer to Reset 67
11.2.1.1 Protocol 67
11.2.1.2 Transfer Rate 67
11.2.1.3 Supply Voltage 67
11.2.1.4 Logical Channels 67
11.2.1.5 Clock Stop Mode 68
11.2.2 SIM/WIM implementation 68
11.2.3 WIM Only or WIM with Other Applications 68

11.3 DESCRIPTION OF CARD COMMANDS 68
11.3.1 Mapping Service Primitives to Card Commands 71
11.3.2 Managing Logical Channel 75
11.3.2.1 MANAGE CHANNEL Open 75
11.3.2.2 MANAGE CHANNEL Close 76
11.3.3 Application selection 77
11.3.3.1 SELECT Application, Direct Method 78
11.3.4 Verification Related Operations 79
11.3.4.1 VERIFY 79
11.3.4.2 DISABLE VERIFICATION REQUIREMENT 80
11.3.4.3 ENABLE VERIFICATION REQUIREMENT 81
11.3.4.4 CHANGE REFERENCE DATA 82
11.3.4.5 RESET RETRY COUNTER 82
11.3.5 Operations Related to Data Storage 83
11.3.5.1 SELECT FILE 83
11.3.5.2 READ BINARY 84
11.3.5.3 UPDATE BINARY 84
11.3.6 Cryptographic Operations 85
11.3.6.1 MANAGE SECURITY ENVIRONMENT 86
11.3.6.2 MSE - RESTORE 86
11.3.6.3 MSE - SET 87
11.3.6.4 PERFORM SECURITY OPERATION 88
11.3.6.5 PSO - ENCIPHER, Key Transport 89
11.3.6.6 PSO - ENCIPHER, Key Agreement 89
11.3.6.7 PSO - DECIPHER, Application Level 91
11.3.6.8 PSO - COMPUTE DIGITAL SIGNATURE 92
11.3.6.9 PSO - VERIFY DIGITAL SIGNATURE 93
11.3.6.10 PSO - COMPUTE CRYPTOGRAPHIC CHECKSUM 94
11.3.6.11 MSE - DERIVE KEY 95
11.3.6.12 ASK RANDOM 96
11.3.6.13 GENERATE PUBLIC KEY PAIR 97
11.3.6.14 GENERATE KEY ASSURANCE 100
11.3.6.15 Other Commands 104
11.3.6.16 GET RESPONSE 104
11.3.7 Status Words 105
11.3.7.1 Status Words for the WIM Native Mode 105
11.3.7.2 Status Words for the WIM SCP Mode 108
11.4 USAGE OF THE COMMANDS 110
11.4.1 Open Logical Channel 110
11.4.2 Select Application 110
11.4.3 Read Configuration 110
11.4.4 Perform WTLS RSA handshake 110
11.4.5 Perform WTLS ECDH_ECDSA Handshake 114
11.4.6 Perform Application Level Signature 115
11.4.7 Perform Application Related Deciphering 116
11.4.8 Perform TLS RSA handshake 117
11.4.9 Generate Asymmetric Key Pair 121
11.4.10 Generate Key Assurance 122
11.5 USAGE OF THE TLS CHECK VALUE 122
12. WIM ELECTRONIC IDENTIFICATION PROFILE OF PKCS#15 123
12.1 PKCS#15 OBJECTS 123
12.1.1 Private Keys 123
12.1.2 Certificates 123
12.1.3 Data Objects 123
12.1.4 Authentication Objects 123
12.1.4.1 Recommended PIN Format 123
12.2 ACCESS CONTROL RULES 124
12.3 ATTRIBUTE FORMATS 125
13. IMPLEMENTATION NOTES 127

13.1 IMPLEMENTING WIM IN A GSM SIM CARD 127
13.2 WIM FOR NETWORKS NOT UTILIZING A SMARTCARD BASED SIM 127
13.3 USING LOGICAL CHANNELS 127
13.4 SAVING CERTIFICATES 128
13.5 USAGE OF PINS 128
13.6 USING THE WIM FOR NON-WAP APPLICATIONS 130
13.6.1 Signing 130
13.6.2 Private Key Decryption 130
13.6.3 Certificate Storage 130
13.7 SERVER RSA PUBLIC KEY CONSTRAINTS WHEN USING T=0 131
14. WIM STATIC CONFORMANCE REQUIREMENT 133
14.1 WIM OPTIONS 133
14.1.1 General WIM Options 133
14.1.2 WIM ICC Options 136
14.2 ME OPTIONS 138
14.2.1 General ME Options 138
14.2.2 ME Use of WIM ICC 141
APPENDIX A. CHANGE HISTORY (INFORMATIVE) 143
A.1 APPROVED VERSION HISTORY 143
A.2 DRAFT/CANDIDATE VERSION 1.2 HISTORY 143

1. Scope

WAP [WAPARCH] security functionality includes the Wireless Transport Layer Security [WAPWTLS], the Transport Layer Security [TLS10] and application level security, accessible using the Wireless Markup Language Script [WMLScript] and the Crypto Object [ESMPCrypto] for ECMAScript Mobile Profile [ESMP]. For optimum security, some parts of the security functionality need to be performed by a tamper-resistant device, so that an attacker cannot retrieve sensitive data. Such data is especially the permanent private keys used in the WTLS handshake with client authentication, and for making application level electronic signatures (such as confirming an application level transaction). In WTLS, also the master secrets, protecting secure sessions, are relatively long living – which could be several days. This is in order to avoid frequent full handshakes, which are relatively heavy both computationally and due to large data transfer. Master secrets are used as a source of entropy, to calculate MAC keys and message encryption keys, which are used to secure a limited number of messages, depending on usage of WTLS.

The WAP Identity Module (WIM) is used in performing WTLS, TLS and application level security functions, and especially, to store and process information needed for user identification and authentication. The functionality presented here is based on the requirement that sensitive data, especially keys, can be stored in the WIM, and all operations where these keys are involved can be performed in the WIM.

An example of a WIM implementation is a smart card. In the phone, it can be the Subscriber Identity Module (SIM) card or an external smart card. The way, which a phone and a smart card interact, is specified as a command-response protocol, using Application Protocol Data Units (APDU) specific to this application. This specification is based on ISO7816 series of standards on smart cards and the related GSM specifications [GSM11.11], where applicable.

This specification concentrates on defining an interface between the part of a WAP client device that is not considered tamper-resistant, and a tamper-resistant component, the WIM.

A basic requirement for WIM implementation is that it is tamper-resistant. This means that certain physical hardware protection is used, which makes it unfeasible to extract or modify information in the module (volatile, non-volatile memory and other parts). Technology used in smart cards is examples of this kind of protection. Regular mobile phones and PDAs cannot be considered tamper-resistant. For these devices, e.g. extracting information from the module may be difficult but still feasible with a proper equipment.

This specification does not define exact requirements for tamper-resistance. Businesses can enforce certain requirements and policies using PKI based mechanisms. Applications should only accept certificates signed by Certification Authorities that are known to fulfil the requirements and policies.

PKI functionality (including WTLS and TLS client authentication with private keys, and WMLScript and ECMAScript digital signatures) can be implemented in pure software in normal PDAs or phones, using password protection, encryption etc. However, such implementations cannot be considered as WIM implementations, and are out of scope of this specification. At the same time, service interfaces defined in this specification may be useful for designing internal software interfaces for these implementations.

2. References

2.1 Normative References

[IOPPROC] "OMA Interoperability Policy and Process", Version 1.1, Open Mobile Alliance™, OMA-IOP-Process-V1_1, URL:http://www.openmobilealliance.org/

[ASN1] ITU-T Recommendation X.680 (1997) | ISO/IEC 8824-1:1998, Information technology – Abstract Syntax Notation One (ASN.1): Specification of basic notation.

[DER] ITU-T Recommendation X.690 (1997) | ISO/IEC 8825-1:1998, Information technology – ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER).

[GSM11.11] Digital cellular telecommunications systems (Phase2+); Specification of the Subscriber Identity Module – Mobile Equipment (SIM – ME) interface (GSM 11.11 version 5.4.0).

[GSM11.12] Digital cellular telecommunications systems (Phase2); Specification of the 3 Volt Subscriber Identity Module – Mobile Equipment (SIM – ME) interface (GSM 11.12 version 4.3.0).

[GSM11.18] Digital cellular telecommunications systems (Phase2+); Specification of the 1.8 Volt Subscriber Identity Module – Mobile Equipment (SIM – ME) interface.

[ISO 7816-1] Identification Cards – Integrated Circuit(s) Cards with Contacts – Part 1: Physical characteristics.

[ISO 7816-2] Identification Cards – Integrated Circuit(s) Cards with Contacts – Part 2: Dimensions and location of the contacts.

[ISO 7816-3] Identification Cards – Integrated Circuit(s) Cards with Contacts – Part 3: Electronic signals and transmission protocols.

[ISO 7816-4] Information Technology – Identification Cards – Integrated Circuit(s) Cards with Contacts – Part 4: Interindustry commands for interchange.

[ISO 7816-5] Identification Cards – Integrated Circuit(s) Cards with Contacts – Part 5: Numbering system and registration procedure for application identifiers.

[ISO 7816-6] Identification Cards – Integrated Circuit(s) Cards with Contacts – Part 6: Interindustry data elements.

[ISO 7816-8] Identification Cards – Integrated Circuit(s) Cards with Contacts – Part 8: Security related interindustry commands.

[P1363a] "Standard Specifications For Public Key Cryptography: Additional Techniques", IEEE P1363a / D7 (Draft Version 7). URL: http://grouper.ieee.org/groups/1363/

[RFC2313] PKCS #1: RSA Encryption, version 1.5, IETF RFC 2313, B Kaliski, March 1998. URL: ftp://ftp.isi.edu/in-notes/rfc2313.txt.

[PKCS15] PKCS #15 v1.1: Cryptographic Token Information Syntax Standard", RSA Laboratories, June 6, 2000. URL: ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-15/pkcs-15v1_1.pdf

[PKCS15TC1] PKCS #15 v1.1 Technical Corrigendum 1, RSA Laboratories, October 24, 2000. URL: ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-15/pkcs-15v1_1tc1.pdf

[RFC2315] PKCS #7: Cryptographic Message Syntax, version 1.5, IETF RFC 2313, B Kaliski, March 1998. URL: ftp://ftp.isi.edu/in-notes/rfc2313.txt.

[RFC2119] "Key words for use in RFCs to Indicate Requirement Levels", Bradner, S., March 1997. URL: http://www.ietf.org/rfc/rfc2119.txt

[TLS10] "The TLS Protocol, Version 1.0," rfc 2246, T. Dierks, C. Allen, January 1999. URL: http://www.ietf.org/rfc/rfc2246.txt

[TS102.221] Smart Cards; UICC-Terminal interface; Physical and logical characteristics (ETSI TS 102 221, version 4.3.0)

[TLSProfile] "TLS Profile and Tunneling Specification ", WAP-219-TLS, WAP Forum. URL: http://www.openmobilealliance.org

[WAPWCMP] "Wireless Control Message Protocol Specification", WAP-202-WCMP. URL: http://www.openmobilealliance.org

[WAPWTLS] "Wireless Transport Layer Security Specification", WAP-261-WTLS, WAP Forum., URL: http://www.openmobilealliance.org

[X9.62] "The Elliptic Curve Digital Signature Algorithm (ECDSA)", ANSI X9.62 – 1998 (Approved: January 7, 1999).

[RFC2104] HMAC: Keyed-Hashing for Message Authentication, H. Krawczyk, M. Bellare, R. Canetti, RFC2104, http://www.ietf.org/rfc/rfc2104.txt?number=2104

[NIST SP 800-38A 2001 ED] NIST SP 800-38A 2001 ED, Recommendation for Block Cipher Modes of Operation, December 2001. Available from http://csrc.nist.gov/publications/nistpubs/800-38a/sp800-38a.pdf

2.2 Informative References

[ESMPCrypto] "ECMA Script Crypto Library", Open Mobile Alliance™ , OMA-WAP-ECMACR-V1_1, URL:http://www.openmobilealliance.org/

[ESMP] "ECMAScript Mobile Profile", OMA-WAP-ESMP-v1_0, Open Mobile Alliance ™. URL: http://www.openmobilealliance.org

[WAPARCH] "WAP Architecture Specification", WAP-210-WAPArch, WAP Forum. URL: http://www.openmobilealliance.org

[WMLScript] "WMLScript Language Specification", WAP-193-WMLScript, WAP Forum. URL: http://www.openmobilealliance.org

[S/MIME] "S/MIME Version 2 Message Specification", Dusse, S., Hoffman, P., Ramsdell, B., Lundblade, L., Repka, L., March 1998. URL: ftp://ftp.isi.edu/in-notes/rfc2311.txt

[SSL] "The SSL 3.0 Protocol", Netscape Communications Corp., November 1996.

3. Definitions And Abbreviations

3.1 Definitions

For the purposes of this specification the following definitions apply.

Integrated Circuit Card

See Smart card

Smart card

A device with an embedded microprocessor chip. A smart card is used for storing data and performing typically security related (cryptographic) operations. In WAP context, a smart card may be the GSM Subscriber Identity Module (SIM) or a card used in a secondary card reader of a WAP phone.

Wireless Identity Module

A tamper-resistant device which is used in performing WTLS, TLS and application level security functions, and especially, to store and process information needed for user identification and authentication.

3.2 Abbreviations

For the purposes of this specification the following abbreviations apply.

AID	Application Identifier
AODF	Authentication Object Directory File
APDU	Application Protocol Data Unit
API	Application Programming Interface
ASN	Abstract Syntax Notation
AT	Authentication Template
ATR	Answer To Reset
BCD	Binary Coded Decimal
CA	Certification Authority
CCT	Cryptographic Checksum Template
CDF	Certificate Directory File
CLA	CLAss
CRDO	Control Reference Data Object
CRT	Control Reference Template
CT	Confidentiality Template
DE	Data Element
DER	Distinguished Encoding Rules
DF	Dedicated File
DH	Diffie-Hellman
DIR	Directory file
DO	Data Object
DODF	Data Object Directory File
DODF-wim	DODF containing WTLS and TLS session related data as defined in this specification
DS	Digital Signature
DSI	Digital Signature Input
DST	Digital Signature Template
EC	Elliptic Curve
ECC	Elliptic Curve Cryptography
ECIES	Elliptic Curve Integrated Encryption Scheme
ECDH	Elliptic Curve Diffie-Hellman
ECDSA	Elliptic Curve Digital Signature Algorithm
EF	Elementary File
GKA	Generate Key Assuarance
GSM	Global System for Mobile Communication
HT	Hash Template
IC	Integrated Circuit
ICC	Integrated Circuit(s) Card
ID	Identifier
IDO	Inter-industry Object
INS	Instruction Byte
IV	Initialisation Vector
MAC	Message Authentication Code
ME	Mobile Equipment
MF	Master File
MSE	Manage Security Environment command
OBKG	On-Board Key Generation
ODF	Object Directory File
OID	Object Identifier
OMT	Object Modeling Technique
OSI	Open System Interconnection
PDU	Protocol Data Unit

PIN	Personal Identification Number
PIX	Proprietary Application Identifier Extension
PK	Public Key
PPS	Protocol Parameter Selection
PRF	Pseudo-Random Function
PrKDF	Private Key Directory File
PSO	Perform Security Operation command
PuKDF	Public Key Directory File
RID	Registered Application Provider Identifier
RFU	Reserved for Future Use
RSA	RSA (Rivest, Shamir, Adleman) public key algorithm
SAP	Service Access Point
SCP	Smart Card Platform
SDU	Service Data Unit
SE	Security Environment
SHA-1	Secure Hash Algorithm
SIM	Subscriber Identity Module
SK	Secret Key
SMS	Short Message Service
SSL	Secure Sockets Layer
SW1/SW2	Status Word 1 / Status Word 2
TLS	Transport Layer Security
TLV	Tag-Length-Value
TPDU	Transmission Protocol Data Unit
UICC	Universal ICC
WAP	Wireless Application Protocol
WIM	Wireless Identity Module
WML	Wireless Markup Language
WMLScript	Wireless Markup Language Script
WDP	Wireless Datagram Protocol
WTLS	Wireless Transport Layer Security

3.3 Document Conventions

This specification uses the keywords "MUST", "MUST NOT", "SHOULD", "SHOULD NOT", "MAY" etc., as specified in [RFC2119].

All sections and appendixes, except "Scope" (Section 1) and "Architectural Overview" (Section 4) are normative, unless they are explicitly indicated to be informative.

4. Architectural Overview

A model of layering the protocols in WAP is illustrated Figure 1: Wireless Application Protocol Reference Model. The layering of WAP protocols and their functions is similar to that of the ISO OSI Reference Model [ISO7498] for upper layers. Layer Management Entities handle protocol initialisation, configuration, and error conditions (such as loss of connectivity due to the mobile terminal roaming out of coverage) that are not handled by the protocol itself.

The WIM is a tamper-resistant device. It is used to enhance security of the implementation of the Security Layer and certain functions of the Application Layer. The WIM-SAP is defined in order to describe the WIM functionality that is common to all kind of WIM implementations.

The information structure is based on [PKCS15] and [PKCS15TC1] which enables a flexible information format on a cryptographic token. It uses on object model that makes it possible to access keys, certificates, authentication objects and proprietary data objects in a simple device (with simple read/write, and access control features).

The WIM functionality can be implemented on a smart card. A smart card implementation is based on ISO7816 series of standards. The WIM is defined as an independent smart card application, which makes it possible to implement it as a WIM-only card or as a part of multi-application card containing other card applications, like the GSM SIM. The WIM application is designed so that it is possible to implement it with current smart card technology.

Use of generic cryptographic features with standard interfaces like ISO7816 and PKCS#15 can make it interesting to use the WIM also for non-WAP applications, like SSL, S/MIME etc.

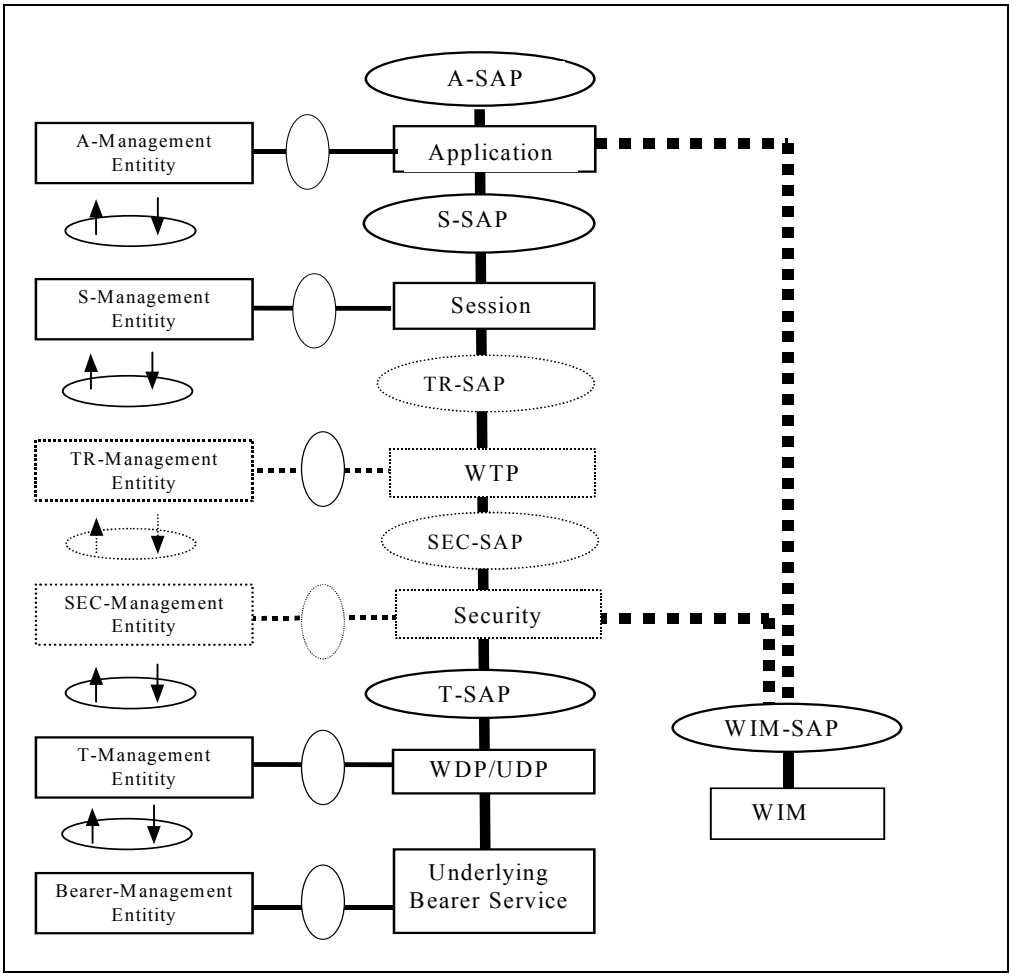


Figure 1: Wireless Application Protocol Reference Model

5. WAP Security Operations

This chapter presents how implementation of WAP security functionality may be supported in the WIM. The specific implementation may expect additional functionality and services being present. Those services are described in relevant standards.

5.1 On-board key generation and enrolment

On-board key generation and enrolment functionalities enable the generation and registration of keys into a PKI. Unlike the WAP model where key pairs were provisioned in a factory and distributed manually, these functionalities allow for new key pairs to be generated in the field.

In some scenarios the ability to include assurances that the key was generated in a secure manner is required. These assurances may specify that a key was generated per a specific policy and on some approved Security Element (SE), such as a WIM.

5.1.1 Generate Asymmetric Key pair

This functionality may be used by authorised application to generate, post issuance, asymmetric key pair. The private & public elements of the asymmetric key pair are generated on board. The private element never leaves the WIM whereas the public element is kept available for key registration to the PKI.

Application may make use of that functionality using ECMAScript to generate asymmetric key pairs for any type of operations i.e. WTLS/TLS and application security operations.

5.1.2 Generate Assurance Key

This functionality may be used by authorised application during key registration process, to include assurances that the key was generated under a specific security policy on-board the WIM (i.e. an approved security element).

An assurance signature or an assurance message authentication code (MAC) may be then included in the registration request. The information that is signed or MACed includes the public key and may include additional information indicating the kind of assertion that is being made. This assurance information is then verified by the PKI to prove that the key was generated under a specific security policy on-board the WIM. This mechanism currently can indicate not only if the key was generated on-board, but may also be used to indicate if a key was injected into the WIM. In addition, the registration messages generated by this functionality shall be based on industry standard PKCS#10 message formats, ensuring tight integration with the existing CA and PKI infrastructure.

5.2 WTLS and TLS Operations

For WTLS and TLS, the WIM is used for the following purposes

- performing cryptographic operations during the handshake, especially those that are used for client authentication
- securing long-living WTLS and TLS secure sessions

The WIM is used to protect permanent, typically certified, private keys. The WIM stores these keys and performs operations using these keys. The operations are

- signing operation (eg, ECDSA or RSA) for client authentication when needed for the selected handshake scheme
- key exchange operation using a fixed client key (eg, ECDH key, in ECDH_ECDSA handshake)

So, the private keys never leave the WIM.

It is essential to have good quality unpredictable random numbers in some handshake schemes (eg, RSA) where a random number is used as a part of the pre-master secret. The WIM is REQUIRED to support random number generation. The ME MAY take advantage of random numbers generated by the WIM.

The WIM may store needed certificates: CA and user certificates. Storage of trusted (root) CA certificates has significance also from security point of view: they must not be altered – but they can be exposed without danger. CA certificates may be stored by WIM issuers, or by a user at a later time. If there are many certificates, there may be a need to store them in the phone. Anyway, they are subject to change. So, the phone should be able to download new certificates over the air and store them itself or save them in the WIM.

From security point of view, there is no requirement of storing user certificates in a tamper resistant place. Storing certificates in the WIM may be useful from point of view of logistics and portability. Note that in WTLS, the server may retrieve a client's certificate from its own sources. Also, it is possible to store a certificate url (instead of the certificate itself) in the WIM.

The WIM maintains information on algorithms that it supports. The phone retrieves that information from the WIM.

Permanent key pairs may be generated in the WIM or stored there as a part of the manufacturing or personalization process.

The WIM is used to protect secure sessions, in addition to private keys. The WIM supports the following functionality

- calculation (ECDH key exchange) or generation (RSA key exchange) of the pre-master secret
- calculation and storage of the master secret for each secure session
- derivation and output of key material (for MAC, encryption keys, IV, finished check), based on the master secret

So, the master secret and the pre-master secret used to calculate that, never leave the WIM.

The phone stores MAC and message encryption keys as long as they are currently needed. For WTLS, these keys have a limited lifetime which may be negotiated during the WTLS handshake – in the extreme case they are used for a single message only. The phone also may delete them from its memory when the user exits from the secure WAP application. These keys can always be derived anew from the master secret if needed.

An attacker who obtains a message encryption key can read as many messages as is agreed in the key refresh configuration (in the extreme case, a single message). An attacker who obtains a MAC key can impersonate the compromised party during as many messages as is agreed in the configuration (in the extreme case, a single message).

WIM implementations are RECOMMENDED to support TLS and REQUIRED to support at least one of TLS and WTLS.

5.3 WAP Application Security Operations

Application level security operations that use the WIM include signing and unwrapping a key. Both these operations use a private key that never leaves the WIM. These operations are meant to be generic in order to serve any applications defined in WAP (eg, using WMLScript or ECMAScript) or outside WAP.

5.3.1 Unwrapping a Key

Unwrapping a key is needed when an application receives a message key enciphered with a public key that corresponds to a private key in the WIM. The ME sends the wrapped key to the WIM. The WIM deciphers it using the private key and returns the unwrapped key. The ME may use the unwrapped key to decipher the attached message.

WIM implementations MAY support and ME implementations MAY use key unwrapping, and if they do, either with RSA or ECC MUST be supported. In case of ECC ECIES SHOULD be supported.

5.3.2 Digital Signature

Digital signing may be used for authentication or non-repudiation purposes (eg, sign a document or confirm a transaction). For non-repudiation, a separate key is usually used, and the user is requested to enter authentication information (PIN) for every signature made. Note that in order to support non-repudiation, the signature key must never leave a tamper-resistant device.

For signing some data, the ME calculates a hash of the data, formats it according to the requirements of the application and sends the formatted hash to the WIM. The WIM calculates the digital signature using the private key, and returns the digital signature.

WIM implementations MUST support signing either with RSA or ECC. In case of ECC, ECDSA MUST be supported.

ME implementations MAY use WIM signing, and if they do, either RSA or ECC MUST be supported. In case of ECC, ECDSA MUST be supported.

5.3.3 Signature Verification

Signature verification by WIM may be used in cases where an application needs verification capability (e.g. certificate or end entity signature verification) but the verification algorithm is not present in the ME, or the verification algorithm implementation is more efficient in the WIM. It is not anticipated that this would be practical in all cases, e.g., due to the fact that WIM cryptographic capabilities are likely to be targeted for handling keys suitable for end entities and may not be sufficient for handling keys used by CAs.

In verification, the ME sends the public key, the signature and the hash of the data to the WIM. The WIM returns the status of verification.

WIM implementations MAY support and ME implementations MAY use signature verification.

6. Service Interface Definition

The service definition for WIM covers simple storage functionality and security functionality used for WTLS, TLS and application security. The interface is described using service primitives.

6.1 Notations Used

6.1.1 Definition of Service Primitives and Parameters

Communication between the WIM and the entities using it is accomplished by means of service primitives. Service primitives represent, in an abstract way, the logical exchange of information and control between the WIM and other entities.

Service primitives consist of commands and their respective responses associated with the services requested of another entity. The general syntax of a primitive is:

X-Service.type (Parameters)

where X designates the entity providing the service. For this specification X is “WIM” for the WIM.

Service primitives are not the same as an application-programming interface (API) and are not meant to imply any specific method of implementing an API. Service primitives are an abstract means of illustrating the services provided by the entity. The mapping of these concepts to a real API and the semantics associated with a real API are an implementation issue and are beyond the scope of this specification.

6.1.2 Primitive Types

The primitives types defined in this specification are:

Type	Abbreviation	Description
request	req	Used when a user of the module is requesting a service from the module
indication	ind	The module providing a service uses this primitive type to notify the user of a module about an event
confirm	cnf	The module providing the requested service uses the confirm primitive type to report that the activity has been completed successfully

6.1.3 Service Parameter Tables

The service primitives are defined using tables indicating which parameters are possible and how they are used with the different primitive types. For example, a simple confirmed primitive might be defined using the following:

Primitive / Parameter	S-primitive	
	req	*cnf*
Parameter 1	M	
Parameter 2		O

If some primitive type is not possible, the column for it will be omitted. The entries used in the primitive type columns are defined in the following table:

M	Presence of the parameter is mandatory – it MUST be present
C	Presence of the parameter is conditional depending on values of other parameters
O	Presence of the parameter is a user option – it MAY be omitted
P	Presence of the parameter is a service provider option – an implementation MAY not provide it
	The parameter is absent
*	Presence of the parameter is determined by the lower layer protocol
(=)	The value of the parameter is identical to the value of the corresponding parameter of the preceding service primitive

6.2 Description of Primitives

6.2.1 Device Control Primitives

6.2.1.1 WIM-OpenService

This primitive is used in order to open the WIM, before using any other primitives. The primitive may imply things like selecting a proper service application. The implementation may contain obtaining a service handle to be used in subsequent operations.

Note that getting information on existence of WIMs or selection of a certain WIM is out of scope of this interface definition.

Primitive	WIM-OpenService	
Parameter	***req***	***cnf***
-	-	-

6.2.1.2 WIM-CloseService

This primitive is used after using other primitives.

Primitive	WIM-CloseService	
Parameter	***req***	***cnf***
-	-	-

6.2.2 Verification Related Primitives

These primitives are used to verify that the user of the WIM is a legitimate user. These primitives need to be used to access information or perform operations that need certain authorization.

A single user model is used. The verification is based on comparison of verification data presented by the user, to reference data stored in the WIM.

Typically, the verification status remains in power until the WIM service is closed. However, for performing digital signatures for non-repudiation purposes, verification should be done each time the signature is used.

6.2.2.1 WIM-PerformVerification

This primitive is used to compare verification data (eg, PIN entered by the user) with the reference data in the WIM (eg, correct PIN stored in the WIM).

This primitive is used to get access to private objects in the WIM.

Primitive / Parameter	WIM-PerformVerification	
	req	*cnf*
Reference Data ID	M	
Verification Data	M	

Reference Data ID indicates which reference data (eg, file path, PIN reference) should be used for comparison.

Verification Data is the data that the WIM should compare with the reference data.

6.2.2.2 WIM-DisableVerificationRequirement

This primitive is used to disable the verification mechanism.

Primitive / Parameter	WIM-DisableVerificationRequirement	
	req	*Cnf*
Reference Data ID	M	
Verification Data	M	

Reference Data ID indicates which reference data (eg, file path, PIN reference) should be used for comparison.

Verification Data is the data that the WIM should compare with the reference data.

6.2.2.3 WIM-EnableVerificationRequirement

This primitive is used to enable the verification mechanism.

Primitive / Parameter	WIM-EnableVerificationRequirement	
	req	*cnf*
Reference Data ID	M	
Verification Data	M	

Reference Data ID indicates which reference data (eg, file path, PIN reference) should be used for comparison.

Verification Data is the data that the WIM should compare with the reference data.

6.2.2.4 WIM-ChangeReferenceData

This primitive is used to change the reference data in the WIM.

Primitive	WIM-ChangeReferenceData	
Parameter	***Req***	***cnf***
Reference Data ID	M	
Verification Data	M	
New Reference Data	M	

Reference Data ID indicates which reference data (eg, file path, PIN reference) should be used for comparison, and changed.

Verification Data is the data that the WIM should compare with the reference data.

New Reference Data is the data that the WIM should replace the existing reference data with.

6.2.2.5 WIM-UnblockReferenceData

This primitive is used to unblock the reference data (reset the retry counter) and set a new reference data.

Primitive	WIM-UnblockReferenceData	
Parameter	***Req***	***cnf***
Reference Data ID	M	
Unblock Data	M	
New Reference Data	M	

Reference Data ID indicates which reference data (eg, file path, PIN reference) should be used.

Unblock Data is the data that is required by the WIM in order to unblock the reference data (reset the retry counter).

New Reference Data is the data that the WIM should replace the existing reference data with.

6.2.3 Data Access Primitives

Organization of data in the WIM is based on files, which are referenced using a file path.

The following structures of files are defined

- transparent (binary) – the file is seen as a sequence of octets
- formatted (record based) – the file is seen as a sequence of individually identifiable records

In a formatted file, records may be organized as a sequence (linear structure) or as a ring (cyclic structure). Formatted files are not used in this version of the specification.

6.2.3.1 WIM-OpenFile

This primitive is used to open a file in the WIM, to be accessed.

Primitive / Parameter	WIM-OpenFile	
	req	*Cnf*
Path	M	
Status		O

Path indicates the file to be opened.

Status contains information on the file opened, such as file size.

6.2.3.2 WIM-CloseFile

This primitive is used to close a file in the WIM. (For some type of devices, this primitive does not apply.)

Primitive / Parameter	WIM-CloseFile	
	req	*Cnf*
-	-	-

6.2.3.3 WIM-ReadBinary

This primitive is used to read (a portion of) a file.

Primitive	WIM-ReadBinary	
Parameter	***req***	***Cnf***
Offset	M	
Length	M	
User Data		M

Offset indicates the offset from the beginning of the file.

Length indicates the amount of data to be read.

User Data is the requested (portion of the) file.

6.2.3.4 WIM-UpdateBinary

This primitive is used to update data in a file.

Primitive	WIM-UpdateBinary	
Parameter	***req***	***Cnf***
Offset	M	
User Data	M	

Offset indicates the offset from the beginning of the file.

User Data is the data that should be written.

6.2.4 Cryptography Primitives

6.2.4.1 WIM-ComputeDigitalSignature

This primitive is used to compute a digital signature for application layer security or during the WTLS and TLS handshakes.

Primitive	WIM-ComputeDigitalSignature	
Parameter	***req***	***cnf***
Private Key ID	M	
User Data	M	
Signature		M

Private Key ID identifies the key used in the signature.

User Data is the data that is to be signed.

Signature is the result of the signature computation.

6.2.4.2 WIM-VerifySignature

This primitive is used to verify a signature received from a peer by using the public key that corresponds to the private key used to generate the signature (eg, CA public key or peer public key).

WIM takes the signature and the corresponding digest as the input and verifies that the signature is valid by using the public key.

Primitive	WIM-VerifySignature	
Parameter	***req***	***cnf***
Public Key	M	
Digest	M	
Signature	M	

Public Key is the public key that corresponds to the private key that generated the *Signature*.

Digest is the hash of the original data that is signed. This can be a 20 byte SHA-1 hash or a formatted hash (ie, containing a header required by an application).

Signature is the digitally signed digest by using the private key (e.g, CA private key or peer private key).

6.2.4.3 WIM-GetRandom

This primitive is used to get a random number of needed length from the WIM. The random number generated needs to be unpredictable and of good quality.

Primitive	WIM-GetRandom	
Parameter	***req***	***cnf***
Length	M	
Random		M

Length indicates the length of the required random number.

Random is the random number returned.

6.2.4.4 WIM-KeyTransport

This primitive is used to transport a shared key to another peer, using public key encryption.

The WIM generates a value then encrypts the value with server public key and returns the result to the ME. The WIM builds the pre-master secret and keeps it in its memory for the next operation.

The value consists of the client version number and random byte. The size of these fields and the pre-master secret definition depend on the transport layer security protocol:

- WTLS: 1 byte client version number and 19 random bytes; the pre-master secret is the resulting 20 byte value appended with the server public key
- TLS: 2 byte client version number and 46 random bytes; the pre-master secret is the resulting 48 byte value

Primitive / Parameter	WIM-KeyTransport	
	req	*cnf*
Public Key	M	
Additional Data	M	
Transported Key		M

Public Key is the peer's public key.

Additional Data is additional data to be added in the transported key (security protocol version number).

Transported Key is the shared key in a form that should be sent to the other party, to be unwrapped by it.

6.2.4.5 WIM-KeyAgreement

This primitive is used to negotiate a secret, using a Diffie-Hellman scheme.

The WIM performs ECDH calculation based on received server public key and private key contained in the WIM. The negotiated key (pre-master secret) is kept in the WIM memory for the next operation (WIM-DeriveMasterSecret).

Primitive / Parameter	WIM-KeyAgreement	
	req	*cnf*
Private Key ID	M	
Public Key	M	

Private Key ID identifies the key used in the Diffie-Hellman calculation.

Public Key is the peer's public key used in the Diffie-Hellman calculation.

6.2.4.6 WIM-DeriveMasterSecret

This primitive is used to derive the WTLS or TLS master secret based on a pre-master secret that is a result of a preceding WIM-KeyAgreement or WIM-KeyTransport primitive. The master secret can be derived only from a pre-master secret of the same transport layer security protocol.

Primitive / Parameter	WIM-DeriveMasterSecret	
	req	*cnf*
Input Data	M	
Master Secret ID	M	

Input Data is used as input for key derivation.

Master Secret ID identifies the resulting WTLS or TLS master secret.

6.2.4.7 WIM-PHash

This primitive is used to calculate a block of data (eg, a key block) based on a WTLS or TLS master secret located in the WIM.

Primitive / Parameter	WIM-PHash	
	req	*cnf*
Master Secret ID	M	
Input Data	M	
Length	M	
Block		M

Master Secret ID identifies the WTLS or TLS master secret used as source in the calculation.

Input Data is used as input for calculation.

Length indicates the amount of data to be returned.

Block indicates the calculated block of data.

6.2.4.8 WIM-Decipher

This primitive is used to decipher an enciphered message key, for application security

Primitive / Parameter	WIM-Decipher	
	req	*cnf*
Private Key ID	M	
Enciphered Data	M	
Data		M

Private Key ID identifies the key used in deciphering.

Enciphered Data is the data that is to be decrypted.

Data is the result of the deciphering.

6.2.4.9 WIM-Generate Asymmetric Key Pair

This primitive is used to generate a new asymmetric key pair.

Primitive	WIM-GenerateAsymmetricKeyPair	
Parameter	***req***	***cnf***
Private Key ID	M	
Authentication Data	O	
Encrypted PIN value	O	
Encrypted PIN label	O	
Encrypted private & public key label	O	
User PIN Value	O	
Response Data		M

Private Key ID identifies the key that should be generated.

Authentication *Data* is the data that authenticates the entity that is authorized to invoke this function.

Encrypted PIN value is a new encrypted value for the PIN that protects the key pair to generate[1].

Encrypted PIN label is a new encrypted label for the PIN that protects the key pair to generate[1].

Encrypted private & public key label is a new encrypted label for the key pair to generate[1].

Response Data is either the public key hash of the newly generated key pair or a challenge and a WIM serial number if authentication is required.

6.2.4.10 WIM-GenerateKeyAssurance

This primitive is used to output Assurance information of the key referenced and present in the WIM, along with an Assurance signature.

Primitive	WIM-GenerateKeyAssurance	
Parameter	***req***	***Cnf***
Key Reference	M	
Authentication Data	O	
ResponseData		M

Key Reference identifies the key pair of which an Assurance signature is requested.

Authentication Data may be required to verify that the entity invoking this primitive is authorized to do so.

Response Data is either the requested Assurance information and the Assurance signature or a Challenge and a WIM serial number if authentication is required

[1] The encryption is using 3DES in CBC mode as described in section 15.2 of [NIST SP 800-38A 2001 ED]. The initial chaining value for CBC modes shall be zero.

6.2.5 Exceptions

6.2.5.1 WIM-Exception

This primitive is used to inform about errors, warnings or other events.

Primitive	WIM-Exception
Parameter	***ind***
Error Type	M

Error Type indicates the type of the event (error, warning etc) occurred in the WIM.

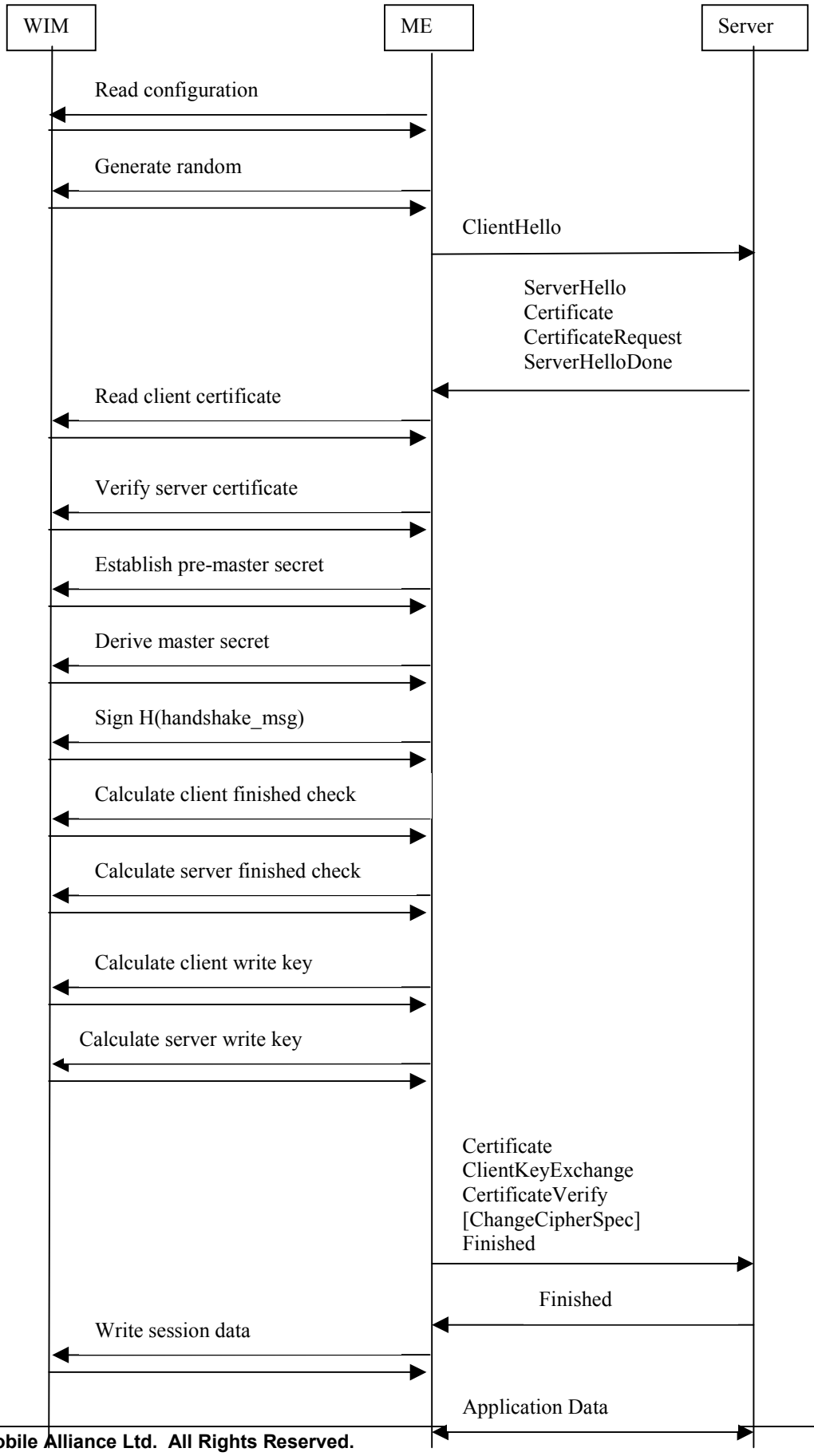
7. WIM Operations in WTLS

This chapter describes messages sent during a WTLS handshake between the WTLS server and client, and between the ME and the WIM. The message flow between the WTLS client and server is described in [WAPWTLS]. The message flow between the ME and the WIM is described at a functional level, using service primitives.

WIM implementations that support WTLS MUST support either RSA or ECDH_ECDSA handshake.

7.1 RSA Handshake

In an RSA handshake, the WIM is used to provide client identity and authentication. This involves retrieving the client public key or certificate from the WIM, and performing the signature operation proving the client's identity. The WIM is also used to generate a high quality random number for the pre-master secret, and to encrypt the pre-master secret with the server public key, derive the master secret and calculate all values based on the master secret.



Read configuration

Before starting the handshake procedure with the server, the ME needs to know which algorithms the WIM supports and information on keys and certificates stored in the WIM. The ME may read the key information from the WIM, in order to send the server the client's identity in a form of a public key hash, in the ClientHello message of the WTLS handshake. The ME may have the configuration and other data in a cache, in order not to read it during every new handshake. For reading the configuration the ME uses data access primitives: WIM-OpenFile, WIM-ReadBinary etc.

Generate random

The ME may use the WIM to generate 12 random bytes, to be used in the ClientHello message. Primitive used: WIM-GetRandom.

Read client certificate

If the server is not able to obtain the client certificate from its own sources, based on client identity received in ClientHello, the server may request the client to send its certificate. The ME may then read a certificate stored in the WIM, using data access primitives: WIM-OpenFile, WIM-ReadBinary etc.

Verify server certificate

If the WIM supports the WIM-VerifySignature, the ME MAY use this primitive to verify the CA signature of the server certificate. If this is not supported by the WIM, the ME must perform the entire verification. In any case the ME must verify the signed data in the certificate.

Establish pre-master secret

The WIM generates a 20 byte value consisting of the client version number (1 byte) and 19 random bytes. It encrypts the value with the server public key and returns the result to the ME. The pre-master secret is the 20 byte value appended with server public key. The WIM keeps the pre-master secret in its memory for the next operation.

The ME uses the WIM-KeyTransport primitive. The primitive returns the encrypted value, to be sent to the server.

Derive master secret

Using the PRF, the WIM derives a master secret based on the pre-master secret established in the previous operation, and a seed value received from the ME. WTLS PRF using SHA-1 MUST be used. The WIM stores the master secret persistently, to be accessible under a certain session/key id.

The ME uses the WIM-DeriveMasterSecret primitive. The parameter Master Secret ID identifies the resulting master secret. The Input Data parameter has the value "master secret" + ClientHello.random + ServerHello.random

Sign H(handshake_messaged)

The WIM signs a hash of handshake messages transmitted so far between the client and the server. The signature is returned to the ME to be used in the CertificateVerify message.

The ME uses the WIM-ComputeDigitalSignature primitive. The primitive returns the signature.

Calculate client finished check

Using the PRF, the WIM calculates a requested number of bytes based on the master secret, and a seed value received from the ME. The WIM returns the bytes to be used by the ME in the Finished message.

The ME uses the WIM-PHash primitive with the Input Data parameter “client finished” + H(handshake_messages)

The primitive returns a 12 byte block.

Calculate server finished check

As Calculate client finished check, with a different label and H(handshake_messages).

The ME uses the WIM-PHash primitive with the Input Data parameter “server finished” + H(handshake_messages)

The primitive returns a 12 byte block.

Calculate client write key block

Using the PRF, the WIM calculates a requested number of bytes based on the master secret, and a seed value received from the ME. The WIM returns the bytes to be used by the ME as the client write key block.

The ME uses the WIM-PHash primitive with a Input Data parameter

“client expansion” + seq_num + server_random + client_random

The primitive returns as many bytes as requested.

Note that the seq_num at the first creation of the key block is zero. This operation, with a different sequence number, is used each time when a key block is refreshed (or when the same key block is needed again but it was erased from the phone memory).

Calculate server write key block

As Calculate client write key block, with a different seed.

The ME uses the WIM-PHash primitive with a Input Data parameter

“server expansion” + seq_num + server_random + client_random

The primitive returns as many bytes as requested.

Write address and session data

The ME stores all information that is needed to resume the session later on. This covers address and session related data. The master secret is handled internally by the WIM. Primitives used: WIM-OpenFile, WIM-UpdateBinary etc.

Note that the Derive master secret operation must be preceded by Establish pre-master secret operation. All WIM-PHash operations mush be performed after the Derive master secret operation. The client finished check must be calculated before the server finished check. The Write session data operation must be performed only after the server Finished message has been verified. Otherwise, the order of the operations may depend on the implementation of the ME.

In an RSA handshake with client authentication, when using a server key which is not longer than supported by the WIM, and the certificate requested to be used is associated with a private key in the WIM, the ME MUST

- use the WIM for master secret negotiation and storage (establish pre-master secret, derive master secret).
- use the WIM for signing the hash of handshake messages (sign H(handshake_messages))

- use the WIM for calculating needed key material (client finished check, server finished check, client write key block, server write key block)

The ME SHOULD store peer address and session data in the WIM (storing may be not necessary in case the session is not meant to be resumed later).

The ME MUST be able to use the following data (since this data can be cached, reading does not have to occur during every handshake)

- CA certificates for verification of server certificates
- user certificates

7.2 ECDH_ECDSA Handshake

In ECDH_ECDSA handshake, the WIM is used to provide the client's identity and make the ECDH shared key calculation based on a fixed private key on the WIM. The ability to calculate the shared key authenticates the client to the server (and vice versa). The WIM is also used to derive the master secret and calculate all values based on that.

Read configuration
Generate random
Verify server certificate

As in RSA handshake.

Establish pre-master secret

The WIM performs ECDH shared key calculation based on received server public key and private key contained in the WIM. The shared key (pre-master secret) is kept in the WIM memory for the next operation.

The ME uses the WIM-KeyAgreement primitive.

Derive master secret
Calculate client finished check
Calculate server finished check
Calculate client write key block
Calculate server write key block
Write address and session data

As in RSA handshake.

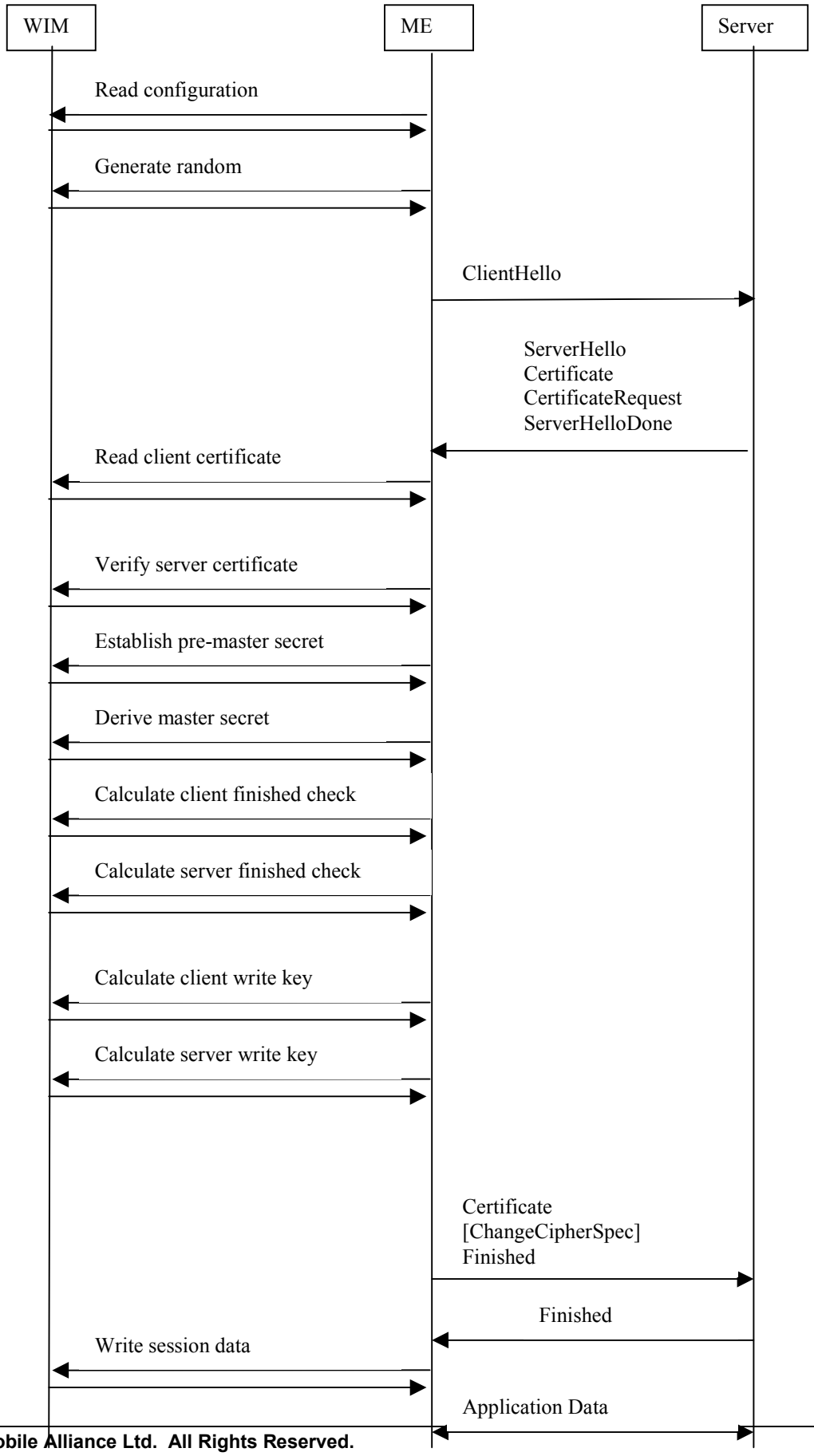
In an ECDH handshake with client authentication, in case the server key is using an elliptic curve supported by the WIM, and the certificate requested to be used is associated with a private key in the WIM, the ME MUST

- use the WIM for master secret negotiation and storage (establish pre-master secret, derive master secret)
- use the WIM for calculating needed key material (client finished check, server finished check, client write key block, server write key block)

The ME SHOULD store peer address and session data in the WIM (storing may be not necessary in case the session is not meant to be resumed later).

The ME MUST be able to use the following data (since this data can be cached, reading does not have to occur during every handshake)

- CA certificates for verification of server certificates
- user certificates

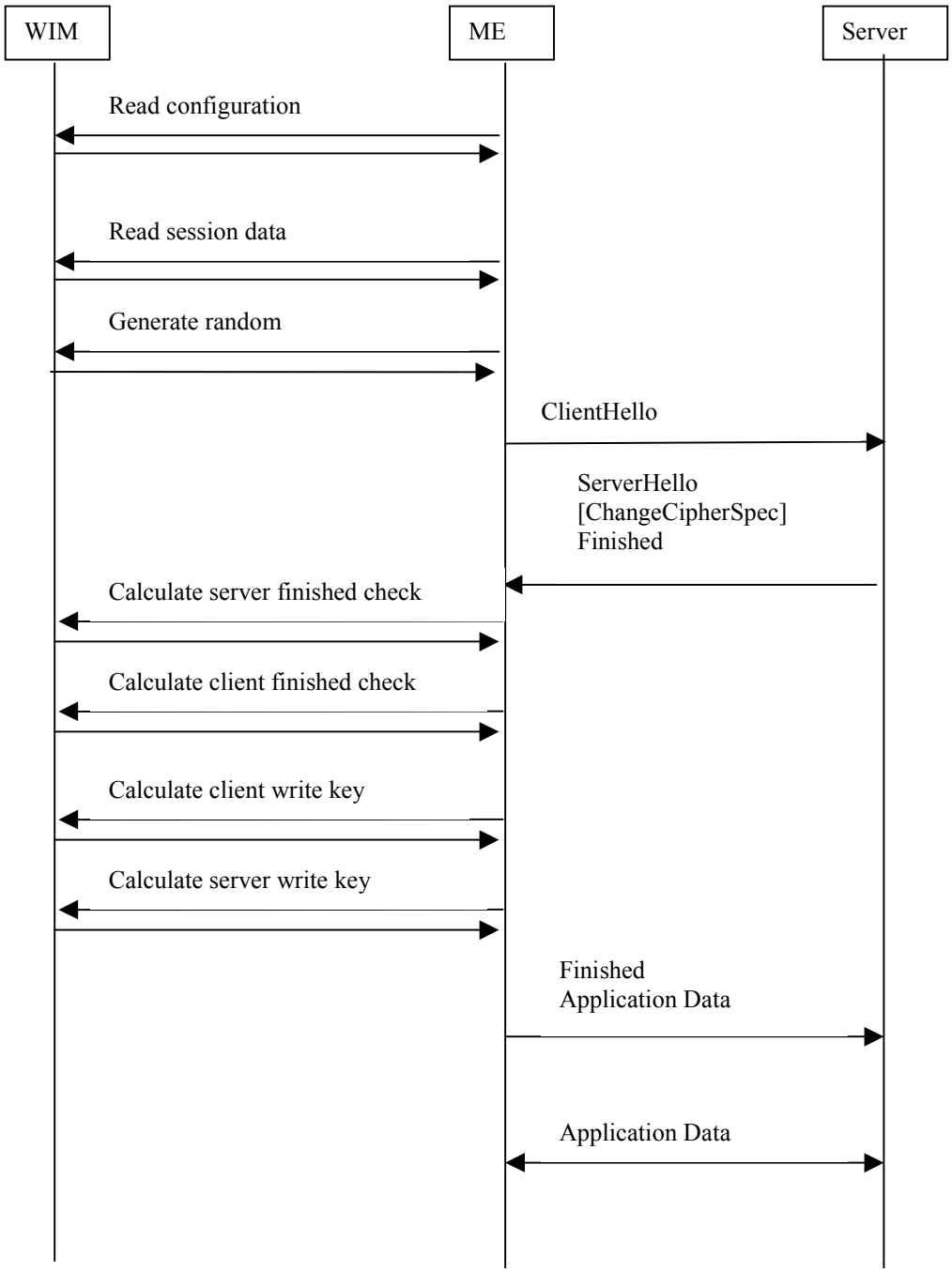


7.3 Abbreviated Handshake

In abbreviated handshake, the WIM is used to calculate all values that are based on the master secret of the current session.

WIM operations used for that are similar to those in a full handshake. Read session data is used to read all address and session related data, needed to resume the session. Note that the ME may keep session related data in a cache in order not to read it from the WIM every time it is needed. The master secret is handled internally by the WIM.

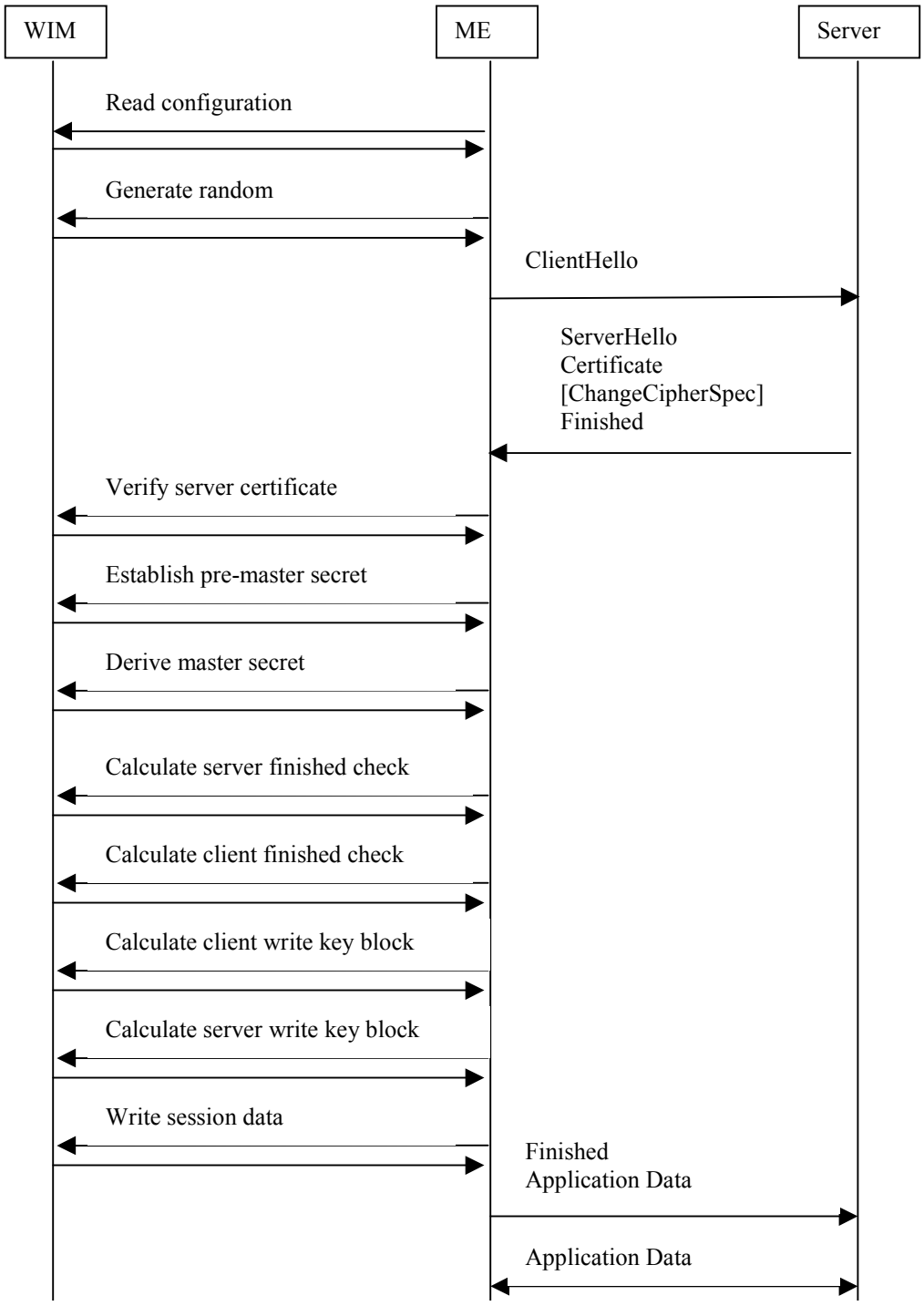
In an abbreviated handshake (resuming a session where client has been authenticated and the WIM was previously used during a full handshake to create the session), the ME MUST use the WIM for calculating needed key material (client finished check, server finished check, client write key block, server write key block).



7.4 Optimised ECDH_ECDSA Handshake

In an optimized ECDH_ECDSA handshake, the server retrieves the client certificate from its own sources, and is able to finish the handshake after receiving ClientHello.

The WIM operations are the same as in the non-optimized ECDH_ECDSA handshake. The client certificate is not read from the WIM.



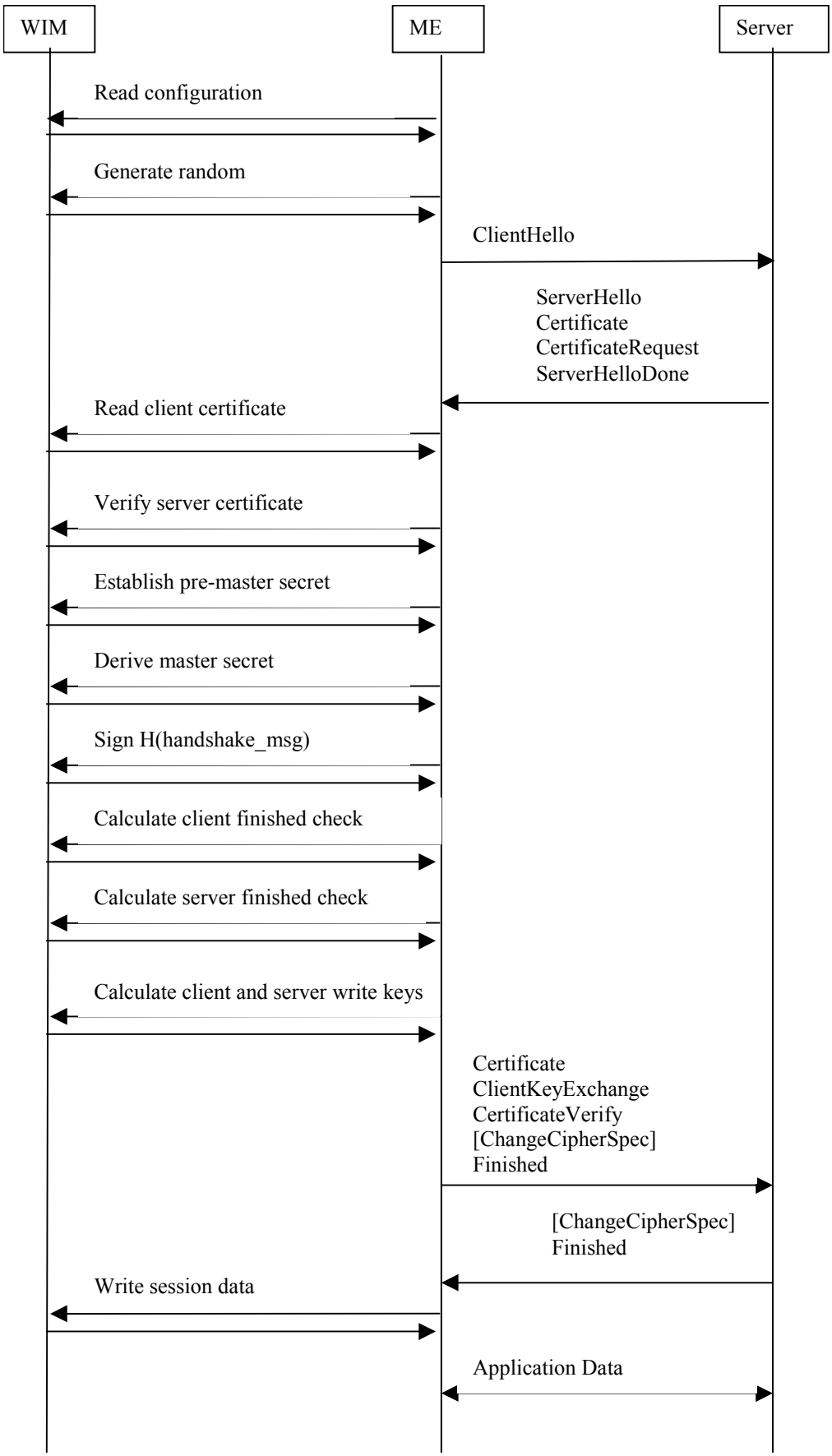
8. WIM Operations in TLS

This chapter describes messages sent during a TLS handshake between the TLS server and client, and between the ME and the WIM. The message flow between the TLS client and server is described in [TLS10]. The message flow between the ME and the WIM is described at a functional level, using service primitives.

WIM implementations that support TLS MUST support RSA handshake. Support of ciphersuites not mandated in [TLSProfile] is not considered here.

8.1 RSA Handshake

In an RSA handshake, the WIM is used to provide client identity and authentication. This involves retrieving the client public key or certificate from the WIM, and performing the signature operation proving the client's identity. The WIM is also used to generate a high quality random number for the pre-master secret, and to encrypt the pre-master secret with the server public key, derive the master secret and calculate all values based on the master secret.



Read configuration

As in WTLS RSA handshake (Section 7.1).

Generate random

The ME may use the WIM to generate 28 random bytes, to be used in the ClientHello message. Primitive used: WIM-GetRandom.

Read client certificate

[TLS10] does not support URL as a reference to the client certificate. Therefore, if client authentication is appropriate for the selected cipher suite, the server may request the client to send its certificate. The ME may then read a certificate stored in the WIM, using data access primitives: WIM-OpenFile, WIM-ReadBinary etc.

Verify server certificate

As in WTLS RSA handshake (Section 7.1).

Establish pre-master secret

The WIM generates a 48 byte value consisting of the client version number (2 byte) and 46 random bytes. It encrypts the value with the server public key and returns the result to the ME. The pre-master secret is the 48 byte value. The WIM keeps the pre-master secret in its memory for the next operation.

The ME uses the WIM-KeyTransport primitive. The primitive returns the encrypted value, to be sent to the server.

Derive master secret

Using the TLS PRF, the WIM derives a master secret based on the pre-master secret established in the previous operation, and a seed value received from the ME. TLS PRF using SHA-1 and MD5 MUST be used. The WIM stores the master secret persistently, to be accessible under a certain session/key id.

The ME uses the WIM-DeriveMasterSecret primitive. The parameter Master Secret ID identifies the resulting master secret. The Input Data parameter has the value “master secret” + ClientHello.random + ServerHello.random

Sign H(handshake_messaged)
Calculate client finished check
Calculate server finished check

As in WTLS RSA handshake (Section 7.1).

Calculate client and server write keys block

Using the TLS PRF, the WIM calculates a requested number of bytes based on the master secret, and a seed value received from the ME. The WIM returns the bytes to be used by the ME as the client write key block.

The ME uses the WIM-PHash primitive with a Input Data parameter

“key expansion” + server_random + client_random

The primitive returns as many bytes as requested.

Write address and session data

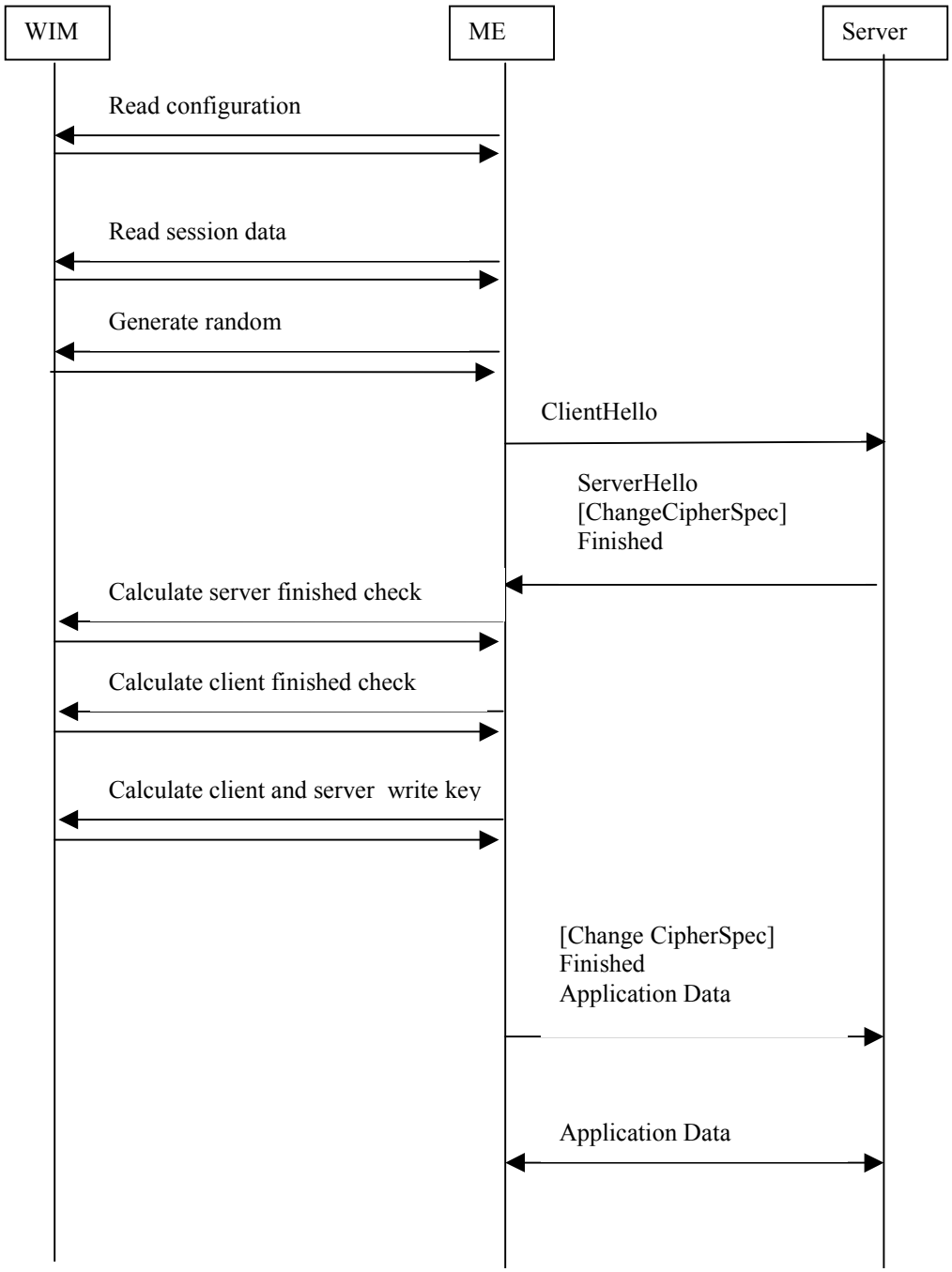
As in WTLS RSA handshake (Section 7.1).

8.2 Abbreviated Handshake

In abbreviated handshake, the WIM is used to calculate all values that are based on the master secret of the current session.

WIM operations used for that are similar to those in a full handshake. Read session data is used to read all session related data, needed to resume the session. Note that the ME may keep session related data in a cache in order not to read it from the WIM every time it is needed. The master secret is handled internally by the WIM.

In an abbreviated handshake (resuming a session where client has been authenticated and the WIM was previously used during a full handshake to create the session), the ME MUST use the WIM for calculating needed key material (client finished check, server finished check, client and server write key block).



9. Information Format

The WIM needs to store the following data

- information on properties of the module: supported algorithms etc
- key pairs for authentication, key establishment and digital signature
- own certificates for each key pair
- trusted CA certificates
- data related to WTLS sessions (including WTLS master secrets)
- data related to TLS sessions (including TLS master secrets)
- information on protection of the data with PINs

Some data is accessed only internally by the WIM, eg, private keys, so it can be implementation specific.

Note that besides the above, there may be data not related to security. Eg, some data may be related to portability, ie, the possibility to store user related information on the WIM, in order to be able to change a portable WIM device into another phone and have access to the information saved earlier.

The WIM information is formatted according to the PKCS#15 [PKCS15]. This chapter explains how PKCS#15 is applied in the WIM.

The format is described using the concept of Dedicated Files (DF) and Elementary Files (EF) defined in [ISO7816-4] for smart cards. A file can be referenced with a file path. For other storage media, an analogous concept may be used.

9.1 Contents of the Files

All Elementary Files (EF) are binary (transparent) files. This makes it possible to access one or several logical records at a time, or a part of a logical record. The Peers EF and Sessions EF data consists of fixed length records. Therefore, the ME should calculate the proper offset based on the known logical record lengths for these two files.

Data syntax of most files is in accordance with PKCS#15, and is described using ASN.1 [ASN1] and DER [DER]. The guideline has been to use as simple ASN.1 structures as possible, to enable easy parsing by the ME. Some WTLS specific data is described using the WTLS presentation language [WAPWTLS]. For most places, the file syntax is interpreted by the ME only, the WIM acting as a storage media.

In accordance with PKCS#15, records may be erased by replacing the outermost tag with a ‘00’ byte, though note that this is not consistent with [ISO7816-4]. In accordance with [ISO7816-4], padding bytes may be inserted before, between or after TLV-coded record objects. Such padding bytes must be of value ‘FF’. The EF(UnusedSpace) file must not be used to indicate unused space/records within a EF(DF). Instead, the methods indicated above (a zero tagged record and/or padding with 'FF' bytes) should be used.

9.2 WTLS `bitmask` Type

To describe single bits in WTLS presentation, a bitmask type is used:

```
bitmask { b0(v0), b1(v1), …, bn(vn), [[n]] }
```

A bitmask occupies as many bytes as needed to store the maximum value, taking into account that one byte can store 8 bits. Eg, a bitmask can be 1 or 2 bytes, corresponding to 8 or 16 bits. Bit 0 is the most significant bit. So, if the bitmask is 1 byte, the bit zero has the value 0x80.

Eg, the following definition would cause one byte to be used to carry fields of type CarOptions.

```
bitmask { conditioning(0), airbag(1), automatic(2), (7) } CarOptions;
```

For

```
CarOptions car_options = conditioning | automatic;
```

the value would be 80 | 20 = A0.

9.3 ISO Object Identifiers

ISO Object Identifiers (OID) are needed for certain objects. When possible, OIDs assigned by relevant standard bodies are used. For WAP specific objects, WAP has its own OID tree:

wap OBJECT IDENTIFIER ::= {joint-iso-itu-t(2) identified-organizations(23) 43}

wap-wsg OBJECT IDENTIFIER ::= {wap 1}

9.4 PKCS#15 Application Directory Contents

This section describes the EFs of the PKCS#15 application directory, DF(PKCS15). The reader should read this chapter along with [PKCS#15].

9.4.1 EF(ODF)

The mandatory Object Directory File (ODF) ([PKCS15], section 5.5.1) consists of pointers (file paths) to other EFs (PrKDFs, PuKDFs, CDFs, DODFs and AODFs), each one containing a directory over PKCS#15 objects of a particular class (here and below, a “directory” means a list of objects).

The EF(ODF) is not modifiable by the user.

Contents of a file path field are according to [PKCS15]. If it is two bytes long, it references a file by its file identifier. If it is longer than two bytes, it references a file either by an absolute or relative path (i.e., concatenation of file identifiers) where 'relative' means relative to the WIM DF.

If paths (i.e., not just file identifiers) are used in referencing from different object directories, then the ODF MUST contain information indicating the WIM DF (i.e., the current DF after application selection; the ME needs this information to get back to the WIM DF after selecting another DF). The information is included by letting the ODF contain a record of type PKCS15Objects.dataObjects, which use the PathOrObjects.objects alternative. The object contained is a single opaqueDO. In this object, the CommonDataObjectAttributes.applicationOID has the value

wap-wsg-wimpath OBJECT IDENTIFIER ::= {wap-wsg 3 }

ObjectValue.direct field has as the value the absolute path of the WIM DF as an OCTET STRING (eg, 3F 00 50 15).

An example EF(ODF):

```
{
    privateKeys  : path : {
        path '4401'H -- Reference by file identifier
    },
    certificates : path : {
        path '4402'H -- Reference by file identifier
    },
    dataObjects : path : {
        path '4403'H -- Reference by file identifier
    },
    dataObjects  : objects {
             opaqueDO : {
                          commonObjectAttributes {},
                          classAttributes {
                                       applicationOID {wap-wsg-3}
                          },
                          typeAttributes direct : OCTET STRING:'3F00 5015'H
             }
    }

}
```

9.4.2 Private Key Directory Files (PrKDFs)

The Private Key Directory Files ([PKCS15], section 5.5.2) contain directories of private keys known to the PKCS#15 application. At least one PrKDF MUST be present in a WIM.

Each logical record of a PrKDF, describing a single private key, MUST contain the following fields

- human readable label to describe the key (`commonObjectAttributes.label`)
- common flags (`commonObjectAttributes.flags`)
- identifier for the associated authentication object (`commonObjectAttributes.authId`)
- 20-byte public key SHA-1 hash, as defined in [PKCS15], used as a key identifier, to correlate private keys, associated public keys and certificates (the keyIdentifiers field MUST be omitted) (`CommonKeyAttributes.iD`)
- usage field (`CommonKeyAttributes.usage`)
- card specific key reference used to reference the key in cryptographic operations. It is used by the card to identify the private key in the path (`CommonKeyAttributes.keyReference`)
- reference to corresponding entry in EF(TokenInfo) (`keyInfo`). This field is required only if there are several algorithm related entries in the EF(TokenInfo), ie, if the device supports several algorithms or an algorithm with several different parameters (eg, elliptic curve).
- for an RSA key, the modulus length (`modulusLength`)
- file path, to be used in setting the Security Environment (the `index` and `length` fields MUST be omitted). For PrKDF and other files, the path field according to [PKCS15]. If it is two bytes long, it references a file by its file identifier. If it

is longer than two bytes, it references a file either by an absolute or relative path (ie, concatenation of file identifiers) where 'relative' means relative to the PrKDF or other directory file.

The PrKDFs are typically not modifiable by the user, since generating keys is assumed to a part of the manufacturing/personalization process.

Specificities of PrKDF used for on-board key generation

A `CommonKeyAttributes.iD` field with 20 bytes all set to zero means that the key is not initialized i.e. the on-board key generation operation may update the above value to the valid public key SHA-1 hash.

A type `KeyInfo` with the optional `supportedOperations` field MUST be present with the flag `KeyInfo.paramsAndOps.SupportedOperations.generate-key` set, to indicate a PrKDF for which the key generation is possible.

9.4.3 Public Key Directory Files (PuKDFs)

The Public Key Directory Files ([PKCS15], section 5.5.4) contain directories of public keys known to the PKCS#15 application. At least one PuKDF MUST be present on a WIM which contains public keys.

Note that a WIM may contain no public keys as such since it may be enough to store a certificate containing a public key. Note that the public key hash (needed in some operations) is used as the key identifier for private keys and associated certificates.

9.4.4 Certificate Directory Files (CDFs)

The Certificate Directory Files ([PKCS15], section 5.5.5) contain directories of certificates known to the PKCS#15 application. At least one CDF MUST be present on a WIM which contains certificates (or references to certificates).

Each logical record of a CDF, describing a single certificate, MUST contain the following fields

- human readable label to describe the certificate (`commonObjectAttributes.label`)
- common flags (`commonObjectAttributes.flags`)
- 20-byte public key SHA-1 hash, as defined in [PKCS15], to correlate the certificate with a certain private key (`CommonCertificateAttributes.iD`)
- file path, index (binary offset), and length, to be used in selecting the file and binary read operations, or a certificate url
- the 20-byte public key SHA-1 hash of the issuer key (`CommonCertificateAttributes.requestId`) (this field need not used for root CA certificates, unless it is necessary for maintaining a fixed record length)

Note that referencing a certificate using an url is not compatible with [TLS10].

A logical record of a CDF, describing a CA certificate MUST also contain the field `CommonCertificateAttributes.authority`.

Each logical record of a CDF, describing a single certificate, MAY contain the following optional field:

- hash of the certificate (`CommonCertificateAttributes.certHash`). The hash is calculated over the DER encoding of the complete certificate. This field can be used to send the hash of a cert as an authenticated attribute when the cert itself is not available, i. e. the URL is available only.

A CDF pointed by a `certificates` field in the ODF, contains references to certificates issued to the WIM user. Such a CDF MAY also contain a field containing the hash of the issuer name from the certificate (CommonCertificateAttributes.identifiers, the issuerNameHash KEY-IDENTIFIER). This field is used to speed up searches for a particular client certificate, e.g. when performing client-side authenticated TLS. (If some user certificates are modifiable and some are not then there may be two distinct CDFs pointed by a `certificates` field in the ODF. One is grouping all references to modifiable certificates and another is grouping all references to non-modifiable certificates).

A CDF pointed by a `trustedCertificates` field in the ODF, contains references to trusted certificates. Both the CDF and the EF(s) containing the certificate data pointed to MUST NOT be modifiable by the user. These certificates are considered trusted by the WIM issuer and should thus be trusted by the user, too. The ME to verify a server in a WTLS or TLS handshake, or to verify signatures in downloaded content, eg, downloaded applications can use them. Such a CDF MAY also contain an optional field to store a sequence of OIDs that represent privileges associated with the certificate (`CommonCertificateAttributes.trustedUsage`), and a field containing the hash of the subject name from the certificate (`CommonCertificateAttributes.identifiers`, the `subjectNameHash` KEY-IDENTIFIER). The latter field is used to speed up searches for a particular trusted certificate, e.g. when verifying signed content.

A CDF pointed by a `usefulCertificates` field in the ODF, contains references to CA certificates that are updateable by the user.

9.4.5 Data Object Directory Files (DODFs)

These files contain directories of data objects (not keys or certificates) ([PKCS15], section 6.5.5) known to the PKCS#15 application. At least one DODF must be present on a WIM.

WTLS and TLS session related data is referenced using opaque Data Objects ‘Peers-wtls’, ‘Sessions-wtls’ and ‘Sessions-tls’ described in a single DODF, DODF-wim. The opaque Data Object ‘Peers-tls’ is not defined in this version of the specification.

The logical records of the DODF-wim MUST contain the following fields

- flags (private, modifiable)
- authentication object identifier, indicating the authentication object protecting the file itself and also protecting usage of the master secrets handled internally by the WIM
- object identifier indicating ‘Peers-wtls’, ‘Sessions-wtls’ and ‘Sessions-tls’, see below
- file path, index (binary offset), and length, to be used in selecting the file and binary read operations

Informational note: label and applicationName attributes should be omitted to save memory space.

The DODF-wim SHOULD NOT be modifiable by the user.

For other possible data objects, a separate DODF is used. The ME MUST support more than one DODF.

9.4.6 Authentication Object Directory Files (AODFs)

The Authentication Object Directory Files ([PKCS15], section 5.5.7) contain directories of authentication objects (e.g. PINs) known to the PKCS#15 application. At least one AODF must be present on a WIM, which contains authentication objects coupled to the PKCS#15 application.

Each logical record of a AODF, describing a single authentication object, MUST contain the following fields

- human readable label to describe the PIN

- common flags (value: private)
- authentication object identifier
- pin flags (`PinAttributes.pinFlags`)
- type of PIN (`PinAttributes.pinType`)
- minimum length (`PinAttributes.minLength`)
- stored length (`PinAttributes.storedLength`)
- padding character (`PinAttributes.padChar`)
- qualifier of the reference data (to be used as the P2 parameter in verification related ICC commands) (`PinAttributes.pinReference`)
- file path to be used for verification related operations (the index and length fields MUST be omitted) (`PinAttributes.path`)

The first object in the first AODF listed in EF(ODF) is considered as a General PIN (PIN-G). If not otherwise indicated, all files (eg, CDF, PrKDF) are protected with this PIN.

9.4.7 EF(TokenInfo)

The mandatory TokenInfo elementary file ([PKCS15], section 5.5.8) shall contain generic information about the token as such and it's capabilities, as seen by the PKCS15 application.

The EF(TokenInfo) indicates predefined SE numbers, like WTLS_RSA, TLS_RSA, WIM_GENERIC_RSA, WTLS_ECDH and WIM_GENERIC_ECC. The numbers are indicated with ISO Object Identifiers:

wap-wsg-idm-se OBJECT IDENTIFIER ::= {wap-wsg 1}

wap-wsg-idm-se-wtlsrsa OBJECT IDENTIFIER ::= {wap-wsg-idm-se 1}

wap-wsg-idm-se-wimgenericrsa OBJECT IDENTIFIER ::= {wap-wsg-idm-se 2}

wap-wsg-idm-se-wtlsecdh OBJECT IDENTIFIER ::= {wap-wsg-idm-se 3}

wap-wsg-idm-se-wimgenericecc OBJECT IDENTIFIER ::= {wap-wsg-idm-se 4}

wap-wsg-idm-se-tlsrsa OBJECT IDENTIFIER ::= {wap-wsg-idm-se 5}

The TokenInfo contains the following fields (all fields MUST be present unless otherwise stated below)

- version (for this specification, the version is v1) (`version`)
- serial number that uniquely identifies the device (`serialNumber`). In case of an ICC, this is the ICC identification number, as specified in [PKCS15]. Even for non-ICC implementations it is recommended to use an ISO/IEC 7812-1 conformant number whenever it is possible. As an alternative, a hash of one of the public keys in the device may be used. (It is not essential which public key is used as input. The public key hash is calculated as specified in [PKCS15].) It should be possible to display this value using the ME. In some cases it may be possible to print this number on the WIM device.
- manufacturer information (`manufacturerID`) (optional)

- application label (`tokenInfo.label`). This field MUST begin with the version text which for this version is "WIM 1.01". If any additional identifying information of the application is to be appended to the version text then the additional information MUST be separated from the version text by a space character.
- flags (`tokenflags`)
- predefined security environments (`seInfo`)
- supported algorithms (`supportedAlgorithms`)

9.4.8 EF(UnusedSpace)

The UnusedSpace elementary file ([PKCS15], section 5.5.9) is used to keep track of the unused space in already created elementary files.

9.4.9 Other elementary files in the PKCS#15 directory

These files will contain the actual values of objects (such as private keys, public keys, certificates and application specific data) referenced from within PrKDFs, PuKDFs, CDFs or DODFs.

9.4.10 'Peers-wtls' Data Object

The 'Peers' data object is a PKCS#15 opaque data object. The object type is identified with an application OID (CommonDataObjectAttributes.applicationOID).

wap-wsg-idm-file OBJECT IDENTIFIER ::= {wap-wsg 2}

wap-wsg-idm-file-peer OBJECT IDENTIFIER ::= {wap-wsg-idm-file 1}

wap-wsg-idm-file-session OBJECT IDENTIFIER ::= {wap-wsg-idm-file 2}

Informational note : the label (`CommonObjectAttributes.label`) and application name

(`CommonDataObjectAttributes.applicationName`) should be omitted to save memory space.

The actual data is contained in a separate file, pointed to by Path. The data contains records each of which represents one peer with a link to one secure session. Note that each peer is identified with an (address, port) pair. Many such pairs can potentially use a single secure session (eg, different bearers for a single WAP server).

```
bitmask { in_use(0), favourite(1), (7) } EntryOptions;
```

Item	Description
in_use	The entry is in use. (Unused entries should have initial values 00h.)
favourite	The ME SHOULD give a favourite entry preference, when not all entries can be kept in the WIM, due to space or other limitations.

```
struct {
   EntryOptions        entry_options;
   uint8               session_number;
   uint16              port;
   opaque              address[18];
} PeerEntry;
```

Item	Description
entry_options	Options for this entry.
session_number	The record number in the Sessions EF (1, 2, …). This number is also equal to the key reference of the master secret.
port	Port number, as specified in [WAPWDP].
address	Address, as specified in [WAPWCMP] (The specification contains address type, length and value. The specified address is left justified and padded with 'FF'). The 18-byte address has internally a TLV structure. So, the maximum length of the address value is 16 bytes, sufficient for ipv6 address.

9.4.11 'Sessions-wtls' Data Object

The 'Sessions' data object is a PKCS#15 opaque data object. The object type is identified with an application OID (CommonDataObjectAttributes.applicationOID).

Informational note : the label (`CommonObjectAttributes.label`) and application name

(`CommonDataObjectAttributes.applicationNam`e) should be omitted to save memory space.

The actual data is contained in a separate file, pointed by Path. The data contains records, each of which represents one secure session. The data included there includes all information to resume a session.

```
bitmask { resumable(0), server_authenticated(2), client_authenticated(3), (7) }
SessionOptions;
```

Item	Description
resumable	The session is resumable, ie, it is possible to make an abbreviated handshake using the session_id.
server_ authenticated	The server has been authenticated based on a certificate.
client_ authenticated	The client has been authenticated based on a certificate.

```
struct {
   EntryOptions        entry_options;
   SessionOptions      session_options;
   uint8               session_id_length;
   opaque              session_id[8];
   MACAlgorithm        mac_algorithm;
   BulkCipherAlgorithm bulk_cipher_algorithm;
   CompressionMethod   compression_algorithm;
   uint8               private_key_id[4];
   uint32              creation_time;
} SessionEntry;
```

Item	Description
entry_options	Options for this entry.
session_options	Options for the session.
session_id_length	Length of the session id.
session_id	Id of the secure session.
mac_algorithm	Algorithm used for message authentication.
bulk_cipher_ algorithm	Algorithm used for bulk encryption.
compression_ algorithm	Algorithm to compress data.
private_key_id	First 4 bytes of the private key Id (Identifier, calculated as hash of the public key) of the client private key used during the handshake. The client can use this information for optimization of new handshakes.
creation_time	Time of the creation of the session (from ServerHello message).

The ME SHOULD maintain a correspondence between the record number in the Sessions EF (1,2,…) and the master secret key reference used in MSE - Derive Key. This will help ensure reliable resumption of secure sessions.

9.4.12 'Peers-tls' Data Object

The 'Peers-tls' data object is a PKCS#15 opaque data object. Its internal structure is not defined in this version of the specification. The following application OID is reserved for future use.

wap-wsg-idm-file-peer-tls OBJECT IDENTIFIER ::= {wap-wsg-idm-file 3}

9.4.13 'Sessions-tls' Data Object

The 'Sessions' data object is a PKCS#15 opaque data object. The object type is identified with an application OID (`CommonDataObjectAttributes.applicationOID`).

wap-wsg-idm-file-session-tls OBJECT IDENTIFIER ::= {wap-wsg-idm-file 4}

Informational note : the label (`CommonObjectAttributes.label`) and application name

(`CommonDataObjectAttributes.applicationNam`e) should be omitted to save memory space.

The actual data is contained in a separate file, TLS Sessions EF, pointed by Path. The data contains records, each of which represents one secure session. The data included there includes information that may be used by the ME to check that the session in the WIM corresponds to the session stored in the ME memory.

```
struct {
   opaque              check[4];
} TlsSessionEntry;
```

Item	Description
Check	A 4-byte check value. This value can be used by the ME in order to check that a session stored in the ME corresponds to a session stored in the WIM.

The ME should maintain a correspondence between the record number in the TLS Sessions EF (1,2,…) and the master secret key reference used in MSE - Derive Key. The ME should, before trying resumption, check that the check value corresponds to a value stored internally by the ME. This will help ensure reliable resumption of secure sessions.

The ME to determine the number of TLS sessions supported by the WIM can use the number of the records.

A (non-zero) random value should be used as the check value. A value where all bytes are zero, is reserved to indicate an unused session.

A collision of two check values is unlikely. In order this to happen, the same WIM device would have to be used in two terminals during a lifetime of a TLS session (typically less than 24 hours). Additionally, the check values would have to collide. In the unlikely event of collision, resumption of a TLS session would fail with an error condition, and a new session would have to be created.

9.5 An Example WIM Layout

Below is an example of the WIM layout. The EFs that are implementation dependant are with *italics*.

MF

EF(DIR)
DF Another Application
…

DF(PKCS15)

EF(TokenInfo)
EF(ODF)
EF(AODF)
EF(PrKDF)
EF – Private key for authentication and key exchange
EF – Private key for digital signatures
EF(CDF) – for user certificates
EF – Certificate for authentication and key exchange key
EF – Certificate for non-repudiation key
EF(CDF) – for CA certificates that can be updated by the user
EF – CA certificate stored by user
EF(CDF) – for certificates that are read-only (“trusted certificates”)
EF – CA certificate stored by WIM issuer
EF(DODF-wim)
EF – Master secrets of WTLS sessions
EF – Master secrets of TLS sessions
EF – Peers-wtls
EF – Sessions-wtls
EF – Sessions-tls
EF(UnusedSpace)

10. Security Environments

The concept of a Security Environment (SE) is defined in [ISO7816-8] for smart cards. For this specification, this concept may be applicable to also other types of WIM implementation.

10.1 Security Environment Definition

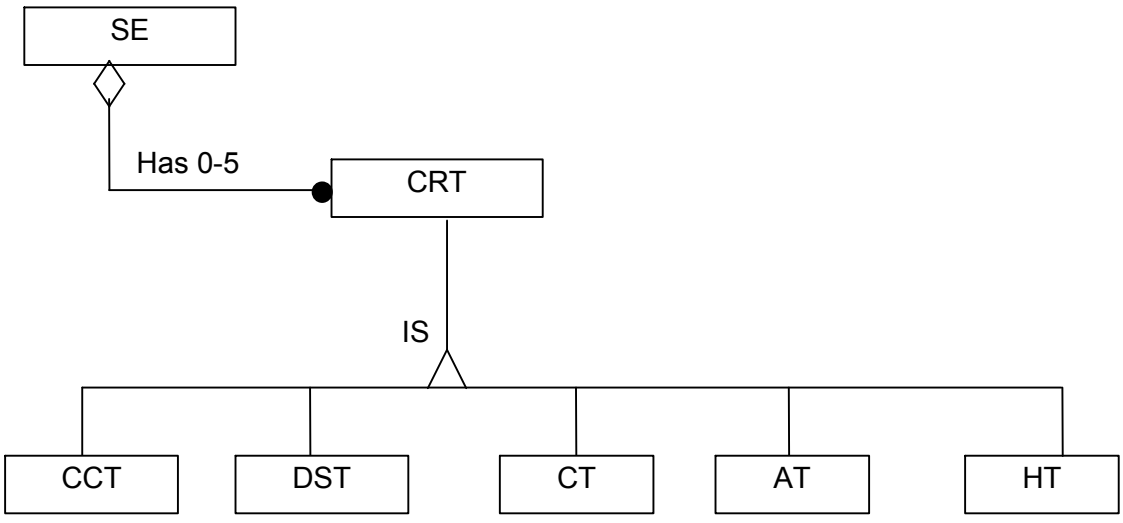
A Security Environment (SE) is a logical container of a set of fully specified security mechanisms which are available for reference in security related commands. Each SE specifies references to the cryptographic algorithm(s) to be executed, the mode(s) of operation, the key(s) to be used and any additional data needed by a security mechanism. It may define templates describing data elements stored in the WIM or resulting from some computation. It may also provide directions for handling the data resulting from a computation to be stored in the WIM.

An SE may contain several CRTs (Control Reference Templates) each of which is of a different type as follows:

- DST Digital signature template
- CT Confidentiality template
- CCT Cryptographic checksum template
- AT Authentication template
- HT Hash Template

There is only one CRT of each type in an SE, which means a maximum of five templates. Each template is an instance of the above types.

The following OMT diagram illustrates the relations between SE, CRT and CRT's subtypes.



Using this framework we can define N Security Environments, each of them can be used for another application context and has a different number. To switch between the different SEs we use the RESTORE command followed by the SE number:

MANAGE SECURITY ENVIRONMENT – Restore (SE number)

There are two types of operations that can be performed in Security Environments:

- MANAGE SECURITY ENVIRONMENT (MSE)
- PERFORM SECURITY OPERATION (PSO)

MSE Set is used to set attributes values within a CRT in the currently selected SE. It has the role of preparing all needed values and references that will be used in the following PERFORM SECURITY OPERATION commands. The PERFORM SECURITY OPERATION command, on the other hand, selects and execute a specific security operation (ComputeDigitalSignature, Encipher etc.).

The MANAGE SECURITY ENVIRONMENT can set, for example, the private key reference in a Digital Signature Template (DST). If this private key reference was set to 1, for example, the following PSO ComputeDigitalSignature will look for the private key which has reference 1 in the private key files that are defined within the application context, and use it in the digital signature calculation.

Each SE has several CRTs with several predefined attributes which can be set by the ME before using this template. We will use a C++ like notation to list all the attributes and the available operations within each CRT in each SE. This notation is used for illustration purposes, not to define any implementation.

In accordance with [ISO7816-8], references to attributes associated to a SE and CRTs are resolved with respect to the DF selected at the time the security mechanism is used to perform a computation. It will be the responsibility of the ME to ensure that it has selected the DF relevant to an application before executing any commands in order to establish a secure session for that application. In that way the data relevant to that application (eg, private keys) will be used when the WIM executes the security mechanisms.

Each attribute will have one of the following types:

Ref	Is a pre-defined logical reference to an item. (For a key, it denotes a file path and a key reference.)
Key	Is a byte string representing a key for a cryptography algorithm. It can be of any length.

When we precede an attribute type with **Private** it means that this attribute cannot be read nor changed by the ME. It is shown only in order to help the reader understand how information can be found and how the security mechanisms operate in the security environments.

When we precede an attribute type with **Transient** it means that each time this attribute is set, it is used in the following PERFORM SECURITY OPERATION only. After the execution of this PSO the attribute value is set to “undefined” and it means that its value will not influence nor be used in the following PSO operations.

These kind of attributes are usually being used in a sequence of MSE and PSO operations that form together a specific function. Usually the sequence of operations is as follows:

- MSE Set – in order to set the value of the attribute
- PSO. The PERFORM SECURITY OPERATION uses that value and set its value to “undefined” after the operation finished. “Undefined” means that this attribute value will not influence nor be used in the next PSO operation.

PERFORM SECURITY OPERATION (PSO) is executed within a certain CRT context only. This means that these operations use the attributes that are already set within this CRT. In Object Oriented design we will say that the command for the operation is sent to the CRT object. If the relevant CRT does not exist within the current SE the command will fail.

For example the `PSO-ComputeDigitalSignature` is processed within the DST in the currently selected SE. It uses the `privateKeyRef` along with other attributes which are set within the DST. If there is no DST within the currently selected SE the APDU command to invoke the above function will fail and the returned status word will indicate this.

10.2 WTLS Security Environments

For the WAP-WTLS application there are two predefined SEs with their associated number. These Security Environments are already pre-configured to provide the needed contexts to execute all needed WTLS operations. The defined WTLS Security Environments are:

WTLS_RSA Security Environment which is already pre-configured for handling the RSA key exchange suite in the WTLS handshake protocol.

WTLS_ECDH Security Environment which is already pre-configured for handling the ECDH exchange suite in the WTLS handshake protocol.

In the next section we list all WTLS Security Environments along with their CRTs. Each CRT is presented with its internal attributes and the operations that are being performed within its context.

10.2.1 WTLS_RSA Security Environment

The WTLS_RSA SE is being used in all WTLS Key Exchange Suites that involve the RSA algorithm. This is done in the Handshake phase of the WTLS protocol. In the WTLS_RSA Security Environment there are three CRTs:

DST - For the digital signatures and associated operations

CT - For key transport

CCT - For deriving master secret and keyblock calculation

10.2.1.1 DST

```
WTLS_RSA_DST {
               Ref                     algorithmRef;
               Ref                     privateKeyRef;
   Private     Key                     privateKey;

               Key                     verificationPublicKey;
   Transient   Byte                    verificationDigest[];

   OPERATIONS:
       MSE-Set(Ref privateKeyRef, …);
       PSO-ComputeDigitalSignature(byte[] stringToSign, …);

       MSE-Set(Key verificationPublicKey, byte[] verificationDigest, …);
       PSO-VerifySignature(byte[] Signature, …);

};
```

Attributes description:

`algorithmRef`	The reference of the RSA algorithm used in this template. The algorithms that the WIM support are described in a file in the WIM. (Note: In the WTLS_RSA SE it is already defined according to the WTLS specification and should not be changed).
`privateKeyRef`	The reference of the private key to use in the `PSO-ComputerDigitalSignature` operation. The private key file itself is being searched in the application context.
`privateKey`	The private key that is indicated with the `privateKeyRef`. The private key is searched and found in application context.
`verificationPublicKey`	The public key used for signature verification (ie, public key corresponding to the private key that was used for signing the `verificationDigest`).
`verificationDigest`	The digest used in signature verification.

Operations:

`MSE-Set`	Used to set attribute values in the DST.
`PSO-Compute DigitalSignature`	Compute a digital signature using the predefined attributes that were set in the DST and the parameters sent with this command.

PSO-VerifySignature	Verify a signature using the predefined attributes that were set (verificationPublicKey, verificationDigest) and the parameters sent with this command.

10.2.1.2 CT

```
WTLS_RSA_CT {
              Key          serverPublicKey;
   Transient  Byte         clientVersion;
   Transient  Byte         randomNumber[19];
              Ref          algorithmRef;

   OPERATIONS:
           MSE-Set(Byte clientVersion, Key serverPublicKey, …);
           PSO-Encipher(…);
};
```

Attributes description:

serverPublicKey	The public key of a server that is set via the MSE-Set command. The PSO-Encipher operation will use this key to encipher data that will then be sent to that server.
clientVersion, randomNumber	**These attributes are set and used in the "WIM-KeyEstablish" WTLS function implementation.**
algorithmRef	**The reference of the RSA algorithm used in this template. The algorithms that the WIM supports are described in a file in the WIM. (Note: In the WTLS_RSA SE it is already defined according to the WTLS specification and should not be changed.)**

Operations:

MSE-Set	Is used to set attribute values in the CT.
PSO-Encipher	Compute a cryptogram using the predefined attributes that were set in the CT (eg. serverPublicKey) and the parameters sent with this command (plain value for example). The cryptogram is the pre-master secret and is set in the CCT template if the operation completed successfully

10.2.1.3 CCT

The CCT is used to implement the WTLS PRF function.

```
WTLS_RSA_CCT {
   Transient   Key     preMasterSecret;
               Ref     masterSecretRef;
   Private     Key     masterSecret;
               Byte    ResultLength;
               Ref     algorithmRef;

   OPERATIONS:
       MSE-DeriveKey(Ref masterSecretRef, Byte[] inputData, …);
       PSO-ComputeChecksum(Byte[] inputData);
};
```

Attributes description:

`preMasterSecret`	Calculated and internally set in the WIM-KeyTransport primitive. It is then used in the `PSO-DeriveKey` operation to calculate the master secret. (master secrets belong to different sessions with remote servers and are stored in a private "master secret" file in the WIM).
`masterSecretRef`	The reference of the master secret.
`masterSecret`	The value of the master secret, stored privately by the module.
`resultLength`	The expected length of the result to calculate by the PRF function
`algorithmRef`	The reference of the PRF algorithm used in this template.

Operations:

MSE-DeriveKey	Derive a certain master secret based on a pre-master secret, available in the module.
PSO-ComputeChecksum:	Compute a cryptographic checksum (using the WTLS PRF algorithm) based on a selected master secret and input data.

10.2.2 WTLS_ECDH SECURITY ENVIRONMENT

The WTLS_ECDH SE is being used in all WTLS Key Exchange Suites that involve the ECDH algorithm. This is done in the Handshake phase of the WTLS protocol. In the WTLS_ECDH Security Environment there are three CRTs:

DST - For the digital signatures and associated operations

CT - For key agreement

CCT - For deriving master secret and keyblock calculation

10.2.2.1 DST

The attributes and operations in this template are the same as in the WTLS_RSA_DST except that the used algorithm is ECDSA.

Note that currently defined WTLS key exchange mechanisms do not have ECDSA operation.

10.2.2.2 CT

The Confidentiality Template in this SE is used for EC Diffie-Hellman key agreement. The PSO-Encipher operation in this template implements the ECDH key agreement calculation.

```
WTLS_ECDH_CT {
               Key         serverPublicKey;
               Ref         privateKeyRef;
   Private     Key         privateKey;
               Ref         algorithmRef;

   OPERATIONS:
           MSE-Set(Key serverPublicKey, Ref privateKeyRef, …);
           PSO-Encipher(…);
};
```

Attributes description:

`serverPublicKey` The public key of a server that is set via the MSE-Set command. The `PSO-Encipher` operation will use this key to perform the ECDH calculation of the shared secret.

`privateKeyRef` The reference of the private key to use in the ECDH calculation. The private key file itself is being searched in the application context.

`privateKey` The private key that is indicated with the `privateKeyRef`. The private key is searched and found in application context.

`algorithmRef` **The reference of the ECDH algorithm used in this template. The algorithms that the WIM support are described in a file in the WIM. (Note: In the WTLS_ECDH SE it is already defined according to the WTLS specification and should not be changed.)**

Operations:

`MSE-Set` Used to set attribute values in the CT.

`PSO-Encipher`	Computes a shared pre-master secret using an ECDH algorithm, using `serverPublicKey` and `privateKey`. The calculated pre-master secret is set in the CCT template if the operation completed successfully.

10.2.2.3 CCT

The CCT is used to implement the WTLS PRF function. It has the same functionality as in the WTLS_RSA SE.

10.3 TLS Security Environments

For the WAP-TLS application there is one predefined SE with its associated number. This Security Environment is already pre-configured to provide the needed contexts to execute all needed TLS operations.

The defined TLS Security Environment is:

TLS_RSA	Security Environment which is already pre-configured for handling the RSA key exchange suite in the TLS handshake protocol.

In the next section we list all TLS Security Environments along with their CRTs. Each CRT is presented with its internal attributes and the operations that are being performed within its context.

10.3.1 TLS_RSA Security Environment

The TLS_RSA SE is being used in all TLS Key Exchange Suites that involve the RSA algorithm.

This is done in the Handshake phase of the TLS protocol. In the TLS_RSA Security Environment there are three CRTs:

DST - For the digital signatures and associated operations

CT - For key transport

CCT - For deriving master secret and keyblock calculation

10.3.1.1 DST

```
TLS_RSA_DST {
               Ref                     algorithmRef;
               Ref                     privateKeyRef;
   Private     Key                     privateKey;

               Key                     verificationPublicKey;
   Transient   Byte                    verificationDigest[];

   OPERATIONS:
       MSE-Set(Ref privateKeyRef, …);
       PSO-ComputeDigitalSignature(byte[] stringToSign, …);

       MSE-Set(Key verificationPublicKey, byte[] verificationDigest, …);
       PSO-VerifySignature(byte[] Signature, …);

};
```

Attributes description:

`algorithmRef`	The reference of the RSA algorithm used in this template. The algorithms that the WIM support are described in a file in the WIM. (Note: In the TLS_RSA SE it is already defined according to the TLS specification and should not be changed).
`privateKeyRef`	The reference of the private key to use in the `PSO-ComputerDigitalSignature` operation. The private key file itself is being searched in the application context.
`privateKey`	The private key that is indicated with the `privateKeyRef`. The private key is searched and found in application context.
`verificationPublicKey`	The public key used for signature verification (ie, public key corresponding to the private key that was used for signing the `verificationDigest`).
`verificationDigest`	The digest used in signature verification.

Operations:

`MSE-Set`	Used to set attribute values in the DST.
`PSO-Compute DigitalSignature`	Compute a digital signature using the predefined attributes that were set in the DST and the parameters sent with this command.

PSO-VerifySignature	Verify a signature using the predefined attributes that were set (verificationPublicKey, verificationDigest) and the parameters sent with this command.

10.3.1.2 CT

```
TLS_RSA_CT {
              Key          serverPublicKey;
   Transient  Byte         clientVersion[2];
   Transient  Byte         randomNumber[46];
              Ref          algorithmRef;

   OPERATIONS:
           MSE-Set(Byte clientVersion, Key serverPublicKey, …);
           PSO-Encipher(…);
};
```

Attributes description:

serverPublicKey	The public key of a server that is set via the MSE-Set command. The PSO-Encipher operation will use this key to encipher data that will then be sent to that server.
clientVersion, randomNumber	**These attributes are set and used in the "WIM-KeyTransport" TLS function implementation.**
algorithmRef	**The reference of the RSA algorithm used in this template. The algorithms that the WIM supports are described in a file in the WIM. (Note: In the TLS_RSA SE it is already defined according to the TLS specification and should not be changed.)**

Operations:

MSE-Set	Is used to set attribute values in the CT.
PSO-Encipher	Compute a cryptogram using the predefined attributes that were set in the CT (eg. serverPublicKey) and the parameters sent with this command (plain value for example). The cryptogram is the pre-master secret and is set in the CCT template if the operation completed successfully

10.3.1.3 CCT

The CCT is used to implement the TLS PRF function [TLS10].

```
TLS_RSA_CCT {
   Transient   Key     preMasterSecret;
               Ref     masterSecretRef;
   Private     Key     masterSecret;
               Byte    ResultLength;
               Ref     algorithmRef;

   OPERATIONS:
       MSE-DeriveKey(Ref masterSecretRef, Byte[] inputData, …);
       PSO-ComputeChecksum(Byte[] inputData);
};
```

Attributes description:

`preMasterSecret`	Calculated and internally set in the WIM-KeyTransport primitive. It is then used in the `PSO-DeriveKey` operation to calculate the master secret. (master secrets belong to different sessions with remote servers and are stored in a private "master secret" file in the WIM).
`masterSecretRef`	The reference of the master secret.
`masterSecret`	The value of the master secret, stored privately by the module.
`resultLength`	The expected length of the result to calculate by the TLS PRF function
`algorithmRef`	The reference of the TLS PRF algorithm used in this template.

Operations:

MSE-DeriveKey	Derive a certain master secret based on a pre-master secret, available in the module.
PSO-ComputeChecksum:	Compute a cryptographic checksum (using the TLS PRF algorithm) based on a selected master secret and input data.

10.4 Generic Security Environments

10.4.1 WIM_GENERIC_RSA Security Environment

The WIM_GENERIC_RSA SE is being used for generic (eg, WAP application level operations).

In the WIM_GENERIC_RSA Security Environment there are two CRTs:

DST - For digital signatures

CT - For deciphering

10.4.1.1 DST

The Digital Signature template for the WIM_GENERIC_RSA SE has the same attributes as the WTLS_RSA SE, plus the GenerateAsymmetricKeyPair operation.

```
GENERIC_RSA_DST {
            Ref             algorithmRef;
            Ref             privateKeyRef;
Private     Key             privateKey;

            Key             verificationPublicKey;
Transient       Byte                    verificationDigest[];

OPERATIONS:
    MSE-Set(Ref privateKeyRef, …);
    PSO-ComputeDigitalSignature(byte[] stringToSign, …);

    MSE-Set(Key verificationPublicKey, byte[] verificationDigest, …);
    PSO-VerifySignature(byte[] Signature, …);


    GenerateAsymmetricKeyPair (byte[] AuthorisationCode,…);


    GenerateKeyAssurance(byte[] AuthorisationCode,…);




};
```

Attributes description:

`algorithmRef` The reference of the RSA algorithm used in this template. The reference of the algorithm is implicitly known. The algorithm is:

RSA in the WIM_GENERIC_RSA SE and
ECDSA or ECIES in WIM_GENERIC_ECC SE.

`privateKeyRef` The reference of the private key to use in the `PSO-ComputerDigitalSignature` operation and GenerateAsymmetricKeyPair. The private key file itself is being

	searched in the application context.
privateKey	The private key that is indicated with the privateKeyRef. The private key is searched and found in application context.
verificationPublicKey	The public key used for signature verification (ie, public key corresponding to the private key that was used for signing the verificationDigest).
verificationDigest	The digest used in signature verification.

Operations:

MSE-Set	Used to set attribute values in the DST.
PSO-Compute DigitalSignature	Compute a digital signature using the predefined attributes that were set in the DST and the parameters sent with this command.
PSO-VerifySignature	Verify a signature using the predefined attributes that were set (verificationPublicKey, verificationDigest) and the parameters sent with this command.
GenerateAsymmetricKeyPair	Generate a new key pair using the predefined attributes that were set in the DST and the parameters sent with this command
GenerateKeyAssurance	Compute a digital signature over information about a given WIM key pair in order to certify that it belongs to a WIM.

10.4.1.2 CT

```
WIM_GENERIC_RSA_CT {
              Ref         privateKeyRef;
   Private    Key         privateKey;
              Ref         algorithmRef;

   OPERATIONS:
           MSE-Set(Ref privateKeyRef, …);
           PSO-Decipher(byte[] wrappedKey, …);
};
```

Attributes description:

PrivateKeyRef	The reference of the private key to use in the PSO-Decipher operation. The private key file itself is being searched in the application context.

`PrivateKey`	Is the private key that is indicated with the `privateKeyRef`. The private key is searched and found in application context.
`AlgorithmRef`	Is the reference of the RSA algorithm used in this template. The algorithms that the WIM support are described in a file in the WIM. (Note: In the WIM_GENERIC_RSA SE it is already defined and should not be changed).

Operations:

`MSE-Set`	Is used to set attribute values in the CT.
`PSO-Decipher`	Decipher the cryptogram contained in `wrappedKey`.

10.4.2 WIM_GENERIC_ECC Security Environment

The WIM_GENERIC_ECC SE is being used for generic (eg, WAP application level operations). The CRTs and their functionality is the same as in WIM_GENERIC_RSA Security Environment, except that the corresponding ECC operations are used:

ECDSA for PSO-ComputeDigitalSignature and Generate Key Assurance

ECIES for PSO-Decipher. ECIES is specified in [P1363a]

11.Smart Card Implementation

This chapter describes how the WIM functionality is implemented on smart cards. For GSM, the card used for a WIM can be the SIM card [GSM11.11] or an external card.

11.1 Physical Characteristics

The physical characteristics of cards used for WIM MUST be in accordance with [ISO7816-1] and [ISO7816-2], and additional requirements of [GSM11.11], unless otherwise specified.

The card may be of type ID-1 or ID-000 (Plug-in).

11.2 Electronic Signals And Transmission Protocols

The electronic signals and transmission protocols of cards used for WIM MUST be in accordance with [ISO7816-3] and additional requirements of [GSM11.11], [GSM11.12] and [GSM11.18], if it is integrated with a GSM SIM, if applicable.

The electronic signals and transmission protocols of cards used for WIM MUST be in accordance with [ISO7816-3] and additional requirements of [TS102.221] if it is implemented on a UICC, if applicable.

The power consumption of the cards used for WIM MUST comply to [TS102.221].

11.2.1 Answer to Reset

Cards that comply to 7816 standard MUST send an ATR. The actual content of this ATR depends on whether the WIM application is alone in the card or shares the card with another application. See chapter 11.2.2 for SIM/WIM implementation and chapter 11.2.3 for WIM only implementation

11.2.1.1 Protocol

[ISO7816-3] specifies the default protocol as T=0. The ATR will include indication of T=0 protocol support because indication of voltage (T=15).

The ME MUST support T=0, and MAY support T=1. The WIM MUST support T=0 and MAY support T=1. If T=1 is supported by the WIM T=0 must be indicated as the first offered protocol.

11.2.1.2 Transfer Rate

[ISO7816-3] specifies the default transfer rate to 9600 bauds (@3.5712MHz). Higher rates can be negotiated between the WIM and the ME according to the Protocol Parameter Selection procedure (PPS) of the 7816-3 and to transfer rates specified in [GSM11.11]

The ME MUST be able to initiate a PPS procedure if the WIM indicates support of interface parameters and protocols in the ATR, but it is not required that the ME is able to increase the transfer rate.

11.2.1.3 Supply Voltage

As specified by the [ISO7816-3], the ATR MUST include indication of supply voltage (T=15). The ME MUST be able to handle this indication.

11.2.1.4 Logical Channels

The WIM indicates in the historical bytes of the ATR if it supports logical channels, as specified in the [ISO7816-4].

11.2.1.5 Clock Stop Mode

The WIM MUST support clock stop mode [ISO7816-3].

11.2.2 SIM/WIM implementation

If the WIM application shares a device with a SIM application, the SIM ATR is sent.

On the SIM/ME interface, the ME and the SIM/WIM MUST comply to [GSM11.11] for protocol selection, speed enhancement, and to [GSM 11.12] and [GSM11.18] for voltage selection.

The SIM/WIM MUST send voltage indication in the ATR (T=15).

The SIM/WIM device MUST support logical channels. It MUST send the logical channel indication in the historical bytes of the ATR

The ME MUST support logical channels.

The card MUST support 3 V and MAY support 5 V and / or 1.8 V.

The ME MUST support 3 V and MAY support 5 V and / or 1.8 V.

11.2.3 WIM Only or WIM with Other Applications

If the WIM wants to increase the default transfer rate it MUST be able to handle the PPS procedure as specified in [ISO7816-3].

The card MUST support 3 V, for optimal usage with phones. The card SHOULD support 5 V, in order to be able to work with readers that support only 5 V (it depends on the actual usage of the card how important it is to support 5 V).

The ME MUST support 3 V. It is anticipated that many ME card readers support also 5 V to be operable with application cards with only this voltage. However, the WIM implementors should not rely on 5 V being supported.

11.3 Description of Card Commands

This chapter describes card commands. The specification is based on [ISO7816-4] and [ISO7816-8].

The commands are described using Application Protocol Data Units (APDU) [ISO7816-4]. A command APDU consists of

- a mandatory header of four bytes: CLA (class byte), INS (instruction byte) and P1, P2 (parameter bytes)
- a conditional body of variable length: Lc (length of data field), Data field, Le (length of expected data)

A response APDU consists of

- a conditional body of variable length
- a mandatory trailer of two status bytes: SW1, SW2

The mapping between APDUs and TPDUs (Transmission Protocol Data Unit) [ISO7816-3] is performed according to [ISO7816-4].

Note that Le indicates the maximum length of data expected in response. If Le is greater than or equal to the actual number of bytes in the specific operation, the card returns the actual number of bytes. The value Le=0 indicates that the ME is expecting maximum 256 bytes in response and the card should return the actual number of bytes.

Table 1. Card commands

Operation	**CLA**	**INS**	**Reference**
Managing Logical Channel			
MANAGE CHANNEL	00	70	[ISO7816-4], [TS102.221]
Verification related operations			
VERIFY	0X / 8X	20	[ISO7816-4], [TS102.221][2]
ENABLE VERIFICATION REQUIREMENT	0X / 8X	28	[ISO7816-8], [TS102.221]
DISABLE VERIFICATION REQUIREMENT	0X / 8X	26	[ISO7816-8], [TS102.221] [3]
CHANGE REFERENCE DATA	0X / 8X	24	[ISO7816-8], [TS102.221]
RESET RETRY COUNTER	0X / 8X	2C	[ISO7816-8], [TS102.221]
Data storage related operations			
SELECT	0X / 8X	A4	[ISO7816-4], [TS102.221]
READ BINARY	0X / 8X	B0	[ISO7816-4], [TS102.221]
UPDATE BINARY	0X / 8X	D6	[ISO7816-4], [TS102.221]
Cryptographic operations			
MANAGE SECURITY ENVIRONMENT	0X / 8X	22	[ISO7816-8]
PERFORM SECURITY OPERATION	0X / 8X	2A	[ISO7816-8]

[2] In WIM SCP that command has been overloaded so that it can be used to retrieve the PIN status

[3] In WIM SCP the usage of that command to replace a PIN by a Universal PIN is not supported.

ASK RANDOM	8X	84	[ISO7816-4] (GET CHALLENGE)
OBKG & Key enrollment			
GENERATE ASYMMETRIC KEY PAIR	8X	46	[ISO7816-8]
GENERATE KEY ASSURANCE	8X	46	[ISO7816-8]
Other commands			
GET RESPONSE	0X	C0	[ISO7816-4], [TS102.221]

In the CLA byte, the X denotes the logical channel number.

For VERIFY, ENABLE VERIFICATION REQUIREMENT, DISABLE VERIFICATION REQUIREMENT, CHANGE REFERENCE DATA, RESET RETRY COUNTER, SELECT FILE, READ BINARY, UPDATE BINARY, two modes of operation are defined: native mode (CLA=8X) and SCP mode (CLA=0X). Native mode commands are defined in this specification in the subsequent chapters. SCP mode commands are as specified in [TS102.221], except if stated otherwise.

The specification makes it possible for card implementations to support both modes.

A card supporting SCP [TS102.221] MUST also support WIM in the SCP mode. Implementations are REQUIRED to support one of the modes and RECOMMENDED to support both modes. (Future versions may require support of the SCP mode and ease the requirement of supporting the native mode.)

A terminal supporting SCP [TS102.221] MUST also support WIM in the SCP mode. Implementations are REQUIRED to support one of the modes and RECOMMENDED to support both modes. (Future versions may require support of the SCP mode.)

See requirements for support of the modes in the Static Conformance Requirement.

For SELECT, CLA=0X is used of for selecting an application (direct application selection with AID), and CLA=8X for selecting a file in the native mode.

For the native mode, the ‘Reference’ column is informational only. The descriptions of the commands are included in this specification. For the SCP mode, the reference is normative.

For MANAGE SECURITY ENVIRONMENT and PERFORM SECURITY OPERATION, cards support CLA=8X. Additionally, they support CLA=0X for operations in generic security environments.

In addition to the instruction codes specified in cards command table, the following codes are reserved:

Administrative management phase: 'D0', 'D2', 'DE', 'C4', 'C6', 'C8', 'CA', 'CC', 'B4', 'B6', 'B8', 'BA' and 'BC'.

11.3.1 Mapping Service Primitives to Card Commands

The following table presents a mapping of WIM service primitives to the corresponding card commands and parameter names in the card commands for smart card implementation. The exact formats of the parameters are defined in the following sections. This mapping defines the unique smart card implementation at interface level that realizes the abstract service primitives of WIM.

Table 2. Mapping Service Primitives to Card Commands

Service Primitive	**Corresponding Card Commands**	**Reference**
Device Control Primitives		
WIM-OpenService	MANAGE CHANNEL Open SELECT Application	11.3.2.1, 11.4.1 11.3.3, 11.4.2
WIM-CloseService	MANAGE CHANNEL Close	11.3.2.2
Verification Related Primitives		
WIM-PerformVerification *Reference data ID* *Verification data*	VERIFY *Qualifier of the reference data (*)* *Verification data*	11.3.4.1
WIM-DisableVerificationRequirement *Reference data ID* *Verification data*	DISABLE VERIFICATION REQUIREMENT *Qualifier of the reference data (*)* *Verification data*	11.3.4.2
WIM-EnableVerificationRequirement *Reference data ID* *Verification data*	ENABLE VERIFICATION REQUIREMENT *Qualifier of the reference data (*)* *Verification data*	11.3.4.3
WIM-ChangeReferenceData *Reference data ID* *Verification data* *New reference data*	CHANGE REFERENCE DATA *Qualifier of the reference data (*)* *Verification data* *New reference data*	11.3.4.4

WIM-UnblockReferenceData *Reference data ID* *Unblock data* *New reference data*	RESET RETRY COUNTER *Qualifier of the reference data (*)* *Unblock data* *New reference data*	11.3.4.5
Data Access Primitives		
WIM-OpenFile *Path* *Status*	SELECT File *File ID* *FCI*	11.3.5.1
WIM-CloseFile	(not applicable)	
WIM-ReadBinary *Offset* *Length* *User data*	READ BINARY *Offset* *Length expected* *Data read*	11.3.5.2
WIM-UpdateBinary *Offset* *User data*	UPDATE BINARY *Offset* *Bytes to be written*	11.3.5.3
Cryptography Primitives (**)		
WIM-ComputeDigitalSignature *Private key ID* *User data* *Signature*	MSE Set (***) *filePath (tag 81), KPrivRef (tag 84)* PSO Compute Digital Signature *Data to be signed* *Digital signature*	11.3.6.8

WIM-VerifySignature *Public key* *Digest* *Signature*	MSE Set (***) *KPubCA (tag 83)* *Digest (tag 90)* PSO Verify Digital Signature *Signature*	11.3.6.9
WIM-GetRandom *Length* *Random*	ASK RANDOM *Length of random number* *Random number*	11.3.6.12
WIM-KeyTransport *Public key* *Additional data* *Transported key*	MSE Set (***) *KPubServer (tag 83)* *Additional data (security protocol version number) (tag 91)* PSO Encipher, Key Transport *Encrypted data*	11.3.6.5, 11.4.4
WIM-KeyAgreement *Private key ID* *Public key*	MSE Set (***) *filePath (tag 81), KPrivRef (tag 84)* *KPubServer (tag 83)* PSO Encipher, Key Agreement	11.3.6.6, 11.4.5
WIM-DeriveMasterSecret *Input data* *Master secret ID*	MSE Derive Key *Seed (tag 94)* *SecretKeyRef (tag 84)*	11.3.6.11
WIM-PHash *Master secret ID* *Input data*	MSE Set (***) *MasterSecretRef (tag 83)* *Output length (tag 96)* PSO Compute Cryptographic Checksum *Data*	11.3.6.9, 11.4.4

Block	*Cryptographic checksum*	
WIM-Decipher *Private key ID* *Enciphered data* *Data*	MSE Set (***) *filePath (tag 81), KPrivRef (tag 84)* PSO Decipher *Data (cryptogram)* *Deciphered data*	11.3.6.7, 11.4.7
WIM-GenerateAsymmetricKeyPair *PrivateKeyID* *Authetication Data* *Encrypted PIN value* *Encrypted PIN value* *Encrypted private & public key label*	MSE Set *filePath (tag 81), KPrivRef (tag 84)* GENERATE ASYMMETRIC KEY PAIR *Authetication Data* (9E or 8E) *Encrypted PIN value (tag C0)* *Encrypted PIN label (tag C1),* *Encrypted private&public key label (tag C2),*	11.3.6.13
WIM-GenerateKeyAssurance *PrivateKeyID* *Authentication Data*	MSE Set *filePath (tag 81), KPrivRef (tag 84)* GENERATE KEY ASSURANCE *AuthenticationData* (9E or 8E)	11.3.6.14
Exceptions		
WIM-Exception	(exceptions may occur with all commands)	

(*) Verification related commands may be preceded by a SELECT File command, in order to select the proper PIN file indicated in the corresponding AODF entry.

(**) Before using PERFORM SECURITY OPERATION (PSO) or MANAGE SECURITY ENVIRONMENT (MSE) Set commands, a MSE Restore command must be issued.

(***) For PERFORM SECURITY OPERATION (PSO) commands, some parameters are set by preceding MANAGE SECURITY ENVIRONMENT (MSE) command(s). Note that once set, the parameters are memorised by the corresponding security environment template as long as the same security environment is used (ie, the logical channel is open and no security environment is restored).

For file access, a single WIM-OpenFile operation may map to several SELECT commands, depending on the semantics and value of the *Path* parameter. ReadBinary or UpdateBinary may map to several corresponding commands, depending on the semantics and the *Length* parameter.

11.3.2 Managing Logical Channel

A logical channel [ISO7816-4] is a link to a card application context.

Command interdependencies on one logical channel are independent of command interdependencies on another logical channel. However, when opened, a logical channel may inherit context information from another logical channel.

Commands referring to a certain logical channel carry the respective logical channel number in the two least significant bits of the CLA byte. Logical channels are numbered from 0 to 3. The basic logical channel (number 0) is permanently available.

There is no interleaving of commands and their responses across logical channels; between the receipt of the command APDU and the sending of the response APDU to that command only one logical channel is active. (This means that for T=0, the ME MUST send the GET RESPONSE command before starting an APDU in another logical channel. Otherwise, the response is lost.)

A logical channel is opened using a MANAGE CHANNEL command, in which the card assigns a channel number and returns it in the response. It remains open until explicitly closed by a MANAGE CHANNEL command.

11.3.2.1 MANAGE CHANNEL Open

Description

This command opens a logical channel, other than the basic one. The card assigns a channel number and returns it in the response. It remains open until explicitly closed by a MANAGE CHANNEL Close command.

Command APDU

CLA	00
INS	70
P1	00
P2	00
Lc	**Empty**
Data	**Empty**
Le	1

Response APDU

Data	Assigned logical channel number
SW1-SW2	Status bytes

11.3.2.2 MANAGE CHANNEL Close

Description

This command closes a logical channel. Note that the basic channel (number 0) cannot be closed.

Command APDU

CLA	00
INS	70
P1	80
P2	01, 02 or 03 (corresponding the logical channel number)
Lc	**Empty**
Data	**Empty**
Le	Empty

Response APDU

Data	Empty
SW1-SW2	Status bytes

11.3.3 Application selection

The WIM application may have to reside on the card with other applications, eg, GSM. It is selected using an Application Identifier (AID) which is a combination of a Registered Application Provider Identifier (RID) and a Proprietary Application Identifier Extension (PIX) [ISO7816-5].

The WIM application has typically an own Dedicated File (DF) which is indicated in the application selection process.

The card MUST support direct application selection (using the full AID as a parameter for the SELECT command). The ME MUST support direct application selection.

In addition, the card MAY have an EF(DIR) file, listing the applications present in the card. The DIR file entry describing the WIM application SHOULD be formatted according to [PKCS15]. The ME is not required to support the 'discretionary data objects' field.

In the case where the WIM application is the only PKCS#15 application in the card, the WIM application is selected using the PKCS#15 AID [PKCS15]. Otherwise, ie, when besides the WIM, there are other PKCS#15 applications that do not confirm to the WIM specification, the WIM application is selected using the WIM specific AID.

The WIM AID is defined as follows. The RID for the WIM AID is the same as defined in [PKCS15], ie, A0 00 00 00 63. The PIX is "WAP-WIM". The full AID for the current version of this recommendation is thus:

A0 00 00 00 63 57 41 50 2D 57 49 4D

The procedure for the ME is the following:

1) Execute SELECT with the PKCS#15 AID (complete AID as specified in [PKCS15])

2) Read the Security Environment information and the label field in EF(TokenInfo) (`tokenInfo.label`)

3) If (2) fails (reading fails or there is no Security Environment information required for the WIM or the `tokenInfo.label` has an inappropriate value) and direct application selection (step (1)) succeeded, execute SELECT with the WIM AID.

The field `tokenInfo.label` indicates the application, as defined in 9.4.7.

After selecting the application, the current DF is the PKCS#15 (WIM) DF.

Once the application has been selected in a channel (other than zero), the ME SHOULD use this channel and close it before selecting another application in this channel.

11.3.3.1 SELECT Application, Direct Method

Description

A successful SELECT Application sets the current application, using an Application Identifier (AID).

Command APDU

CLA	0X
INS	A4
P1	04
P2	00, 0C or 04
Lc	**Length of the Application Identifier (AID) (only value 0C is allowed)**
Data	**AID**
Le	Empty

Response APDU

Data	Empty or FCP
SW1-SW2	Status bytes

The value of P2 indicates the contents of 'Data' in the response

- '00', no response (in compliance with earlier versions of this specification)
- '0C', no response
- '04', return FCP

The FCP contains the following parameters with corresponding tags:

- '81' Total file size
- '82' File Descriptor
- '84' DF name (AID)
- 'A5' Proprietary, with the following parameters
  - '81' Power consumption, the maximum value being 10 mA (encoded as '0A') at any of the defined voltage classes, at a reference frequency 3 MHz
  - '82' Minimum application clock frequency, the value being 3 MHz (encoded as '1E')
- '8A' Life cycle status integer
- '8B' Security attributes
- 'C6' PIN status template DO

11.3.4 Verification Related Operations

The verification process is based on storing in the card a reference data (PIN). In order the user to have access to certain function of the card, he must be able to present verification data that is checked by the card to match the reference data.

The reference data has a fixed length, indicated in AODF. A shorter value entered by the user MUST be padded to the full length.

The commands defined here are meant to be used for a WIM application specific reference data. The global (ie, used for the whole card) reference data should be handled according to relevant specifications.

The AODF entry, corresponding to the reference data, indicates the qualifier of the reference data (used as P2).

11.3.4.1 VERIFY

Description

This command initiates the comparison in the card of the verification data sent from the ME, with the reference data stored in the card.

The security status may be modified as a result of the comparison. Unsuccessful comparisons MUST be recorded by the card (eg, to limit the number of further attempts of the use of the reference data).

Command APDU

CLA	As defined in 11.3
INS	20
P1	00
P2	Qualifier of the reference data
Lc	**LreferenceData**
Data	**Verification data**
Le	Empty

Response APDU

Data	Empty
SW1-SW2	Status bytes

With an empty body, this command may be used, both in native and in SCP mode, to check whether

- the verification is not required (SW1-SW2 = ‘9000’), eg, if the verification requirement has been disabled
- verification is required (SW1-SW2 = ‘6300’)
- verification is required, with indication of remaining tries (SW1-SW2 = ‘63CX’)
- PIN is blocked (SW1-SW2 = ‘6983’ or ‘63C0’).

11.3.4.2 DISABLE VERIFICATION REQUIREMENT

Description

This command is used to disable the verification requirement.

If the PIN verification fails:

- the PIN try counter is updated
- the PIN remains enabled
- the PIN right remains granted;

when the PIN is blocked:

- the PIN remains enabled
- the PIN right is lost;

so, the PIN has to be unblocked using RESET RETRY COUNTER command.

Note: Above, the order is not significant, only the result is.

Command APDU

CLA	As defined in 11.3
INS	26
P1	00
P2	Qualifier of the reference data
Lc	**LreferenceData**
Data	**Verification data**
Le	Empty

Response APDU

Data	Empty
SW1-SW2	Status bytes

Remark: in WIM SCP mode the use of that command to switch to a Universal PIN is not supported.

11.3.4.3 ENABLE VERIFICATION REQUIREMENT

Description

This command is used to enable the verification requirement.

If the PIN verification fails:

- the PIN try counter is updated
- the PIN remains disabled
- the PIN right remains granted;

when the PIN is blocked:

- the PIN remains disabled
- the PIN right remains granted.

Note: Above, the order is not significant, only the result is.

Command APDU

CLA	As defined in 11.3
INS	28
P1	00
P2	Qualifier of the reference data
Lc	**LReferenceData**
Data	**Verification data**
Le	Empty

Response APDU

Data	Empty
SW1-SW2	Status bytes

11.3.4.4 CHANGE REFERENCE DATA

Description

This command is used to initiate the comparison of the verification data with the reference data, and then to conditionally replace the existing reference data with new reference data sent in the command. The verification data length and the new reference data length are both fixed by the PinAttributes.storedLength field of the corresponding AODF record.

Command APDU

CLA	As defined in 11.3
INS	24
P1	00
P2	Qualifier of the reference data
Lc	**2 * LReferenceData**
Data	**Verification data + New reference data (concatenation)**
Le	Empty

Response APDU

Data	Empty
SW1-SW2	Status bytes

11.3.4.5 RESET RETRY COUNTER

Description

This command is used to change the reference data on completion of a successful reset of the reference data retry counter to its initial value.

Command APDU

CLA	As defined in 11.3
INS	2C
P1	00
P2	Qualifier of the reference data
Lc	**LReferenceData + LUnblockData (total value)**
Data	**Unblock data + New reference data (concatenation)**
Le	Empty

Response APDU

Data	Empty
SW1-SW2	Status bytes

11.3.5 Operations Related to Data Storage

Data storage operation defines access to unformatted (binary type) and formatted (record based) files. Access to formatted files is not defined in this version of the specification.

11.3.5.1 SELECT FILE

Description

A successful SELECT sets a current file.

Command APDU

CLA	As defined in 11.3
INS	A4
P1	00
P2	00
Lc	**02**
Data	**File ID**
Le	Empty, or 00

Response APDU

Data	Empty, or the File Control Information (FCI), see below
SW1-SW2	Status bytes

For EF, the FCI, returned by this command contains at least the file size data object as described below. For transparent EF it is the number of allocated bytes of the body part (ie, excluding structural information).

The FCI shall contain at least the following data object:

Byte(s)	**Value**
1	80
2	02
3-4	File size (higher byte first)

For DF and MF, the previous data object shall not be included in the FCI, but other data objects could be provided.

After both EF and DF selection, the ME shall ignore unexpected data objects if provided by the WIM.

11.3.5.2 READ BINARY

Description

This command is used to read (a portion of) a file.

Command APDU

CLA	As defined in 11.3
INS	B0
P1	High byte of the offset (0..7F).
P2	Low byte of the offset.
Lc	**Empty**
Data	**Empty**
Le	Number of bytes to be read

Response APDU

Data	Data read (Le bytes)
SW1-SW2	Status bytes

11.3.5.3 UPDATE BINARY

Description

This command is used to update (a portion of) a file with a string of bytes. This command in only used to replace existing bytes.

Command APDU

CLA	As defined in 11.3
INS	D6
P1	High byte of the offset (0..7F)
P2	Low byte of the offset
Lc	**Number of bytes to be written**
Data	**Bytes to be written**
Le	Empty

Response APDU

Data	**Empty**
SW1-SW2	Status bytes

11.3.6 Cryptographic Operations

The following APDU commands implement cryptographic operations. Before issuing any of these APDU commands a VERIFY should be issued, after selecting the proper DF, otherwise these commands will be rejected.

The following table specifies the tags of Data Objects that can be sent as parameters in the different APDU commands.

Table 3. Tags for Data Objects

Tag	Value
80	Plain value (non BER-TLV coded data)
86	Padding indicator byte followed by cryptogram (plain value not coded in BER-TLV). The padding indicator byte is '00' for asymmetric key operations.
8E	Cryptographic checksum
9E	Digital signature
9A	Input for Digital signature (non BER-TLV coded data)
96	Value of Le
A8	Input template for DS verification

The following table specifies the tags of Control Reference Data Objects (attributes in CRTs).

Table 4. Control Reference Data Objects

Tag	Value	Related CRT
4D	L≠0, extended headerlist of DOs (defines the order and the data items which form the input for the security operations)	B6 (DST)
81	File path (used for indicating the private key file path)	B6 (DST) B8 (CT Asym)
83	Key reference of a public key in asymmetric cases	B6 (DST) B8 (CT Asym)
83	Key reference for direct use in symmetric cases	B4 (CCT)
84	Key reference for computing a session key in symmetric cases	B4 (CCT)
84	Key reference of a private key in asymmetric cases	B6 (DST)

		B8 (CT Asym)
90	Hash code	B6 (DST)
91	L=0; random number provided by the card	B8 (CT Asym)
94	Challenge or data item for deriving a key	B8 (CT Sym)
96	Length of the result to be returned by the PSO-ComputeCryptographicChecksum	B4 (CCT)

Data objects are sent BER-TLV encoded, according to [ISO7816-4], Annex D. In particular, length from 0 to 127 is encoded with a single byte, and length from 128 to 256 is encoded with two bytes. Eg, length 135 is encoded as '81 87'.

11.3.6.1 MANAGE SECURITY ENVIRONMENT

In order to use Security Environments we need to have commands for selecting a given SE and commands for setting non private attributes in its CRTs. We set attributes in a given CRT before issuing a specific calculation command. For example we will set the public key in a CT before issuing an encipher command using this key. We can also select the private key reference (reference in an internal file) in a DST before issuing a Compute Digital Signature command that will use this specified key.

This command can be used in a particular case for deriving a key, see DERIVE KEY Command.

11.3.6.2 MSE - RESTORE

Description

This operation is used to RESTORE a SE by replacing the current SE with the SE number mentioned in this function

Command APDU

CLA	As defined in 11.3
INS	22
P1	F3
P2	SE number
Lc	Empty
Data	Empty
Le	Empty

Response APDU

Data	Empty
SW1-SW2	Status bytes

11.3.6.3 MSE - SET

Description
This operation is used to SET one or several components of the current SE

Command APDU

CLA	As defined in 11.3
INS	22
P1	41 for computation (signing) and deciphering, 81 for enciphering and verification
P2	Tag of the related CRT • B6 (Digital Signature Template) • B8 (Confidentiality Template) • B4 (Cryptographic Checksum Template)
Lc	Length of the Data field
Data	Concatenation (in any order) of CRDOs (defined in Table 4. Control Reference Data Objects).
Le	Empty

Response APDU

Data	Empty
SW1-SW2	Status bytes

11.3.6.4 PERFORM SECURITY OPERATION

The PERFORM SECURITY OPERATION command implements all security related APDU commands. We first describe the general structure of this command and then show, in each sub-section, how it implements different security commands.

Description

This command initiates a set of cryptographic operations such as computation of a digital signature, calculation of a cryptographic checksum, encipherment, decipherment etc. These security operations are related to the Data Objects specified in the parameters P1 and P2 (see below the different operations).

Conditions of use

This command may be preceded by a MANAGE SECURITY ENVIRONMENT command in order to set in a specific CRT the key and algorithm reference if not implicitly known with the current SE. The command may be performed in one or several steps, possibly using the command chaining function.

Command APDU

CLA	As defined in 11.3
INS	2A
P1	Defines the output • tag of the DO for output data (see Table 3. Tags for Data Objects) • 00, if output empty • FF, RFU
P2	Defines the input • tag of the DO for input data (see Table 3. Tags for Data Objects) • 00, if input empty • FF, RFU
Lc	**Length of the Data field**
Data	**Value of DO specified in the parameter P2**
Le	Empty or maximum length of the data expected in response

For response data and status word, see the relevant clause under each operations.

11.3.6.5 PSO - ENCIPHER, Key Transport

Description

In the context of the WTLS_RSA and TLS_RSA Security Environments the PSO-ENCIPHER operation is used to encipher the transported data (key). The original data is set by a preceding MSE command.

Conditions of use

The algorithm and key reference are defined in the current SE under the CT (Confidentiality Template) context.

Command APDU

CLA	As defined in 11.3
INS	2A
P1	86 (padding indicator byte followed by cryptogram)
P2	00 (no input)
Lc	**Empty**
Data	**Empty**
Le	Length of the encrypted data expected in response

Response APDU

Data	Padding indicator byte '00' + RSA encrypted data. PKCS#1 block type 2 is used
SW1-SW2	Status bytes

11.3.6.6 PSO - ENCIPHER, Key Agreement

Description

In the context of the WTLS_ECDH Security Environment the PSO-ENCIPHER operation is used to implement the Diffie-Hellman key agreement. The public key of the other party and ID of own private key are set by a preceding MSE command.

Conditions of use

The algorithm and key reference are defined in the current SE under the CT (Confidentiality Template) context.

Command APDU

CLA	As defined in 11.3
INS	2A (PSO)
P1	86 (cryptogram)
P2	00 (no input)
Lc	**Empty**

Data	**Empty**
Le	Empty

Response APDU

Data	Empty
SW1-SW2	Status bytes

11.3.6.7 PSO - DECIPHER, Application Level

Description

This operation is used to decipher a message key with a private key.

In case of RSA, the cryptogram is an RSA encrypted PKCS#1 block type 2. The card performs decryption with the private key, parses that PKCS#1 block and returns the actual data.

Conditions of use

The algorithm and key reference of the private key used in decryption, are defined in the current SE under the CT (Confidentiality Template) context.

Command APDU

CLA	As defined in 11.3
INS	2A
P1	80 (plain value)
P2	86 (padding indicator byte followed by cryptogram)
Lc	**Length of cryptogram**
Data	**Padding indicator byte '00' + data to be deciphered (encrypted message key)**
Le	Length of the data expected in response

Response APDU

Data	Deciphered data (message key)
SW1-SW2	Status bytes

11.3.6.8 PSO - COMPUTE DIGITAL SIGNATURE

Description

This operation is used to compute a digital signature or initiate the computation. The data to be signed is transmitted in the command data field.

For RSA, signing is performed according to [RFC2313], using block type 1. The signature is returned as an octet string.

For ECDSA, signing is performed according to [X9.62]. The signature is returned as the concatenation of *r* and *s*, each represented most significant byte first using the same number of octets as are required to represent *n* (the order of the base point).

Conditions of use

The algorithm and key reference are defined in the current SE under the DST context (Digital Signature Template), if not implicitly known.

Command APDU

CLA	As defined in 11.3
INS	2A (PSO)
P1	9E (digital signature)
P2	9A (input for digital signature)
Lc	**Length of data to be signed**
Data	**Data to be signed. In the case of WTLS, this is the 20-byte SHA-1 hash of handshake messages. In the case of TLS, this is the concatenation of MD5 and SHA-1 hashes of handshake messages. For application level RSA signatures, it is a DER encoded DigestInfo structure. The WIM should treat this data as opaque, and do PKCS#1 padding in any case.** **For application level ECDSA signatures, it is the SHA-1 hash.**
Le	Length of digital signature.

Response APDU

Data	The resulting digital signature
SW1-SW2	Status bytes

11.3.6.9 PSO - VERIFY DIGITAL SIGNATURE

Description

This operation is used to verify a digital signature. The signature to be verified is transmitted in the command data field. The parameters digest and public key are set by a preceding MSE-SET command.

For RSA, verification is performed according to [RFC2313], using block type 1. However, no DigestInfo decoding is done by the operation. The operation compares the digest that was set in a previous MSE-Set with the digest data that is extracted after removing the padding bytes. The operation succeeds if all digest bytes match.

For ECDSA, verification is performed according to [X9.62].

The successful verification is indicated by the status bytes SW1SW2 0x9000. A failed verification is indicated by the status bytes SW1SW2 0x6A80.

Conditions of use

The algorithm is defined in the current SE under the DST context (Digital Signature Template), if not implicitly known.

Command APDU

CLA	As defined in 11.3
INS	2A (PSO)
P1	00 (output empty)
P2	A8 (digital signature)
Lc	**Length of data**
Data	**9E L Signature**
Le	Empty

Response APDU

Data	Empty
SW1-SW2	Status bytes

11.3.6.10 PSO - COMPUTE CRYPTOGRAPHIC CHECKSUM

Description

This operation is used to compute a cryptographic checksum.

Conditions of use

The algorithm and key reference are already defined in the SE under the CCT (Cryptographic Checksum Template) and correspond to the PRF for the calculating a key block.

Command APDU

CLA	As defined in 11.3
INS	2A (PSO)
P1	8E (cryptographic checksum)
P2	80 (plain value)
Lc	**Length of data.**
Data	**Data to be included in the cryptographic checksum**
Le	Length of the cryptographic checksum

Note. The input data correspond here to the seed data in the PRF and the cryptographic checksum '8E' to the generated data (eg, key block).

Response APDU

Data	The resulting data.
SW1-SW2	Status bytes

11.3.6.11 MSE - DERIVE KEY

Description

This operation is used for deriving a key through the usage of the MANAGE SECURITY ENVIRONMENT command.

Conditions of Use

Command APDU

CLA	As defined in 11.3
INS	22
P1	41 (MSE SET)
P2	B4 (Cryptographic Checksum Template)
Lc	Length of the Data field.
Data	**Concatenation of CRDOs:** 84 01 SecretKeyRef + 94 SeedLength Seed SecretKeyRef identifies the resulting master secret. Note that both parameters are mandatory, but the order is not significant. Key derivation process is initiated when both these parameters are set. Master secret key references outside of the EF(Sessions) record range MUST be rejected by the WIM.
Le	Empty

Response APDU

Data	Empty
SW1-SW2	Status bytes

11.3.6.12 ASK RANDOM

Description

This instruction allows the external world to ask the card for a random number. This random number will be deleted from the card after being used once. (This APDU is the Get Challenge APDU defined in ISO 7816-4.)

Conditions of use

This instruction can be executed anytime. No specific security environment is required.

Command APDU

CLA	8X
INS	84
P1	00
P2	00
Lc	Empty
Data	Empty
Le	Length of random number

Response APDU

Data	Random number
SW1-SW2	Status bytes

11.3.6.13 GENERATE PUBLIC KEY PAIR

Description

This operation is used to generate a new key pair in the WIM.

Conditions of use

The algorithm and key reference are already defined in the SE under the DST (Digital Signature Template), if not implicitly known.

A MSE-Restore command must have selected the WIM_GENERIC_RSA or WIM_GENERIC_ECC Security Environment.

A MSE-SET command must have set the current key and the current template.

If required, the authentication data shall be correct.

Command APDU

CLA	8X
INS	46
P1	00 –this is the first APDU that starts the key generation 04 –this is a successive iteration of the same key generation process
P2	00
Lc	Length of data
Data	00 [{9E L authentication data (digital signature) or 8E L authentication data (HMAC)} [C0 L encrypted New PIN value] [C1 L encrypted New PIN label] [C2 L encrypted new private & public key label]] [C3 L User PIN Value] Note: the parameters between [and] are considered optional. The order of the parameters is not important. Proprietary Tags C0 to CF are reserved for WIM usage. A 00 padding byte MUST always be present, with or without parameters.
Le	Length of expected data

All parameters are sent only in the in first APDU command (P1 = 00) and MUST NOT be sent in the successive iterations (P1 = 04).

The parameters “authentication data”, “encrypted New PIN value”, “encrypted New PIN label” and “encrypted new private & public key label” are optional. However, “encrypted New PIN value”, “encrypted New PIN label” and “encrypted new private & public key label ” MUST NOT be sent if “authentication data” is not included in the parameter list.

The Authentication data parameter is the HMAC, or digital signature, calculated over “encrypted New PIN value”, “encrypted New PIN label”, “encrypted new private & public key label” and the random number that was sent out as a challenge in the previous invocation of the command. This ensures integrity check for these parameters as well as authentication.

Rem: in verifying the HMAC, or digital signature, the sent parameters are concatenated as in the sent order including tags and lengths.

The encrypted parameters (New PIN Value, New PIN label and new private & public key label) are encrypted using 3DES in CBC mode as described in section 15.2 of [NIST SP 800-38A 2001 ED]. The initial chaining value for CBC modes shall be zero and padding M2 [ISO 9797] shall apply. The encryption key SHOULD be different from the authentication key used for gaining the right to generate the key pair.

The parameter "User PIN Value" is optional and is used for updating the value of the PIN that protects the key that is generated. It will be taken into account by the command only if the two following conditions are fulfilled:

- The current PIN that protects the key that needs to be generated is NOT initialized
- The "encrypted New PIN value" is not sent in the parameter list

If both the "User PIN Value" and "encrypted New PIN value" are sent in the parameter list the "encrypted New PIN value" will be used for updating the PIN that protects the key that is generated, and the "User PIN Value" will be ignored.

Processing of the command

The operation generates a new key pair and stores the private key in the private key slot identified by the key reference set in the DST template of the selected Generic Security Environment.

RSA keys MUST be generated in the WIM_GENERIC_RSA Security Environment and ECC key MUST be generated in the WIM_GENERIC_ECC Security Environment.

PrKDF entries for non-initialized keys MUST exist in the WIM with the 20 bytes `classAttributes.iD` all set to zero i.e. the on-board key generation function will update these entries but will not allocate any new entries.

If a PuKDF exist and the key is not initialized the PuKDF entry MUST indicate that the public key length is zero.

Different policies may be applied to each key slot at the discretion of the WIM issuer e.g. number of times that the key can be generated, authentication data required / not required etc. The authentication key (and algorithm) used to compute the authentication data might differ from one key slot to another one.

The command perform the following operations:

1. If authentication is required, verify the sent "authentication data" (digital signature or HMAC), by using the authentication key (and algorithm), and the value of the random number that was sent out as a challenge in the previous invocation of the command. If authentication fails, or no authentication data was sent, then return a new 20 bytes long random value (to be used for the next call), and the WIM serial number.
2. Generate a key pair
3. Store the private key in the private key slot identified by the key reference set in the DST template of the selected Generic Security Environment.
4. If PuKDF is present, update the length of the public key in the PuKDF and store the public key in the corresponding public key slot. If PuKDF is not present, store the public key internally for later use, e.g. in a key registration process.
5. If authentication was needed in step 1, then invalidate the current authentication data i.e. a fresh random number has to be generated before the key generation function can be called again.
6. If "encrypted PIN value" parameter is present, decrypt the PIN value by using the encryption key (and algorithm) associated to the key slot then update the PIN value that protects the generated key with the new PIN value.
7. If "encrypted PIN label" parameter is present, decrypt the label value by using the encryption key (and algorithm) associated to the key slot then update the corresponding label in AODF (label of the PIN that protects the generated private key) with the new label.

8. If "encrypted private & public key label" parameter is present, decrypt the label value by using the encryption key (and algorithm) associated to the key slot then update the corresponding labels in PrKDF (label of the generated private key). If PuKDf exist, update the label of the generated public key with the new label.

9. If "User PIN Value" parameter is present, update the corresponding PIN value if and only if the conditions that where stated above are satisfied.

10. Compute and update the public key hash values in the PrKDF and PuKDF entry (if present). This update should be the last step in the key generation function, in order to guarantee a consistent internal state.

Response APDU

Data	0x90 <Len> <SHA-1 public key hash> or C3 <Len> <20 bytes random number> C4 <Len> <WIM serial number>
SW1-SW2	Status bytes

In order to conform to GSM constraints, by which an APDU command processing time SHOULD NOT exceed 2 seconds, the ME MUST send the command iteratively (in a loop) until the operation is completed. The first APDU command MUST have P1 = 00, and for each successive iteration the sent APDU command MUST have P1 = 04 to indicate that it is a successive iteration of the same operation.

When the key generation process has successfully finished the APDU command returns the generated public key hash and indicates that the processing of the command is completed (61 XX)[4].

The APDU command returns 6200 (warning, no information given) to indicate that the iteration succeeded but that the key generation process is not completed.

If authentication fails, or no authentication data was sent the command returns the following data:

C3<Len><20 bytes random number> || C4<Len><WIM serial number> and indicates that the processing of the command is completed (61 XX)[3]. The WIM serial number MUST not exceed 64 bytes.

The ME MUST not send a new APDU command that starts a key generation if an already running key generation process has not finished.

[4] If the low level protocol is T=0 the data is retrieved with the Get Response TPDU

11.3.6.14 GENERATE KEY ASSURANCE

Description

This command is used to output signed Assurance information about the key pair currently selected in the WIM_GENERIC_RSA or WIM_GENERIC_ECC Security Environments.

Any initialized keys listed in PrKDF files can be selected, even though they are not used in GENERIC Security Environments

Assurance information is also returned signed with the issuer's Assurance key, therefore providing the proof of origin of the currently selected key pair.

This command can be executed as many times as wanted in order to allow a given key to be enrolled in many PKI.
The WIM issuer though may restrict the number of times this command is allowed to be executed.
This command may be subject to an authorization code.

Provided that authorization code is correct or not required, this command performs the actions listed in the "command processing" section below

Assurance Information

The **`AssuranceInfo`** is the structure that is signed or MACed. It is the data on which the assertion is made.
As per genEnrollReq function [ESMPCrypto], the **`AssuranceInfo`** has the following structure:

```
AssuranceInfo::= SEQUENCE {
        deviceID                    UTF8String (SIZE (1..64)) OPTIONAL,
        subjectPKInfo               SubjectPublicKeyInfo,
        keyAssertion                KeyAssertion,
        … -- For future use
    }

    KeyAssertion ::= ENUMERATED {
        injected-into-device (0), generated-on-device(1),
        -- Note: '(0)' and '(1)' are not needed but can be here for clarity
        ...
     }
```

The value of **`deviceID`** identifies the WIM in which the key was generated as an UTF8String. If it is a WIM, the **deviceID** is `the serial number in the EF(TokenInfo) file`.

`SubjectPKInfo` is the public key as defined in [PKCS10].

The **`KeyAssertion`** field is used to indicate if the key was generated on the WIM or if the key was injected into the WIM.

Assurance key

The Assurance key is a WIM issuer or a WIM manufacturer signature key. It is implicitly known by the WIM.

This key can either be a symmetric HMAC key (128bits minimum) or an asymmetric RSA key (1024bit minimum).

When end user keys are injected during WIM personalisation, the assurance signature may also be generated at the time of manufacture. In this case, it is not required that the assurance key resides on the WIM.

For on-board generated keys this signature can only be generated after the key is generated.
The WIM may elect to store the result in non-volatile memory for future use during one (or more) enrolments. For on-board generated keys the assurance key MUST reside on the device.

It is recommended that the assurance keys be generated on-board during the manufacturing stage, however there may be situations where adequate security measures during manufacturing can ensure an equivalent level of security. A unique assurance key SHOULD be used for each device

Command processing

Command processes AssuranceInfo and Assurance key as follow:

- If the Assurance key is an RSA key
  - The WIM applies a PKCS1 padding type 2 to the hash of the AssuranceInfo, formats both as a DER encoded structure and finally signs this structure using the Assurance private key.
  - The AssuranceInfo, the Assurance signature and the Assurance certificate hash is returned
- If the Assurance key is a symmetric key, the WIM perform CMS compliant operations [RFC 2630]
  - The WIM generates a symmetric 128bit session key (eg. a random)
  - The WIM signs the AssuranceInfo hash using HMAC-SHA-1 and the previously generated session key
  - The WIM uses the Assurance key to encrypts the session key. The encryption algorithm used is 3DES in CBC mode (with IV=0) as per [NIST SP 800-38A 2001 ED] and uses padding M2 [ISO 9797]. .
  - The AssuranceInfo, the ciphered session key and the Assurance signature is returned.

After successful execution, the command MUST return the data elements of Table 1 and MUST return either Table 2 or Table 3 data elements depending on the Assurance key type (symmetric or asymmetric). The command returns 90 00 to indicate success.

Note that when T=0 protocol is used, the command returns a 61xx protocol status and the actual data is retrieved via a GetResponse command.

Tag	Length	Description	Specification
C5	Var	AssuranceInfo structure	[WIM]

Table 1

Tag	Length	Description	Specification
C6	Var	Ciphered Session Key (Encrypted KEK)	[WIM] [CMS]
8E	Var	Symmetric Assurance Signature	[7816-4] Cryptogram Checksum

Table 2

Tag	Length	Description	Specification
9E	Var	Signature using asymmetric Assurance key	[7816-8] Digital Signature id

90	Var	Assurance Certificate hash	[7816-4] Hash code

Table 3

After non-successful execution, the command returns one of the following error codes:

'6A88' - Key reference not found (e.g. no key currently selected or key does not exist).

'6983' - Authentication method blocked. (Generate Key Assurance Command cannot be used anymore)

'9000' – Response data available see data elements in Table 4:

Tag	Length	Description	Specification
C3	Var	Random value	[WIM]
C4	Var [1..64 bytes]	WIM serial number	[WIM]

Table 4

If an authorization code is required, this command behave like the GenerateAsymmetricKeyPair command:
The authorization to invoke this command may be different for each key. This means that the internal key that is used to validate the sent authentication data may be different for each key or even apply a different algorithm. It also means that an authentication data may be needed for a certain key slot but may not be needed for another key slot. The command can be sent as many time as needed provided that authorisation is granted in each invocation. It is assumed that the entity that is authorized to send the command, for a given key id, knows how to generate the needed authentication data (the key and algorithms to use). The WIM issuer, who implements the authorisation scheme for each key slot, can communicate the authorisation scheme to authorized entities. All the above protection schemes are set during the WIM personalisation phase.
If the authentication fails, the Generate Key Assurance command returns "9000" with a new Challenge and the WIM serial number.
If the authentication method is blocked after a given number of times (defined by the card manufacturer or card issuer) the GenerateKeyAssurance command returns "6983".

Condition of use

A MSE-Restore command must have selected the WIM_GENERIC_RSA or WIM_GENERIC_ECC Security Environment.

A MSE-SET command must have set the current key and the current template.

If required, the authentication code shall be correct.

Command APDU

CLA	8X
INS	46
P1	01 : Public key in response data field
P2	00 : Implicit reference to assurance key
Lc	Length of the Authorisation data

Data	00 [9E L Digital signature Authentication code or 8E L HMAC Authorisation code]
Le	Length of data in response APDU

(The parameters between [and] are considered optional. The order of the parameters is not important)

Note: 00 is a padding byte must always be present.

Note that TLV length value shall be coded on 2 bytes if length exceeds 7F (eg L = 7F and L= 81 80)

Response APDU

Data	Empty or See above Tables 1, to 4
SW1-SW2	Status bytes

11.3.6.15 Other Commands

11.3.6.16 GET RESPONSE

Description

This instruction is used by the ME only when using the protocol T=0.

This instruction allows the ME to retrieve from the WIM, data computed by the WIM after one of the following instruction has been executed

- SelectFile
- PSO-Decipher
- PSO-ComputeDigitalSignature
- PSO-ComputeCryptographicChecksum

The WIM indicates to the ME that data are available by returning a 61XX status.

The ME MAY send a GET RESPONSE command, but if it does it MUST send the GET RESPONSE command just after the 61XX is issued by the WIM, and retrieve exactly XX bytes. Only a single GET RESPONSE command is allowed to retrieve the data.

Conditions of use

The status 61XX MUST have been issued by the WIM

If a Get Response is to be executed on one channel it MUST be executed before any command is issued on another channel.

Command APDU

CLA	0X
INS	C0
P1	00
P2	00
Lc	Empty
Data	Empty
Le	Length of expected data (equal to the XX value returned by the previous command)

Response APDU

Data	Value of the expected data
SW1-SW2	Status bytes

11.3.7 Status Words

11.3.7.1 Status Words for the WIM Native Mode

The WIM shall return the status words shown in the following table if, and only if, the corresponding conditions indicated apply.

SW1 SW2	Applicable for Commands	Description	ISO7816 Description
61XX	SELECT PSO - Decipher PSO – ComputeCryptographicChecksum PSO – ComputeDigitalSignature GENERATE ASY. KEY PAIR GENERATE KEY ASSURANCE	Normal ending of the command (data of length XX to be recovered by GET RESPONSE). Note that this status is related to the TPDU level	Normal processing, SW2 indicates the number of response bytes still available
6200	MANAGE CHANNEL Open/Close	Cannot open or close a logical channel	No information given
	GENERATE ASY. KEY PAIR	Warning – key generation process has not finished	
6300	VERIFY DISABLE VERIFICATION ENABLE VERIFICATION CHANGE REFERENCE DATA RESET RETRY COUNTER	PIN verification failed	Verification failed
63CX	VERIFY	X further retries allowed (command issued with empty body). This status word is recommended over SW1-SW2 = '6300' since it is implemented in the USIM specification	X further retries allowed (command issued with empty body)
6581	All	Memory failure (eg, data corrupted)	Memory failure
6600	MSE Restore	Security environment cannot be set	The environment cannot be set or modified [ISO7816-8]
	MSE Set PSO	No security environment set or template cannot be set	
6700	All	Lack of Lc, Data, or Le; Unexpected Lc, Data, or Le; Length rejected by the command	Wrong length

6982	MSE PSO	Execution rights no fulfilled	Security status not satisfied
	READ BINARY UPDATE BINARY	Access rights not fulfilled	
6983	VERIFY DISABLE VERIFICATION ENABLE VERIFICATION CHANGE REFERENCE DATA RESET RETRY COUNTER	PIN blocked	Authentication method blocked
	GENERATE ASY. KEY PAIR GENERATE KEY ASSURANCE	Authorization blocked (key generation is not allowed anymore)	
6985	VERIFY DISABLE VERIFICATION ENABLE VERIFICATION	In contradiction with PIN status	Condition of use not satisfied
	MSE Derive Key	Pre-master secret not ready	
	PSO	Internal data not ready or condition of use not satisfied	
	GET RESPONSE	Internal data not ready	
6986	READ BINARY UPDATE BINARY	No current EF	No current EF

6A80	MSE Set MSE Derive key	Incorrect tag	Incorrect parameters in data field
	PSO	Incorrect data	
	GENERATE ASY. KEY PAIR GENERATE KEY ASSURANCE	Bad parameters	
6A82	SELECT Application	Application not found	File not found
	SELECT File	File not found	
	PSO	Key file not found	
	VERIFY	PIN file not selected or found	

6A88	PSO MSE Derive Key GENERATE ASY. KEY PAIR GENERATE KEY ASSURANCE	Private key or master secret reference not found or incorrect Master Secret key reference	Referenced data not found
	VERIFY	PIN reference not found	
6B00	All	Incorrect parameters P1-P2	Wrong parameters P1-P2
6CXX	GET RESPONSE	Length error, the length that MUST be used is XX. Note that this status is related to the TPDU level	Wrong length Le
6D00	All	Unknown INS byte	Instruction code not supported or invalid
6E00	All	Unknown CLA byte	Class not supported
	Commands with CLA = 8X	Using any CLA = 8X command before selecting the application	
6F00	All	Technical problem with no diagnostic given	No precise diagnosis
9000	All	Normal ending of the command	Normal processing

11.3.7.2 Status Words for the WIM SCP Mode

For the commands related to cryptographic operations , it is recommended that the WIM return the status words shown in the following table, if and only if, the corresponding conditions indicated apply.

SW1 SW2	Applicable for Commands	Description	ISO7816 Description
61XX	PSO - Decipher PSO – ComputeCryptographicChecksum PSO – ComputeDigitalSignature GENERATE ASY. KEY PAIR GENERATE KEY ASSURANCE	Normal ending of the command (data of length XX to be recovered by GET RESPONSE). Note that this status is related to the TPDU level	Normal processing, SW2 indicates the number of response bytes still available
6200	GENERATE ASY. KEY PAIR	Warning – key generation process has not finished	No information given
6581	MSE PSO	Memory failure (eg, data corrupted)	Memory failure
6600	MSE Restore	Security environment cannot be set	The environment cannot be set or modified [ISO7816-8]
	MSE Set PSO	No security environment set or template cannot be set	
6700	MSE PSO	Lack of Lc, Data, or Le; Unexpected Lc, Data, or Le; Length rejected by the command	Wrong length
6982	MSE PSO	Execution rights no fulfilled	Security status not satisfied
6983	GENERATE ASY. KEY PAIR GENERATE KEY ASSURANCE	Authorization blocked (key generation is not allowed anymore)	
6985	MSE Derive Key	Pre-master secret not ready	Condition of use not satisfied
	PSO	Internal data not ready or condition of use not satisfied	
6A88	GENERATE ASY. KEY PAIR GENERATE KEY ASSURANCE	Private key or master secret reference not found or incorrect Master Secret key reference	Referenced data not found

6A80	GENERATE ASY. KEY PAIR GENERATE KEY ASSURANCE	Bad parameters	Incorrect parameters in data field

For the commands related to management of logical channels, verification and data storage operation, i.e. for MANAGE CHANNEL, GET RESPONSE, VERIFY, ENABLE VERIFICATION REQUIREMENT, DISABLE VERIFICATION REQUIREMENT, CHANGE REFERENCE DATA, RESET RETRY COUNTER, SELECT, READ BINARY, UPDATE BINARY– refer to [TS102.221] for a complete list of status words.

11.4 Usage Of The Commands

This chapter presents detailed interaction schemes between the ME and a WIM implemented on a smart card. These schemes are based on diagrams in chapter 7.

11.4.1 Open Logical Channel

Command	CLA	INS	P1	P2	Lc	Data	Le
MANAGE channel	00	70	00	00	-	-	1

The card returns the assigned logical channel number (01, 02 or 03). The subsequent commands will have this number in the two least significant bits of the CLA byte.

11.4.2 Select Application

The ME selects the WIM application using the PKCS15 AID or the WIM AID, as described in section 11.3.3.

Command	CLA	INS	P1	P2	Lc	Data	Le
SELECT Application	0X	A4	04	00	0C	xx xx xx xx xx xx xx xx xx xx xx xx	-

11.4.3 Read Configuration

The ME reads relevant parts of the files

1. Read the EF(TokenInfo).
2. Read the EF(ODF) to find location of PrKDFs, PuKDFs, CDFs, DODFs and AODFs.
3. Read the AODFs to find out which PINs must be entered to access other files; enter the required PINs.
4. Read the PrKDFs, PuKDFs, CDFs and DODF-wim to find location and relevant parameters of private keys, public keys, certificates and data objects (WTLS and TLS sessions).
5. Read the actual contents of public keys, certificates, WTLS and TLS sessions. (Private key or authentication object contents are not read by the ME.)

11.4.4 Perform WTLS RSA handshake

We assume that the ME has obtained information on the needed keys and certificates.

Restore the Security Environment

The EF TokenInfo indicates the SE number to be used for the WTLS_RSA SE. If not already done in a previous handshake, the SE must be restored using MSE-RESTORE.

Command	CLA	INS	P1	P2	Lc	Data	Le
MSE	8X	22	F3	SE No	-	-	-

Get random number from the card

The ME MAY request a random number from the card, to be placed in ClientHello.random

Command	CLA	INS	P1	P2	Lc	Data	Le
Ask Random	8X	84	00	00	-	-	0C

Verify server certificate

The ME MUST parse and verify the signed data of the server certificate. If the WIM supports signature verification, the ME MAY use the WIM in order to perform the server certificate signature verification. If the WIM does not support this operation the ME MUST perform the verification.

First, the ME sets some components (CA public key and the digest of the certificate data) in the current security environment.

Command	CLA	INS	P1	P2	Lc	Data	Le
MSE	8X	22	81	B6	XX	83 L KPubCA + 90 L Digest	-

Note that KPubCA is the public key of the CA, formatted according to [WAPWTLS]:

KPubCA = ExpLength(2 bytes) + Exponent + ModLength (2 bytes) + Modulus.

In case of a WTLS certificate the Digest is the 20-byte SHA-1 hash of the certificate data.

Secondly, the ME issues a PSO VerifyDigitalSignature operation

Command	CLA	INS	P1	P2	Lc	Data	Le
PSO	8X	2A	00	A8	XX	9E L Signature	XX

Establish pre-master secret

First, the ME sets some components in the current security environment.

Command	CLA	INS	P1	P2	Lc	Data	Le
MSE	8X	22	81	B8	XX	91 01 01 + 91 00 + 83 L KPubServer	-

The tag '91' indicates the version number of the WTLS protocol which is then concatenated to a random number (of 19 bytes) generated inside the WIM. The tag '91' indicates that the card should generate a random number internally.

Note that KPubServer is the public key of the server, formatted according to [WAPWTLS]:

KPubServer = ExpLength(2 bytes) + Exponent + ModLength (2 bytes) + Modulus.

Secondly, the ME issues a PSO Encipher command

Command	CLA	INS	P1	P2	Lc	Data	Le
PSO	8X	2A	86	00	-	-	XX

The card keeps the original value (client version and 19 random bytes) and returns the padding indicator byte '00' followed by the encrypted value. The encrypted value is transmitted to the server. The pre-master secret is the original value concatenated with KPubServer.

Derive master secret

Command	CLA	INS	P1	P2	Lc	Data	Le
MSE	8X	22	41	B4	XX	84 01 MasterSecretRef + 94 SeedLength Seed	-

The card calculates the master secret using the pre-master secret (result of the previous operation) and stores it under MasterSecretRef. This key reference is now remembered by the CCT Template.

Sign H(handshake_messages)

First, the ME sets the private key reference (according to PrKDF) in the DST of the current SE. Also the file path of the key, if not the current one, needs to be indicated.

Command	CLA	INS	P1	P2	Lc	Data	Le
MSE	8X	22	41	B6	XX	81 L filePath + 84 01 KPrivRef	-

Above,

filePath is the value of Path.path

KPrivRef is the value of CommonKeyAttributes.keyReference

Secondly, the ME issues a PSO ComputeDigitalSignature operation.

Command	CLA	INS	P1	P2	Lc	Data	Le
PSO	8X	2A	9E	9A	14	Hash code value	XX

The Le parameter is the length of the digital signature output. Eg, for a 1024 bit key it is 128 bytes.

Calculate Client Finished Check and Server Finished Check

First, the ME sets the length of the hash algorithm output to be 12. The master secret reference is already present in the CCT.

Command	CLA	INS	P1	P2	Lc	Data	Le
MSE	8X	22	41	B4	3	96 01 0C	-

The ME issues a PSO Compute Cryptographic Checksum operation

Command	CLA	INS	P1	P2	Lc	Data	Le
PSO	8X	2A	8E	80	23	“client finished” + H(handshake_messages)	0C

The ME issues a PSO Compute Cryptographic Checksum operation with different parameters (note that the handshake messages here differs from the previous one, since here the previous message is also included)

Command	CLA	INS	P1	P2	Lc	Data	Le
PSO	8X	2A	8E	80	23	“server finished” + H(handshake_messages)	0C

Calculate Client write block and Server write block

First, the ME sets the length of the cryptographic checksum algorithm output to be the length of the needed key block (MAC key, encryption key and IV). Eg, with SHA-1 hash and RC5 encryption it is 20+16+8 = 44. The master secret reference is already present in the CCT.

Command	CLA	INS	P1	P2	Lc	Data	Le
MSE	8X	22	41	B4	3	96 01 2C	-

The ME issues a PSO Compute Cryptographic Checksum operation. Note that the seed length is 16+2+16+16=50

Command	CLA	INS	P1	P2	Lc	Data	Le

PSO	8X	2A	8E	80	32	"client expansion" + seq_num + server.random + client.random	2C

The ME issues a PSO Compute Cryptographic Checksum operation.

Command	CLA	INS	P1	P2	Lc	Data	Le
PSO	8X	2A	8E	80	32	"server expansion" + seq_num + server.random + client.random	2C

Subsequent key blocks that are needed due to the key refresh, are obtained using the PSO Compute Cryptographic Checksum operation as above. There is no need to issue the current master secret reference and cryptographic checksum output length parameter since they are memorized in the CCT Template. However, these parameters MUST be set after this card application is selected (eg, after using another card application) anew, or after using another SE:

Command	CLA	INS	P1	P2	Lc	Data	Le
MSE	8X	22	41	B4	6	83 01 MasterSecretRef + 96 01 2C	-

Note that the ME should save the peer and session parameters after a secure session is negotiated and the Finished messages have been verified.

11.4.5 Perform WTLS ECDH_ECDSA Handshake

We assume that the ME has obtained information on the needed keys and certificates. Also the ME has selected the proper SE. The ME MAY get a random number for ClientHello as in RSA handshake. Also, ME MAY use the WIM for verification of the signature in the server certificate, if that is supported by the WIM; otherwise the ME MUST perform the verification itself.

First, the ME sets its own private key reference and the server public key in the current SE. Also the file path of the key, if not the current one, needs to be indicated.

Command	CLA	INS	P1	P2	Lc	Data	Le
MSE	8X	22	81	B8	XX	81 L filePath + 84 01 PrKeyRef + 83 L KPubServer	-

Secondly, the ME issues a PSO Encipher command

Command	CLA	INS	P1	P2	Lc	Data	Le
PSO	8X	2A	86	00	-	-	-

The card keeps the calculated ECDH shared secret as the pre-master secret.

The subsequent operations are as in RSA handshake.

11.4.6 Perform Application Level Signature

The procedure below describes how an application level digital signature (RSA, ECDSA) is generated.

The ME reads the necessary information on keys and certificates, in order to choose which one to use. Path indicates the file that should be selected and CommonKeyAttributes.keyReference indicates the key reference that should be used for the chosen key. CommonObjectAttributes.authId indicates the authentication object (PIN) used to protect this key.

The ME calculates the hash of the data to be signed. Depending on the application, the ME formats the hash accordingly. Eg, for RSA signature, the ME may need to contruct the digestInfo structure [RFC2313]. For ECDSA signatures, the 20-byte SHA1 hash is used as such.

The ME asks the user to enter the PIN. The ME should use the Label attribute to inform the user about the PIN in question. The ME formats the PIN according to the information in the Authentication object.

The EF TokenInfo indicates the SE number to be used for the WIM_GENERIC_RSA SE or WIM_GENERIC_ECC_SE. If not already done in a previous operation, the SE must be restored using MSE-RESTORE.

Command	**CLA**	**INS**	**P1**	**P2**	**Lc**	**Data**	**Le**
MSE	0X / 8X	22	73	SE No	-	-	-

The ME selects the file according to the path indicated in the Authentication object. The ME sends the formatted PIN to the card. The P2 parameter corresponds to the pinReference parameter in the authentication object.

Command	**CLA**	**INS**	**P1**	**P2**	**Lc**	**Data**	**Le**
Verify	0X / 8X	20	00	pin Ref	YY	FormattedPIN	-

The ME sets the key reference for the private key in the DST of the current SE. Also the file path of the key, if not the current one, needs to be indicated.

Command	**CLA**	**INS**	**P1**	**P2**	**Lc**	**Data**	**Le**
MSE	0X / 8X	22	41	B6	XX	81 L filePath + 84 01 KPrivRef	-

The ME issues a PSO ComputeDigitalSignature operation.

Command	CLA	INS	P1	P2	Lc	Data	Le
PSO	0X / 8X	2A	9E	9A	XX	FormattedHash	YY

The Le parameter is the length of the digital signature output. Eg, for a 1024 bit RSA key it is 128 bytes. For 163 bit ECC key, it is 42 bytes (concatenation of *r* and *s*, each 21 bytes).

11.4.7 Perform Application Related Deciphering

The procedure below describes how an application related deciphering (RSA, ECIES) is performed. This operation can be used for unwrapping a message key.

Based on the wrapped message key, the ME should determine which private key should be used for unwrapping. Path indicates the file that should be selected and CommonKeyAttributes.keyReference indicates the key reference that should be used for the chosen key.

The EF TokenInfo indicates the SE number to be used for the WIM_GENERIC_RSA SE or WIM_GENERIC_ECC_SE. If not already done in a previous operation, the SE must be restored using PSO-RESTORE.

Command	CLA	INS	P1	P2	Lc	Data	Le
MSE	0X / 8X	22	F3	SE No	-	-	-

The ME sets the key reference for the private key in the CT of the current SE. Also the file path of the key, if not the current one, needs to be indicated.

Command	CLA	INS	P1	P2	Lc	Data	Le
MSE	0X / 8X	22	41	B8	XX	81 L filePath + 84 01 KPrivRef	-

The ME issues a PSO Decipher operation.

Command	CLA	INS	P1	P2	Lc	Data	Le
PSO	0X / 8X	2A	80	86	XX	00 + WrappedKey	YY

The padding indicator byte is ‘00’.

The WrappedKey is the message key encrypted with the public key. For RSA, PKCS#1 block type 2 is used. Eg, for a 1024 bit key it is 128 bytes.

The Le parameter is the maximum length of the unwrapped message key.

11.4.8 Perform TLS RSA handshake

We assume that the ME has obtained information on the needed keys and certificates.

Restore the Security Environment

The EF TokenInfo indicates the SE number to be used for the TLS_RSA SE. If not already done in a previous handshake, the SE must be restored using MSE-RESTORE.

Command	CLA	INS	P1	P2	Lc	Data	Le
MSE	8X	22	F3	SE No	-	-	-

Get random number from the card

The ME MAY request a random number from the card, to be placed in ClientHello.random

Command	CLA	INS	P1	P2	Lc	Data	Le
Ask Random	8X	84	00	00	-	-	1C

Verify server certificate

The ME MUST parse and verify the signed data of the server certificate. If the WIM supports signature verification, the ME MAY use the WIM in order to perform the server certificate signature verification. If the WIM does not support this operation the ME MUST perform the verification.

First, the ME sets some components (CA public key and the digest of the certificate data) in the current security environment.

Command	CLA	INS	P1	P2	Lc	Data	Le
MSE	8X	22	81	B6	XX	83 L KPubCA + 90 L Digest	-

Note that KPubCA is the public key of the CA, formatted according to [WAPWTLS]:

KPubCA = ExpLength(2 bytes) + Exponent + ModLength (2 bytes) + Modulus.

In case of a X.509 certificate signed with RSA, the Digest is the DigestInfo of the certificate data encoded as defined in [RFC2313], using DER encoding.

Secondly, the ME issues a PSO VerifyDigitalSignature operation

Command	CLA	INS	P1	P2	Lc	Data	Le

PSO	8X	2A	00	A8	XX	9E L Signature	XX

Establish pre-master secret

First, the ME sets some components in the current security environment.

Command	CLA	INS	P1	P2	Lc	Data	Le
MSE	8X	22	81	B8	XX	91 02 03 01 + 91 00 + 83 L KPubServer	-

The tag '91' indicates the version number of the TLS protocol which is then concatenated to a random number (of 46 bytes) generated inside the WIM. The tag '91' indicates that the card should generate a random number internally.

Note that KPubServer is the public key of the server, formatted according to [WAPWTLS]:

KPubServer = ExpLength(2 bytes) + Exponent + ModLength (2 bytes) + Modulus.

Secondly, the ME issues a PSO Encipher command

Command	CLA	INS	P1	P2	Lc	Data	Le
PSO	8X	2A	86	00	-	-	XX

The card keeps the original value (client version and 46 random bytes) and returns the padding indicator byte '00' followed by the encrypted value. The encrypted value is transmitted to the server. The pre-master secret is the original value.

Derive master secret

Command	CLA	INS	P1	P2	Lc	Data	Le
MSE	8X	22	41	B4	XX	84 01 MasterSecretRef + 94 SeedLength Seed	-

The card calculates the master secret using the pre-master secret (result of the previous operation) and stores it under MasterSecretRef. This key reference is now remembered by the CCT Template.

Sign H(handshake_messages)

First, the ME sets the private key reference (according to PrKDF) in the DST of the current SE. Also the file path of the key, if not the current one, needs to be indicated.

Command	CLA	INS	P1	P2	Lc	Data	Le
MSE	8X	22	41	B6	XX	81 L filePath + 84 01 KPrivRef	-

Above,

filePath is the value of Path.path

KPrivRef is the value of CommonKeyAttributes.keyReference

Secondly, the ME issues a PSO ComputeDigitalSignature operation.

Command	CLA	INS	P1	P2	Lc	Data	Le
PSO	8X	2A	9E	9A	14	Hash code value	XX

The Le parameter is the length of the digital signature output. Eg, for a 1024 bit key it is 128 bytes.

Calculate Client Finished Check and Server Finished Check

First, the ME sets the length of the hash algorithm output to be 12. The master secret reference is already present in the CCT.

Command	CLA	INS	P1	P2	Lc	Data	Le
MSE	8X	22	41	B4	3	96 01 0C	-

The ME issues a PSO Compute Cryptographic Checksum operation

Command	CLA	INS	P1	P2	Lc	Data	Le
PSO	8X	2A	8E	80	23	“client finished” + H(handshake_messages)	0C

The ME issues a PSO Compute Cryptographic Checksum operation with different parameters (note that the handshake messages here differs from the previous one, since here the previous message is also included)

Command	CLA	INS	P1	P2	Lc	Data	Le
PSO	8X	2A	8E	80	23	“server finished” + H(handshake_messages)	0C

Calculate Client and Server write block

First, the ME sets the length of the cryptographic checksum algorithm output to be the length of the needed key block (MAC key, encryption key and IV). Eg, with TLS_RSA_3DES_EDE_CBC_SHA, it is 2*(24+20+8) = 104. The master secret reference is already present in the CCT.

Command	CLA	INS	P1	P2	Lc	Data	Le
MSE	8X	22	41	B4	3	96 01 68	-

The ME issues a PSO Compute Cryptographic Checksum operation. Note that the seed length is 13+32+32=77

Command	CLA	INS	P1	P2	Lc	Data	Le
PSO	8X	2A	8E	80	4D	“key expansion” + server.random + client.random	68

Note that the ME should save the session parameters after a secure session is negotiated and the Finished messages have been verified.

11.4.9 Generate Asymmetric Key Pair

The procedure below describes how the ME uses the WIM to generate a key corresponding to a given private key reference in the WIM.

The ME reads the necessary information on keys and certificates, in order to choose which one to use. Path indicates the file that should be selected and CommonKeyAttributes.keyReference indicates the key reference that should be used for the chosen key.

The EF TokenInfo indicates the SE number to be used for the WIM_GENERIC_RSA SE or WIM_GENERIC_ECC SE. If not already done in a previous operation, one of these SE must be restored using MSE Restore prior to GENERATE_ASYMMETRIC_KEY_PAIR usage.

Command	CLA	INS	P1	P2	Lc	Data	Le
MSE	0X / 8X	22	F3	SE No	-	-	-

The ME sets, in the DST of the current SE, the file path and the key reference of the private key that need to be generated.

Command	CLA	INS	P1	P2	Lc	Data	Le
MSE	0X / 8X	22	41	B6	XX	81 L filePath + 84 01 KprivRef	-

The ME issues a GENERATE_ASYMMETRIC_KEY_PAIR operation.

Command	CLA	INS	P1	P2	Lc	Data	Le
GENERATE_ASYMMETRIC_KEY_PAIR	8X	46	00	00	XX	00 [9E L authentication data (digital signature) or 8E L authentication data (HMAC) [C0 L encrypted New PIN value] [C1 L encrypted New PIN label] [C2 L encrypted new private & public key label] [C3 User PIN Value]]	YY

May need to continue calling, in a loop, by sending the following command as long as the SW 6200 is returned:

Command	CLA	INS	P1	P2	Lc	Data	Le
GENERATE_ASYMMETRIC_KEY_PAIR	8X	46	00	00	XX	00	YY

The Le parameter is the length of the returned data, either public key hash of the newly generated key pair or the Challenge and WIM serial number.

11.4.10 Generate Key Assurance

The procedure below describes how the ME uses the WIM to retrieve signed assurance information corresponding to a given private key in the WIM.

The ME reads the necessary information on keys and certificates, in order to choose which one to use. Path indicates the file that should be selected and CommonKeyAttributes.keyReference indicates the key reference that should be used for the chosen key.

The EF TokenInfo indicates the SE number to be used for the WIM_GENERIC_RSA SE or WIM_GENERIC_ECC SE. If not already done in a previous operation, one of these SE must be restored using MSE Restore prior to GenerateKey Assurance usage.

Command	CLA	INS	P1	P2	Lc	Data	Le
MSE	0X / 8X	22	F3	SE No	-	-	-

The ME sets, in the DST of the current SE, the file path and the key reference of the private key for which assurance information are expected to be signed. The assurance key used is implicitly known.

Command	CLA	INS	P1	P2	Lc	Data	Le
MSE	0X / 8X	22	41	B6	XX	81 L filePath + 84 01 KprivRef	-

The ME issues a GenerateKeyAssurance operation.

Command	CLA	INS	P1	P2	Lc	Data	Le
GENERATE KEY ASSURANCE	8X	46	01	00	XX	00 [9E L Digital signature Authentication code or 8E L HMAC Authorisation code]	YY

The Le parameter is the length of the returned data, either AssuranceInfo, AssuranceSignature, AssuranceCertificate or the Challenge and WIM serial number.

11.5 Usage of the TLS check value

ME MUST check and update the TLS check value.

12.WIM Electronic Identification Profile of PKCS#15

This section describes a profile of [PKCS15] for the WIM. This profile contains the essential parts of the Electronic Identification Profile, and additionally WTLS and TLS specific functionality.

12.1 PKCS#15 objects

12.1.1 Private Keys

A WIM EID module MUST contain at least two private keys, of which one should be usable for application level digital signature purposes and have its key usage flags set to 'nonRepudiation' only. The union of the key usage flags for the other keys should contain the values 'sign' and optionally 'decrypt'. Authentication objects MUST protect all private keys.

Each nonRepudiation key MUST be protected with an authentication object used only for this key. The key length should be sufficient for intended purposes (eg, 1024 bits or more in the RSA case and 160 bits or more in the EC case, assuming all other parameters has been chosen in a secure manner).

The non-repudiation key is a separate key from that used for WTLS- or TLS-authentication. The authentication key may additionally support decryption.

12.1.2 Certificates

For each private key at least one corresponding certificate should reside on the card, either as a URL or as such.

If the WIM application issuer stores CA certificates, it is recommended that they are stored in a protected file, pointed at by a CDF file which is modifiable by the WIM issuer only (or not modifiable at all). This implies usage of the trustedCertificates field in the PKCS15ODF type.

12.1.3 Data Objects

There MUST be exactly one Peers-wtls Data Object and exactly one Sessions-wtls Data Object. When TLS is supported, there MUST be exactly one Sessions-tls Data Object.

12.1.4 Authentication Objects

As follows from the description above, the module MUST be capable of protecting files with authentication objects, and at least two authentication objects must be present. In the PIN case, the PIN MUST be at least 4 characters.

Three incorrect verifications of a certain PIN code MUST block the PIN and all associated security services. A blocked PIN may be unblocked using an unblocking code.

A PIN used for a non-repudiation key MUST be invalidated by the WIM, after computing a digital signature. So, the PIN has to be entered for each digital signature operation separately.

12.1.4.1 Recommended PIN Format

It is recommended that the PIN parameters have the following values (conforming to [GSM11.11])

Attribute	Value
PinAttributes.pinType	ascii-numeric
PinAttributes.minLength	04 (or up to 08)
PinAttributes.storedLength	08
PinAttributes.padChar	FF

12.2 Access Control Rules

The module MUST be capable to perform cryptographical operations. The private keys must be private objects, and marked as 'sensitive', meaning that they MUST NOT ever leave the module. They MAY be replaced.

The table below presents recommended access conditions for WIM files. With these access conditions, all data that is needed for the ME for finding out properties of the token and listing different kind of objects, may be read without cardholder verification. Verification is required for use of cryptographic keys (private keys, WTLS and TLS master secrets), and for updating any data (in cases when updates are allowed). Note that with these access conditions, the TokenInfo flag 'loginRequired' is not set.

The WIM MUST enforce and the ME MUST support access conditions presented below.

File	Access Conditions	Comment
MF	Create: SYS Delete: SYS	
EF(DIR)	Read: ALW Update: SYS	
DF(PKCS15)	Create: SYS Delete: SYS	
EF(TokenInfo)	Read: ALW Update: NEV	
EF(ODF)	Read: ALW Update: SYS	
AODFs	Read: ALW Update: NEV	
PrKDFs	Read: ALW Update: SYS | NEV	
CDFs	Read: ALW Update: CHV | SYS | NEV	A CDF must be possible to modify by the cardholder if it points to certificate objects that are possible to modify by the cardholder. For modification, the PIN-G must be entered.
EF(DODF-wim) pointing to EF(Peers-wtls), EF(Sessions-wtls) and EF(Sessions-tls)	Read: ALW Update: SYS | NEV	

Trusted CDFs	Read: Update:	ALW SYS | NEV	
PIN files	Read: Update:	NEV CHV | SYS | NEV	The PIN value can only be updated by the cardholder using verification related operations.
Key files (private keys, WTLS and TLS master secrets)	Read: Update: Crypt:	NEV SYS | NEV CHV	Cardholder verification is required for all cryptographic operations (MSE, PSO)
Certificate files	Read: Update:	ALW CHV | SYS | NEV	If a certificate file is updateable, then in the corresponding certificate object, the flag 'modifiable' is set (flag 'private' is not set), and the authId field references PIN-G.
EF(Peers-wtls)	Read: Update:	ALW CHV	In the corresponding data object, the flag 'modifiable' is set, the flag 'private' is not set, and the authId field references PIN-G.
EF(Sessions-wtls)	Read: Update:	ALW CHV	In the corresponding data object, the flag 'modifiable' is set, the flag 'private' is not set, and the authId field references PIN-G.
EF(Sessions-tls)	Read: Update:	ALW CHV	In the corresponding data object, the flag 'modifiable' is set, the flag 'private' is not set, and the authId field references PIN-G.
EF(UnusedSpace)	Read: Update:	ALW CHV	

Access conditions are defined according to the table below.

Type	**Meaning**
NEV	The operation is never allowed, not even after cardholder verification.
ALW	The operation is always allowed, without cardholder verification.
CHV	The operation is allowed after a successful cardholder verification.
SYS	The operation is allowed after a system key presentation, typically available only to the card issuer, possibly using specific management operations. Note that these operations are not defined in this specification. So, 'SYS' access is here for informational purposes only.

12.3 Attribute Formats

All data items that are updateable, SHOULD have fixed length within the same file (eg, CDF). The following values SHOULD be used.

Attribute	Bytes	Comment
Path.path	6 or 2	The path is a concatenation of file IDs of the MF, the current DF and the EF, or indicates the EF file ID in the current DF
Path.index	2	When present, always use two bytes for the offset..
Path.length	2	Always use two length bytes, to indicate the size of the object.
Label	32	The text should be left justified, padded with blank characters.
Identifier	20	This is the SHA-1 hash of the public key

The following restrictions apply

- authentication object identifier (`authId`) MUST be encoded with one byte
- file path has a maximal length of 12 bytes
- security environment number is 1…127

13.Implementation Notes

13.1 Implementing WIM in a GSM SIM Card

The WIM is implemented in a GSM SIM Card [GSM11.11] as a card application. The existence of the WIM application has no effect to the GSM functionality. So, a ME that does not support WIM, can still use the GSM application.

A GSM SIM card that has the WIM application MUST support logical channels. The GSM application uses the basic channel (which is conformance with the CLA byte A0 used for GSM). The WIM application uses the channel 1, 2 or 3.

To activate the WIM application, the ME first issues the MANAGE CHANNEL Open command.

Command	CLA	INS	P1	P2	Lc	Data	Le
MANAGE CHANNEL	00	70	00	00	-	-	1

The card should return the assigned logical channel number (01, 02 or 03). Otherwise, the ME concludes that the card does not support WIM.

The subsequent commands will have the logical channel number in the two least significant bits of the CLA byte.

As the next step, the ME attempts to select the WIM application, as defined in section 11.3.3. If application selection fails, the ME concludes that the card does not support WIM.

13.2 WIM for Networks Not Utilizing a Smartcard Based SIM

In networks that do not utilize a smartcard based SIM, the WIM can be implemented

- in a smartcard that contains the WIM application only,
- in a smartcard that contains the WIM and other useful applications, or
- in a tamper-resistant device, other than a smartcard

For smartcard implementations, the WIM has been specified as an independent application. So, it is anticipated, that networks that in the future will introduce SIMs, will be able to integrate the WIM with minimal effort.

Usage of logical channels makes it possible to use the WIM application simultaneously and without interference with the SIM application.

13.3 Using Logical Channels

A WIM ICC implementation is not required to support logical channels if the WIM is the only application in card. The ME MAY, however, issue the MANAGE CHANNEL Open command.

Command	CLA	INS	P1	P2	Lc	Data	Le

MANAGE CHANNEL	00	70	00	00	-	-	1

If the card does not support logical channels, it MUST return the status code SW1SW2 = '6D00'. The ME concludes that the basic channel should be used to access the WIM. (The ME may also use the logical channel related information in the ATR.)

13.4 Saving Certificates

Certificates may be saved, replaced and deleted according to [PKCS15], chapter 5.8.1, assuming that sufficient privileges exist.

To save a certificate the ME should

1. write the certificate data in an unused space of an elementary file
2. update the EF(UnusedSpace)
3. update the CDF

It is possible that due to power loss, not all the above operations are completed. As a consequence, it may occur that all data is not consistent. Eg, it is possible that certificate data is written (step 1) but that space is still marked "unused"; or the space is marked used (step 2), but it cannot be accessed since it is not referenced in a CDF. It is possible to recover from this situation by rewriting the CDFs and EF(UnusedSpace) based on the information in the CDFs. This "garbage collection" may be needed if no suitable space can be found for a new certificate. For recovery, the ME should

1. read the information in the CDF
2. rewrite the CDF so that there are no empty records (in case there were such originally)
3. update the EF(UnusedSpace) so that free space records point to area that is not referenced by the CDF

13.5 Usage of PINs

The first object in the first AODF listed in EF(ODF) is considered as a General PIN (PIN-G). If not otherwise indicated, all files (eg, CDF, PrKDF) are write-protected with this PIN. Obviously, EF(ODF) and EF(AODF) SHOULD be readable without a PIN verification. (Note: according to 12.2, files like CDF, PrKDF, DODF and certificate files should be readable without entering a PIN.)

In a typical case, the PIN-G is used to protect all files (which need to be protected) and keys except non-repudiation keys. If the PIN-G is not disabled, the ME must send the PIN-G after the WIM application is selected, in order to be able to use keys and perform other operations that require the PIN-G. More precisely, the ME SHOULD do the following when the secure functions are required the first time

1. Open a logical channel, if logical channels are used
2. Select the WIM application
3. Read EF(TokenInfo)
4. Read EF(ODF)

5. Read EF(AODF)
6. Read the information about PIN-G
7. Find out if PIN-G is enabled
8. Enter the PIN-G (if operations requiring PIN-G are used)

The ME does not need to send the PIN-G, unless or before protected operations are performed. E.g. the ME may read files that are readable without a PIN. Also, PIN-G need not be sent before using non-repudiation keys, since they are protected with a separate PIN.

After the PIN-G is entered, it remains valid until the logical channel is closed. The ME SHOULD close the logical channel when the secure functions are no more required. When the logical channel is opened again, the PIN will be required. If logical channels are not supported, the ME SHOULD reset the card in order to validate the PIN.

Note that a non-repudiation key is protected with a non-repudiation PIN (PIN-NR). If there are several non-repudiation keys, then each key MUST be protected with a separate PIN-NR.

When the user is asked to enter PIN-G or PIN-NR, it should be made clear to the user that the PIN entry procedure is safe and the entered PIN is not going to be sent across the network. Under any circumstances the entered PIN (PIN-G, PIN-NR) MUST NOT be sent to the WAE.

All private key operations are protected either with PIN-G or PIN-NR, as MUST be indicated in the PKCS #15 structure. PIN-G protects all operations with Master Secrets, as MUST be indicated in the PKCS #15 structure.

PIN-G also protects all operations with public keys and pre-master secrets (e.g. Encipher, Verify Signature, MSE-Derive key). On the other hand all MSE operations for setting attributes in Security Environments do not demand a prior verification of PIN-G.

Note that WIM on a smart card must not use PIN references that are reserved in [TS102.221].

13.6 Using the WIM for Non-WAP Applications

Besides WTLS, TLS and WAP application layer security (through WMLScript and ECMAScript) the WIM may be used to secure non-WAP applications that require a tamper resistant device to perform the following functionality

- signing for authentication purposes (eg, SSL [SSL])
- signing for non-repudiation purposes (eg, S/MIME [S/MIME])
- private key decryption (eg, S/MIME)
- storage of user certificates (eg, SSL, S/MIME)
- storage of trusted CA certificates (eg, SSL, S/MIME, Java security)

For WTLS, the WIM is used for managing secure sessions, so that the WTLS pre-master secret and master secret never leave the tamper resistant module. The WIM does not support analogous functionality for SSL.

13.6.1 Signing

Signing is described in chapter 6.2.2. The corresponding smart card operations are described in chapter 11.3.6.

The WIM RSA signing operation is performed according to [RFC2313], using block type 1. So all conforming applications can take advantage of this feature. The input for signing is a formatted hash, or any byte string, the length of which is limited to the value allowed by the [RFC2313] for a certain modulus size. For SSL and TLS client authentication, the formatted hash given as the input is a concatenation of SHA-1 and MD5 hashes (36 bytes). For S/MIME and other [RFC2315] compatible applications, a `DigestInfo` structure is used as the input.

The WIM ECDSA signature is performed according to [X9.62]. So, all conforming applications can take advantage of this feature.

For a non-repudiation key, the WIM verification (entering the PIN) is required for each signature separately.

13.6.2 Private Key Decryption

Private key decryption (deciphering) is described in chapter 6.2.1. The corresponding smart card operations are described in chapter 11.3.6.

For RSA, the WIM decryption is based on the assumption that the public key encryption is done according to [RFC2313] block type 2. So, all conforming applications can take advantage of this feature. For [RFC2315] compatible applications, this feature can be used to decrypt the content-encryption key, eg, to decrypt a received S/MIME message.

13.6.3 Certificate Storage

User certificates used for, eg, SSL client authentication or signing S/MIME messages, can be stored in the WIM. In this case, the certificates must confirm to relevant standards. However, due to the large size of these certificates it may not be optimal to store these certificates in the WIM. In this case, the WIM may store a certificate URL, or the certificate may be retrieved from a directory using the key identifier (public key hash) as a search criteria. Note that from security point of view, there is no requirement to store user certificates in a tamper resistant device.

The WIM may store trusted CA certificates so that they cannot be modified by the user, ie, the user cannot add or delete certificates in the trusted certificates list. This feature may be useful for verifying servers (eg, SSL, TLS) or downloaded applications (eg, in Java applications).

The WIM storage format is compatible with [PKCS15], which makes it easier for non-WAP host side applications to access information stored in the WIM.

13.7 Server RSA Public Key Constraints When Using T=0

The following chapter applies for smart card implementing the transmission protocol T=0 and to servers wishing to transfer Server RSA Public keys to the WIM for Key Transport operation or Signature verification.

Because of T=0 transport protocol constraints, the data field of APDUs sent to the WIM cannot exceed 255 bytes. In addition, data are sent to the WIM wrapped into TLV data element, reducing the remaining length for actual data as explained below.

In short, it is recommended that the length of the Server RSA Public exponent is at maximum 3 bytes.

In case larger exponents are needed, this chapter indicates how to calculate the maximum size for the exponent.

The maximum size supported by the MSE SET commands preceding both PSO_VerifyDigitalSignature and PSO_Encipher Key Transport: In the MSE command preceding a PSO VerifyDigitalSignature a digest TLV and a public key TLV are sent to the WIM in the following format:

Digest Tag	Digest Len	Digest	Publickey Tag	Publickey Len	Exp Len	Exp	Mod Len	Modulus

For example

- 22 bytes are used for the digest, its tag and length,
- 3 bytes are used for PublicKey tag and length,
- 4 bytes are used for Exponent length and Modulus Length (WTLS coding)

In this case the maximum public key exponent length can not exceed 98 bytes, if 128-byte modulus is used.

In the MSE command preceding the PSO Encipher Key Transport operation, a client version TLV, a random TL and a Public key TLV are sent to the WIM in the following format:

Tag	Len	Client Version	Tag	Len	Tag	Len	ExpLen	Exp	ModLen	Modulus

For example

- 3 bytes are used for the client version, its tag and length
- 2 bytes are used for the random tag and length (00, no Value field here)

- 3 bytes are used for PublicKey tag and length,
- 4 bytes are used for Exponent length and Modulus Length (WTLS coding)

In this case the maximum public key exponent length can not exceed 115 bytes, if 128-byte modulus is used.

14.WIM Static Conformance Requirement

This static conformance requirement [IOPPROC] lists a minimum set of functions that can be implemented to help ensure that WIM implementations and ME implementations will be able to inter-operate. The "Status" column indicates if the function is mandatory (M) or optional (O).

14.1 WIM Options

14.1.1 General WIM Options

Item	Function	Reference	Status	Requirement
WIM-ICC-050	Transport layer security supported	5.2	M	WIM-ICC-001 OR WIM-ICC-051
WIM-ICC-001	WTLS supported	5.2	O	WIM-ICC-017 AND WIM-ICC-018 AND WIM-ICC-020 AND WIM-ICC-027
WIM-ICC-051	TLS supported	5.2	O	WIM-ICC-052 AND WIM-ICC-053 AND WIM-ICC-054 AND WIM-ICC-056 AND WIM-ICC-140
WIM-ICC-002	Generic (application level) functionality Signing of hash	5.3.2	M	WIM-ICC-023 OR WIM-ICC-025
WIM-ICC-003	Generic (application level) functionality Unwrap (decipher) a key	5.3.1	O	WIM-ICC-024 OR WIM-ICC-026
WIM-ICC-004	Data storage PKCS#15 ODF	9.4.1	M	
WIM-ICC-005	Data storage PKCS#15 TokenInfo	9.4.7	M	
WIM-ICC-006	Data storage PKCS#15 PrKDF	9.4.2	M	
WIM-ICC-007	Data storage PKCS#15 PuKDF	9.4.3	O	
WIM-ICC-008	Data storage PKCS#15 CDF	9.4.4	M	
WIM-ICC-009	Data storage PKCS#15 CDF trusted certificates	9.4.4	O	
WIM-ICC-090	Data storage PKCS#15 CDF useful certificates	9.4.4	O	
WIM-ICC-010	Data storage PKCS#15 AODF	9.4.6	M	
WIM-ICC-011	Data storage PKCS#15 DODF-wim	9.4.5	M	
WIM-ICC-012	Data storage PKCS#15 UnusedSpace	9.4.8	M	
WIM-ICC-013	Data storage Private key, use by ME	9.4.9, 12.1.1	M	

WIM-ICC-014	Data storage Public key, read by ME	9.4.3	O	
WIM-ICC-015	Data storage Certificate, read by ME	9.4.9, 12.1.1	M	
WIM-ICC-016	Data storage Certificate, store by ME	9.4.9, 13.4	M	
WIM-ICC-017	Data storage WTLS Peers	9.4.10	O	
WIM-ICC-018	Data storage WTLS Sessions	9.4.11	O	
WIM-ICC-052	Data storage TLS Session	9.4.13	O	
WIM-ICC-019	Random number generation	5.2	M	
WIM-ICC-020	WTLS key exchange algorithms; at least one supported	7	O	WIM-ICC-021 OR WIM-ICC-022
WIM-ICC-021	WTLS RSA	7.1	O	WIM-ICC-023 AND WIM-ICC-028 AND WIM-ICC-130
WIM-ICC-022	WTLS ECDH	7.2	O	WIM-ICC-025 AND WIM-ICC-029 AND WIM-ICC-131
WIM-ICC-053	TLS key exchange algorithm RSA	8	O	WIM-ICC-023 AND WIM-ICC-028 AND WIM-ICC-130
WIM-ICC-023	Generic (application level) algorithms. RSA signing	5.3.2	O	WIM-ICC-028
WIM-ICC-024	Generic (application level) algorithms. RSA decryption	5.3.1	O	WIM-ICC-028
WIM-ICC-025	Generic (application level) algorithms. ECDSA signing	5.3.2	O	WIM-ICC-030 OR WIM-ICC-032
WIM-ICC-026	Generic (application level) algorithms. ECIES decryption	5.3.1	O	WIM-ICC-030 OR WIM-ICC-032
WIM-ICC-027	WTLS pseudo-random function based on SHA-1	7	O	
WIM-ICC-054	TLS pseudo-random function based on SHA-1 and MD5	8	O	
WIM-ICC-028	Minimum RSA modulus length is 1024 bits, when RSA supported	12.1.1	O	
WIM-ICC-029	ECC key length (bits), when ECC supported Minimum 160	12.1.1	O	
WIM-ICC-060	If ECC is used, at least one basic curve MUST be supported	WTLS App. A	O	WIM-ICC-030 OR WIM-ICC-032
WIM-ICC-030	ECC basic curves Curve 5 (163 bits)	WTLS App. A	O	
WIM-ICC-032	ECC basic curves Curve 7 (160 bits)	WTLS App. A	O	
WIM-ICC-033	ECC non-basic curves Curve 1 (113 bits)	WTLS App. A	O	
WIM-ICC-034	ECC non-basic curves Curve 3 (163 bits)	WTLS App. A	O	

WIM-ICC-083	ECC non-basic curves Curve 4 (113 bits)	WTLS App. A	O	
WIM-ICC-031	ECC non-basic curves Curve 6 (112 bits)	WTLS App. A	O	
WIM-ICC-035	ECC non-basic curves Curve 8 (112 bits)	WTLS App. A	O	
WIM-ICC-036	ECC non-basic curves Curve 9 (160 bits)	WTLS App. A	O	
WIM-ICC-080	ECC non-basic curves Curve 10 (233 bits)	WTLS App. A	O	
WIM-ICC-081	ECC non-basic curves Curve 11 (233 bits)	WTLS App. A	O	
WIM-ICC-082	ECC non-basic curves Curve 12 (224 bits)	WTLS App. A	O	
WIM-ICC-037	Private keys Authentication key Note: This key MUST be a separate key to each of the non-repudiation keys but may be used for decryption.	12.1.1	M	
WIM-ICC-038	Private keys Decryption key (application level) Note: This key MUST be a separate key to each of the non-repudiation keys but may be combiden with the Authentication key.	12.1.1	O	
WIM-ICC-039	Private keys Non-repudiation key (application level) Note: Each non-repudiation key MUST be a separate key and separate to the Authentication and Decryption key(s).	12.1.1	M	
WIM-ICC-040	PIN handling Recommended PIN format	12.1.4.1	M	
WIM-ICC-041	Digital signature verification RSA	5.3.3	O	
WIM-ICC-042	Digital signature verification ECDSA	5.3.3	O	
WIM-ICC-043	PKCS#15 file path fields Use of fields	9.4.1	M	WIM-ICC-044 OR WIM-ICC-045
WIM-ICC-044	PKCS#15 file path fields Use of 2-byte file identifiers	9.4.1	O	
WIM-ICC-045	PKCS#15 file path fields Use of absolute or relative paths	9.4.1	O	
WIM-ICC-056	PKCS#15 certificates contain issuerNameHash	9.4.4	O	

14.1.2 WIM ICC Options

Item	Function	Reference	Status	Requirement
WIM-ICC-101	(Item removed)			
WIM-ICC-102	Direct application selection	11.3.3.1	M	
WIM-ICC-103	(Item removed)			
WIM-ICC-104	Logical channels. A WIM ICC that supports also some other applications (eg, GSM SIM) MUST support logical channels.	11.3.2	O	WIM-ICC-105
WIM-ICC-105	ICC commands MANAGE CHANNEL MANAGE CHANNEL MUST be supported by an ICC that supports multiple applications.	11.3.2	O	
WIM-ICC-106	ICC commands VERIFY	11.3.4.1	M	
WIM-ICC-107	ICC commands DISABLE VERIFICATION	11.3.4.2	O	
WIM-ICC-108	ICC commands ENABLE VERIFICATION	11.3.4.3	O	
WIM-ICC-109	ICC commands CHANGE REFERENCE DATA	11.3.4.4	M	
WIM-ICC-110	ICC commands RESET RETRY COUNTER	11.3.4.5	M	
WIM-ICC-111	ICC commands SELECT	11.3.5.1	M	
WIM-ICC-112	ICC commands READ BINARY	11.3.5.2	M	
WIM-ICC-113	ICC commands UPDATE BINARY	11.3.5.3	M	
WIM-ICC-114	ICC commands MANAGE SECURITY ENVIRONMENT	11.3.6.1	M	
WIM-ICC-115	ICC commands PERFORM SECURITY OPERATION (all but Key Transport and Key Agreement)	11.3.6.4 11.3.6.7, 11.3.6.8, 11.3.6.9, 11.3.6.10	M	WIM-ICC-130 OR WIM-ICC-131
WIM-ICC-130	ICC commands PERFORM SECURITY OPERATION Key Transport	11.3.6.4, 11.3.6.5	O	
WIM-ICC-131	ICC commands PERFORM SECURITY OPERATION Key Agreement	11.3.6.4, 11.3.6.6	O	
WIM-ICC-116	ICC commands ASK RANDOM	11.3.6.12	M	
WIM-ICC-117	ICC commands GET RESPONSE	11.3.6.16	M	
WIM-ICC-118	ICC size; at least one supported	11.1	M	WIM-ICC-119 OR WIM-ICC-120
WIM-ICC-119	ID-1	11.1	O	
WIM-ICC-120	ID-000 (Plug-in)	11.1	O	

WIM-ICC-121	Transmission protocols T=0	11.2	M	
WIM-ICC-122	Transmission protocols T=1	11.2	O	
WIM-ICC-123	Supply voltage 3 V	11.2	M	
WIM-ICC-124	Supply voltage 5 V	11.2	O	
WIM-ICC-125	Enforce access control rules	12.2	M	
WIM-ICC-127	Command modes	11.3	M	WIM-ICC-128 OR WIM-ICC-129
WIM-ICC-128	Native mode	11.3	O	
WIM-ICC-129	SCP mode	11.3	O	
WIM-ICC-135	Supply voltage 1.8 V	11.2	O	
WIM-ICC-140	Support TLS check value	11.5	O	
WIM-ICC-141	Icc command Generate asymmetric key pair	11.3.6.13	O	
WIM-ICC-142	Icc command Generate key assurance	11.3.6.14	O	

14.2 ME Options

14.2.1 General ME Options

Item	Function	Reference	Status	Requirement
WIM-C-001	WTLS	[WAPWTLS], 7	O	WIM-C-019 AND WIM-C-020 AND WIM-C-022 AND WIM-C-039
WIM-C-050	TLS	[TLS10], 8	O	WIM-C-052 AND WIM-C-053 AND WIM-C-055 AND WIM-C-140
WIM-C-002	Generic (application level) functionality Signing of hash	5.3.2	O	WIM-C-025 OR WIM-C-027
WIM-C-003	Generic (application level) functionality Unwrap (decipher) a key	5.3.1	O	WIM-C-026 OR WIM-C-028
WIM-C-004	Data storage PKCS#15 ODF	9.4.1	M	
WIM-C-005	Data storage PKCS#15 TokenInfo	9.4.7	M	
WIM-C-006	Data storage PKCS#15 PrKDF	9.4.2	M	
WIM-C-007	Data storage PKCS#15 PuKDF	9.4.3	O	
WIM-C-008	Data storage PKCS#15 CDF	9.4.4	M	
WIM-C-009	Data storage PKCS#15 CDF trusted certificates	9.4.4	M	
WIM-C-090	Data storage PKCS#15 CDF useful certificates	9.4.4	M	
WIM-C-010	Data storage PKCS#15 AODF	9.4.6	M	
WIM-C-011	Data storage PKCS#15 DODF	9.4.5	M	
WIM-C-012	Data storage PKCS#15 UnusedSpace	9.4.8	M	
WIM-C-013	Data storage Use private key	9.4.2	M	
WIM-C-014	Data storage Read public key	9.4.3	O	
WIM-C-015	Data storage Read user certificate	9.4.4	M	
WIM-C-016	Data storage Store user certificate	9.4.4	M	
WIM-C-017	Data storage Read CA certificate	9.4.4	M	
WIM-C-018	Data storage Store CA certificate	9.4.4	M	

WIM-C-019	Data storage WTLS Peers	9.4.10	O	
WIM-C-020	Data storage WTLS Sessions	9.4.11	O	
WIM-C-052	Data storage TLS Sessions	9.4.13	O	
WIM-C-021	Use of random numbers generated by the WIM	5.2	O	
WIM-C-022	WTLS key exchange algorithms; at least one supported	7	O	WIM-C-023 OR WIM-C-024
WIM-C-023	WTLS RSA	7.1	O	WIM-C-029 AND WIM-C-030 AND WIM-C-130
WIM-C-024	WTLS ECDH	7.2	O	WIM-C-029 AND WIM-C-131 AND (WIM-C-032 OR WIM-C-034)
WIM-C-053	TLS key exchange algorithms RSA	8	O	WIM-C-054 AND WIM-C-030 AND WIM-C-130
WIM-C-025	Generic (application level) algorithms RSA signing	5.3.2	O	WIM-C-030
WIM-C-026	Generic (application level) algorithms RSA decryption	5.3.1	O	WIM-C-030
WIM-C-027	Generic (application level) algorithms ECDSA signing	5.3.2	O	WIM-C-032 OR WIM-C-034
WIM-C-028	Generic (application level) algorithms ECIES decryption	5.3.1	O	WIM-C-032 OR WIM-C-034
WIM-C-029	Use WTLS pseudo-random function based on SHA-1	7	O	
WIM-C-054	Use TLS pseudo-random function based on SHA-1 and MD5	8	O	
WIM-C-030	If RSA is supported, the minimum expected RSA modulus length is 1024 bits, for signing performed in WIM	12.1.1	O	
WIM-C-060	If ECC is used, at least one MUST be supported	WTLS App. A	O	WIM-C-032 OR WIM-C-034
WIM-C-032	ECC basic curves Curve 5 (163 bits)	WTLS App. A	O	
WIM-C-034	ECC basic curves Curve 7 (160 bits)	WTLS App. A	O	
WIM-C-035	ECC non-basic curves Curve 1 (113 bits)	WTLS App. A	O	
WIM-C-036	ECC non-basic curves Curve 3 (163 bits)	WTLS App. A	O	
WIM-C-031	ECC non-basic curves Curve 4 (113 bits)	WTLS App. A	O	
WIM-C-033	ECC non-basic curves Curve 6 (112 bits)	WTLS App. A	O	
WIM-C-037	ECC non-basic curves Curve 8 (112 bits)	WTLS App. A	O	

WIM-C-038	ECC non-basic curves Curve 9 (160 bits)	WTLS App. A	O	
WIM-C-080	ECC non-basic curves Curve 10 (233 bits)	WTLS App. A	O	
WIM-C-081	ECC non-basic curves Curve 11 (233 bits)	WTLS App. A	O	
WIM-C-082	ECC non-basic curves Curve 12 (224 bits)	WTLS App. A	O	
WIM-C-039	Use authentication key for WTLS client authentication	12.1.1	O	
WIM-C-040	Use authentication key for decryption	12.1.1	O	
WIM-C-041	Use decryption key (application level)	12.1.1	O	
WIM-C-042	Use non-repudiation key (application level)	12.1.1	O	
WIM-C-055	Use authentication key for TLS client authentication	12.1.1	O	
WIM-C-043	PIN handling Recommended PIN format	12.1.4.1	M	
WIM-C-044	Use WIM for digital signature verification RSA	5.3.3	O	
WIM-C-045	Use WIM for digital signature verification ECDSA	5.3.3	O	
WIM-C-046	PKCS#15 file path fields Support of 2-byte file identifiers	9.4.1	M	
WIM-C-047	PKCS#15 file path fields Support of absolute or relative paths	9.4.1	M	
WIM-C-048	Generate asymmetric key pair Handling	11.3.6.13	M	
WIM-C-049	Generate key assurance Handling	11.3.6.14	M	
WIM-C-140	Check and update TLS value	11.5	O	WIM-C-052

14.2.2 ME Use of WIM ICC

Item	Function	Reference	Status	Requirement
WIM-C-101	Direct application selection	11.3.3.1	M	
WIM-C-102	(Item removed)			
WIM-C-103	Logical channels. A ME that uses some other application on ICC WIM (eg, GSM SIM), MUST support logical channels.	11.3.2	O	WIM-C-104
WIM-C-104	ICC commands MANAGE CHANNEL	11.3.2	O	
WIM-C-105	ICC commands VERIFY	11.3.4.1	M	
WIM-C-106	ICC commands DISABLE VERIFICATION	11.3.4.2	M	
WIM-C-107	ICC commands ENABLE VERIFICATION	11.3.4.3	M	
WIM-C-108	ICC commands CHANGE REFERENCE DATA	11.3.4.4	M	
WIM-C-109	ICC commands RESET RETRY COUNTER	11.3.4.5	M	
WIM-C-110	ICC commands SELECT	11.3.5.1	M	
WIM-C-111	ICC commands READ BINARY	11.3.5.2	M	
WIM-C-112	ICC commands UPDATE BINARY	11.3.5.3	M	
WIM-C-113	ICC commands MANAGE SECURITY ENVIRONMENT	11.3.6.1	M	
WIM-C-114	ICC commands PERFORM SECURITY OPERATION (all but Key Transport and Key Agreement)	11.3.6.4 11.3.6.7, 11.3.6.8, 11.3.6.9, 11.3.6.10	M	WIM-C-130 OR WIM-C-131
WIM-C-130	ICC commands PERFORM SECURITY OPERATION Key Transport	11.3.6.4, 11.3.6.5	O	
WIM-C-131	ICC commands PERFORM SECURITY OPERATION Key Agreement	11.3.6.4, 11.3.6.6	O	
WIM-C-132	ICC commands GENERATE ASYMMETRIC KEY PAIR	11.3.6.13 11.4.9	M	
WIM-C-133	ICC commands GENRATE KEY ASSURANCE	11.3.6.14 11.4.10	M	
WIM-C-115	ICC commands ASK RANDOM	11.3.6.12	O	
WIM-C-116	ICC commands GET RESPONSE	11.3.6.16	M	
WIM-C-117	ICC size; at least one supported	11.1	M	WIM-C-118 OR WIM-C-119

WIM-C-118	ID-1	11.1	O	
WIM-C-119	ID-000 (Plug-in)	11.1	O	
WIM-C-120	Transmission protocols T=0	11.2	M	
WIM-C-121	Transmission protocols T=1	11.2	O	
WIM-C-122	Supply voltage, SIM card 3 V	11.2	M	
WIM-C-123	Supply voltage, SIM card 5 V	11.2	O	
WIM-C-124	Supply voltage, external card 3 V	11.2	M	
WIM-C-125	Supply voltage, external card 5 V	11.2	O	
WIM-C-126	Support access control rules	12.2	M	
WIM-C-127	Command modes A 3G terminal using USIM with SCP is REQUIRED to use the SCP mode with a USIM-WIM (in this case the card is guaranteed to support this). With an auxiliary ICC, the terminal is also recommended to use the SCP mode. Should the card not support it (e.g older version card), it would respond with an error indicating an incorrect CLA byte, after which the terminal can switch to the native mode.	11.3	M	WIM-C-128 OR WIM-C-129
WIM-C-128	Native mode	11.3	O	
WIM-C-129	SCP mode	11.3	O	
WIM-C-135	Supply voltage, SIM/USIM card 1.8 V	11.2	O	
WIM-C-136	Supply voltage, external card 1.8 V	11.2	O	

Appendix A. Change History (Informative)

A.1 Approved Version History

Reference	Date	Description
OMA-WAP-WIM-V1_1-20021024-C	15 June 2004	Part of WPKI package

A.2 Draft/Candidate Version 1.2 History

Document Identifier	Date	Sections	Description
Candidate Version OMA-WAP-WIM-V1_2-20050322-C	22 March 2005		TP ref: OBKG V1.1 - TP Ref: OMA-TP-2005-0091-OBKG-V1_0-for-Candidate-approval
Draft Version OMA-WIM-v1_2-20050202-D	2 February 2005		Version updated according to the consistency review.
Draft Version OMA-WIM-v1_2-20040401-C	1 April 2004		OMA-WAP-WIM-v1_1 in which were incorporated the OBKG features, On-Board Key Generation (Generate Asymmetric Key pair) and Generate Key Assurance.
Draft version OMA-WAP-WIM-V1_2-20040820-D	20 August 2004	2	Change of file name, to respect OMA naming conventions, corrections of references and formatting of the Change History according to OMA templates