DC2000 Security Target

(Common Criteria)

Thales e-Security Ltd
4th/5th Floors
149 Preston Road
Brighton
BN1 6AS

Tel: +44 (0) 1273 384600
Fax: +44 (0) 1273 384601

Document History

Issue	Date	Description
00A	6th May 2001	Initial Release
00B	29th May 2001	Updated following review
00C	31st May 2001	Updated following further comments
00D	6th November 2001	Corrected inconsistent version numbering
00E	3rd December 2001	Added: Demonstration of consistency between Environment assumptions and Environment Objectives . Brief description of DSA (section 8.1.1.3) Updated: Section 9.2 (Assurance Measures) with respect to changes in extent of possible re-use from previous evaluations. Statement on Evaluation Assurance Level to make the Security Target suitable for any Evaluation Assurance Level up to and including EAL 5 (section 11.2.4). References (section 2) to include part numbers for all referenced documentation. Corrected: Typographic errors.
00F	27th May 2002	Corrected and updated references section Updated references in Section 9.2. Changed the address on front page.
00G	19th August 2002	Added statement that the objectives are designed to meet the threats, as requested by Australian CB (DSD).
00H	29th January 2003	Updated following comments from Australian and UK CBs.
00I	4th February 2003	Updated following comments from CMG.
00J	6th March 2003	Updated following comments from UK CB and CMG. The main changes were as follows:

		Completed the separation of sections relevant to the DC2K and the SGSS throughout the document Moved section concerning unit management from section 9.1 to section 5 Included specification of the security functions in section 9.1 Moved the environment assumptions/objectives correlation to the security objectives rationale
00K	29th June 2004	Changes made as a result of CB review. In particular: Scope of TOE clarified Claims for distinct SGSS TOE removed
00L	2nd July 2004	Some further changes at the request of the CB: "CESG" removed from the glossary Clarifications in sections 5.2.2 and 9.1.3.2
00M	5th April 2005	Change of SGSS acronym definition

Contents

1 Glossary

CA	Certificate Authority
CAPS	CESG Assisted Products Scheme
CC	Common Criteria
DC2K	DataCryptor 2000
DEK	Data Encryption Key
FIPS	Federal Information Processing Standards
FPGA	Field-Programmable Gate Array
IP	Internet Protocol
KEK	Key Encryption Key
PCB	Printed Circuit Board
SFP	Security Function Policy
SGSS	Secure Generic Sub-System
TOE	Target of Evaluation
TSF	TOE Security Functions
TSP	TOE Security Policy

2 References

[1] CCIMB-99-031, CCIMB-99-032, CCIMB-99-033, Common Criteria Version 2.1 Parts 1, 2 and 3
[2] 0562a226, "DataCryptor 2000 Version Under Evaluation"
[3] 0562a227, "DataCryptor 2000 Protocols Under Evaluation"
[4] 0562a228, "DataCryptor 2000 Cryptographic Algorithms Under Evaluation"
[5] 0550a109, "Key Management Specification"
[6] 0520a103, "Engineering Configuration Management Procedure"
[7] 0562a245, "DataCryptor 2000 Configuration List"
[8] 0562a246, "DataCryptor 2000 Acceptance Plan"
[9] 0520a206, "StarTeam Version Control"
[10] 0530a101, "Company Documentation"
[11] 0530a108, "Software Tools"
[12] 0520a135, "Packing and Despatch"
[13] 1270a274, "DataCryptor 2000 Series – Installation Guide"
[14] 1270a275, "DataCryptor 2000 Series – User Guide"
[15] dc2000.sdt - DataCryptor 2000 SDL
[16] 0562a247, "DataCryptor 2000 Architectural Design"
[17] 0562a248, "DC2000 Representation Correspondence Analysis"
[18] 0562a243, "DC2000 Security Policy Model & Functional Specification/Security Policy Model Correspondence"
[19] 0562a249, "DataCryptor 2000 Life Cycle Model"
[20] 0530a100, "C Coding Standard"
[21] 0530a130, "VHDL Coding Standard"
[22] 0562a250, "DataCryptor 2000 Test Coverage Analysis"
[23] 0562a251, "DataCryptor 2000 Depth of Testing Analysis"
[24] 0558a254, "DataCryptor 2000 System Test Specification [Link]"
[25] Unused reference
[26] 0558a258, "DataCryptor 2000 Frame Relay Test Specification"
[27] Unused reference
[28] 0558a274, "DataCryptor 2000 IP Test Specification"
[29] X/1105/64 – 665, "DC2K Suite of Functional Tests (Issue 3.2)"
[30] X/1105/64 – 664, "Functional Tests Supplementary for v3.1HMG Unframed Link, IP and Frame Relay DC2K (Version 1.0)"
[31] X/1105/64 – 663, "Supplementary Test For v3.1HMG"
[32] 0562a203, "DataCryptor 2000 v3.11 Functional Test Report"
[33] 0562a252, "DC2000 Guidance Documentation Analysis"
[34] 0562a268, "DC2000 Descriptive Low-level Design"
[35] 0562a269, "DC2000 Semiformal High-level Design"
[36] 0562a236, "DataCryptor 2000 Cross Reference of Security Functions"
[37] 0562b152, "DataCryptor 2000 Security Target"
[38] 0562a205, "DC2000 Security Target (ComSec)"
[39] 0562a244, "DC2000 - Evaluation Assurance Level"

[40] 0562a253, "DataCryptor 2000 Vulnerabilities Analysis"
[41] 0562a254, "DC2000 Semiformal Functional Specification"
[42] 0562a276, "DC2000 Functional Testing (Common Criteria)"
[43] FIPS PUB 140-1, Security Requirements for Cryptographic Modules
[44] 1270A377, "DC2000 Evaluated Configuration (Common Criteria)"

3 Inter-Document References

The following terms defined in this Security Target may be referenced in other associated documentation. The identifiers *DC2K* and *SGSS* are used to distinguish between threats, objectives and functions that are relevant to the DC2K and the SGSS respectively.

3.1 Threats

T_DC2K_extract_data_from_secure_domain
T_DC2K_record_plaintext_data_from_insecure_domain
T_DC2K_cryptanalyse_data_within_insecure_domain
T_DC2K_access_to_secret_authentication_key
T_DC2K_access_to_secret_key_exchange_alg_keys
T_DC2K_access_to_algorithm_outside_unit
T_DC2K_cryptanalyse_keys_within_insecure_domain
T_DC2K_loss_of_commissioned_unit
T_DC2K_tamper_with_unit
T_DC2K_algorithm_replacement
T_DC2K_certificate_authority_replacement
T_DC2K_key_exchange_certificate_replacement

T_SGSS_access_to_keys_within_unit
T_SGSS_access_to_algorithm_within_unit
T_SGSS_application_replacement

3.2 Assets

A_user_data
A_user_key
A_user_algorithm

3.3 Security Objectives

OBT_DC2K_provide_data_confidentiality
OBT_DC2K_provide_secure_key_management
OBT_DC2K_provide_secure_algorithm_load
OBT_DC2K_provide_secure_CA_load
OBT_DC2K_provide_secure_key_exchange_keyset_load

OBT_SGSS_provide_resistance_to_physical_attack
OBT_SGSS_provide_secure_application_load

OBE_DC2K_transmit_data_through_TOE
OBE_DC2K_check_for_unit_tamper
OBE_DC2K_select_appropriate_settings

OBE_DC2K_management_centre

OBE_SGSS_protect_secure_domain
OBE_SGSS_protect_key_material
OBE_SGSS_protect_algorithms
OBE_SGSS_protect_keyed_unit

3.4 Security Functions

SF_DC2K_data_authentication_implementation
SF_DC2K_key_exchange_algorithm
SF_DC2K_encryption_algorithm

SF_SGSS_data_authentication_implementation
SF_SGSS_Random_Number_Generator
SF_SGSS_alarm_circuitry

4 Introduction

4.1 Security Target Identification

4.1.1 Security Target Information

Security Target Title: DataCryptor 2000 Security Target (Common Criteria)
Part Number: 0562A218
Version Number: As identified by Document History

4.1.2 DC2K Target of Evaluation Information

TOE Title: DataCryptor 2000
Part Number: 1600X320
Version Number: Specified in reference [2]
Evaluation to include: Communications protocols and cryptographic algorithms as specified in references [3] and [4] respectively.

The Security Target has been derived using [1].

4.2 Security Target Overview

This document provides a Security Target for the DataCryptor 2000 (DC2K).

4.2.1 Datacryptor 2000

The DataCryptor 2000 (DC2K) is a range of network encryption products that support several different network protocols (e.g. IP, Frame Relay etc.). The primary purpose of the product is to provide data confidentiality. It has been designed with flexibility in mind and provides a secure soft-upgrade capability to change the network protocol and cryptographic algorithms supported. The DC2K uses public key cryptography techniques to minimise the administrative overhead of key management, and implements sophisticated measures to resist physical attack in order to safeguard key material and sensitive algorithms.

This document describes the security requirements and operating assumptions of the DC2K TOE. Section 5 identifies the core DC2K functionality that forms the DC2K TOE, outlining the logical and physical boundaries of the TOE.

The assumed operational environment is discussed in section 6, as well as the perceived threats within that environment; a statement of the security objectives intended to counter such threats is provided in section 7.

Detailed IT security requirements are discussed in section 8 which is split into functional and assurance aspects. A Target of Evaluation Summary Specification is given in section 9, which provides a description of how the security functions and assurance measures are met by the DC2K.

Section 11 provides a rationale for the security target. In particular it describes the correlation between the security objectives and the threats arising from the environment, a justification of the suitability of the security requirements with respect to the security objectives, and finally the means by which TOE security functions and assurance measures meet the security requirements.

4.2.2 SGSS

The SGSS is a component of the DC2K. It confirms the authenticity of the DC2K application when loaded, provides a random number generator for use by the DC2K application, and includes alarm circuitry which ensures that the unit's sensitive contents are erased in the event of an alarm.

The SGSS is equally capable of supporting another application distinct from the DC2K application. For the benefit of the reader, the assumptions, threats, objectives, requirements and functions specified in sections 6, 7, 8 and 9 are identified as being relevant to the SGSS or specific to the DC2K. However all are relevant to the DC2K TOE.

4.3 Common Criteria Conformance

The TOE conforms to the Common Criteria within the meaning of [1] as follows:

Part 2 Conformant
Part 3 Conformant at the Evaluation Assurance Level specified in reference [39].

No claims are made with respect to conformance to any Protection Profile.

5 Target Of Evaluation Description

A description of the product is provided first. This gives a context to the TOE description that follows.

Note that assurance requirements for the DC2K TOE are specified by [39], whilst the TOE description below specifies the functional scope of the TOE relative to the product. Where [39] specifies multiple assurance requirements the number of implied TOEs increases accordingly.

5.1 Product Description

5.1.1 Product Type

The DataCryptor 2000 is a range of network encryption products.

5.1.2 Basic Purpose

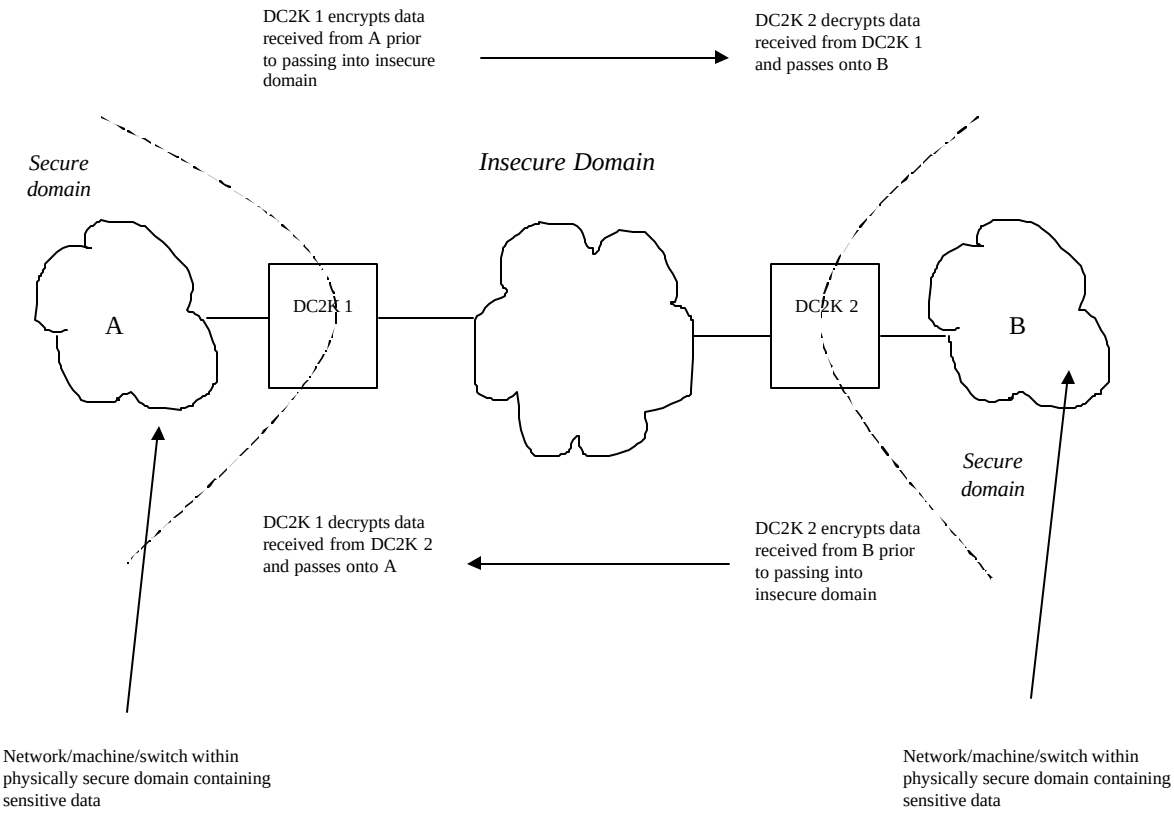


Figure 1 – DC2K usage

Figure 1 gives a pictorial representation of the primary purpose of the DataCryptor 2000. It shows a simple example of sensitive data being transmitted from one physically secured domain to another through a domain in which no physical security is assumed to be present.

At a fundamental level, the units operate in encrypting/decrypting pairs ensuring that confidentiality is afforded to data sent between the two. In this way, sensitive information may be secured whilst it is in an insecure domain. In fact, more typically, groups of DC2Ks are deployed in various configurations to support many different network topologies and protocols, with the data passed between each pair subject to the unit's confidentiality services.

The DC2K does not provide data security services to information whilst it remains within the secure domain as defined in figure 1.

5.1.3 Physical Description

Several host/network interface protocols are supported, and 2 unit management ports are provided. The unit has the following external interfaces:

- power connector
- host data port
- network data port
- an RS232 (serial) management data port
- a 10baseT Ethernet management data port
- front panel keyswitch
- erase button
- LEDs.

The standard unit consists of the Secure Module, also known as the Secure Generic Sub-System (SGSS), and the DC2K baseboard.

The SGSS contains all components relevant to the secure operation of the unit. The board is subjected to physical protection using a mesh and resin technique. Alarm circuitry provided within the SGSS detects intrusion and voltage attacks, as well as movement, extremes of temperature, and pressing the erase button, where the user enables these alarms. In the event of an alarm, all the unit's sensitive contents are erased.

The baseboard provides the power interface and basic communications support.

5.1.3.1 Peripherals

As well as the unit itself, the following items are supplied to the customer.

- external power supply
- network and host cables as necessary
- CD containing unit management software and product installation and user manuals

Network and host cables are supplied where necessary to provide the physical and electrical conversion required between the DataCryptor 2000's proprietary external connectors and the communications protocol used by the customer.

The management software (a standard Windows application) runs on a PC and provides the customer with the capability to configure the unit's security and communications settings as required.

5.1.4 Logical Description

The DataCryptor 2000 consists of the following modules:

- The SGSS application
- The DataCryptor 2000 application
- The encryption and key exchange algorithms which are externally loaded.

5.1.4.1 SGSS Software Application

The SGSS application runs on the SGSS. It consists of a secure bootstrap program that initialises the DC2K system and confirms the authenticity of the DataCryptor 2000 application subsequently loaded. Once the DataCryptor 2000 application has been loaded and its authenticity verified by the bootstrap, operational control passes from the SGSS application to the DC2K application.

5.1.4.2 DataCryptor 2000 Software Application

The DC2K application provides the following functions:

- Cryptographic verification of encryption algorithms, key exchange algorithms, Certificate Authorities and key exchange algorithm keysets
- Authentication of key exchange protocol, as described in [5], section 4.2.1, (validation of public data exchange)
- Unit to management centre communication protocol and data path
- Unit to unit communication protocol and data path support
- Interface to encryption and key exchange algorithms.

5.1.4.3 Key Exchange Algorithm

The externally loaded key exchange algorithm provides the unit with the capability to securely derive a shared key with another unit which can subsequently be used to encrypt and decrypt data transmitted between the two units.

5.1.4.4 Encryption Algorithm

The externally loaded encryption algorithm provides the DC2K with the capability to encrypt and decrypt data transmitted and received respectively.

5.1.5 Unit Management

A unit may be managed (i.e. its security and communications attributes changed) by use of a management centre. The DC2K and the management centre communicate in the same way that two units communicate; firstly a cryptographic key must be established according to the key management

protocol described in [5], and subsequently, all traffic sent between the management centre and the unit is subject to encryption.

To provide this functionality, the management centre has the encryption and key exchange capabilities that are equivalent to those of the DataCryptor 2000 unit. However, the unit (rather than the management centre) enforces security by failing to act on a management request if it cannot decrypt it.

In order to be able to successfully decrypt data sent to it by the management centre, both entities must be using the same key for data encryption. Similarly, in order to agree a common key, both entities must successfully complete the key exchange protocol. This requires the management centre to have access to key material that has been authorised by the same CA as the unit is operating under. Only authorised individuals should have access to such key material.

5.2 DC2K TOE Description

5.2.1 Logical Description

The TOE comprises a subset of the following core DC2K functionality. The subset is defined by section 9.1, which identifies aspects of the TOE security functions for which FIPS or CAPS results are quoted and which are thus excluded from the scope of CC evaluation.

- Cryptographic authentication of the DC2K application by the SGSS
- Cryptographic authentication of encryption algorithms, key exchange algorithms, Certificate Authorities and key exchange algorithm keysets
- Key exchange
- Encryption
- Generation of random numbers by the SGSS for use by DC2K cryptographic algorithms (together with associated diagnostic statistical testing performed by DC2K)
- Detection of and response to the following alarms by the SGSS:
  - Physical intrusion
  - Physical movement
  - High or low voltage
  - High or low temperature.

Cryptographic algorithms and associated key management protocols are specified by those sections of [4] and [5] identified in section 9.1 below. Encryption relates only to the protection of user traffic (and not management traffic).

The host/network interface protocols supported are specified by [3]. Where the above functionality varies according to the communications protocol, the variants are included in the TOE.

Other product functionality excluded from the TOE includes:

- Unit management (however, some specific unit management functionality is used, from a management centre and via the unit's Ethernet management port, in accordance with the installation, commissioning and configuration procedure [44] to achieve the evaluated configuration)
- Use of the front panel keyswitch and erase button
- Use of the communications ports, other than in respect of the cryptographic protection given to user traffic (e.g. remote monitoring via the network port, cryptographic protection of management port traffic)
- Unit status indications given by the LEDs
- Use of multiple user groups (communication within such a group being authenticated by the group Certificate Authority)
- Forced Standby mode (requiring password authentication after power on of the unit)
- Hot Standby functionality.

See also the note, given in the following section, concerning interaction between security functions.

5.2.2 Physical Description

Physically the TOE comprises only those processing and circuitry elements of the product specifically required for operation of the logical TOE functionality specified in the above section.

Logical product functionality excluded from the TOE thus includes interaction between the security functions, other than the interactions between the security functions specified in: (9.1.5 and 9.1.6), (9.1.3.1 and 9.1.5) and (9.1.3.1 and 9.1.6).

Thus, for example, whilst the TOE includes functionality to authenticate a Certificate Authority, the functionality to then load the authenticated Certificate Authority to its place of subsequent use is excluded.

6 Target of Evaluation Security Environment

6.1 Assumptions for the Security Environment of the DC2K

This section describes assumptions concerning the security environment of the DC2K.

6.1.1 Assumed Usage

The assumed usage of the DataCryptor 2000 is as shown in figure 1 of section 5.1.2 to provide data confidentiality to the user's information and data assets.

Additionally, it is assumed that the DataCryptor 2000's data confidentiality capability will be applied to all sensitive data to be passed between two secure domains over an insecure domain, and that appropriate policies exist with respect to:

- Choice of Key lifetime – use of keys for prolonged periods may allow cryptanalysis
- Action in the event of suspected tampering, loss or theft of unit.

Specific advice on these aspects should be sought from the appropriate security authority.

6.1.1.1 Limitations of Use

The DataCryptor 2000 does not provide any protection to data whilst it resides in the secure environment as defined in figure 1 of section 5.1.2. Neither does it provide any protection where data is transmitted from the secure environment that does not pass through the unit, or where the encryption capability has not been applied.

Therefore, it is assumed that:

- The secure environment is protected to a suitable level by appropriate means
- All sensitive data is transmitted through the DataCryptor 2000
- The installation, commissioning and configuration procedure for the evaluated configuration (see [44]) has been followed. Note that this includes setting the temperature and motion alarms.

6.1.2 Protection of Assets

The DataCryptor 2000 provides confidentiality to the user's information assets where those assets have a value of up to £500,000.

6.1.3 Assumed Environment of Operation

6.1.3.1 Management

Within the evaluated configuration, management of the unit is restricted to use of the specified installation, commissioning and configuration procedure [44], which assumes the use of the

appropriate Element Manager management software (supplied with the DC2K) and a management centre PC on which to run such software. It is further assumed that the management centre itself is operated by trusted personnel in a physically secure environment.

6.1.3.2 Physical Protection Measures (DC2K)

It is assumed that physical security measures are applied to information *within the secure domain only* as appropriate to the value of the data being protected. As well as the data itself, such physical protection should be afforded to:

- Any Key material which is held externally to the unit
- Any sensitive key exchange or encryption algorithm held externally to the unit
- The unit itself while keyed
- The management centre (and any network to which the management centre is connected)

6.1.3.3 Connectivity

It is assumed that during normal operation, the unit is connected in an equivalent manner to that shown in figure 1 of section 5.1.2. In addition, it is assumed that when unit installation, commissioning and configuration is required, a separate physically secure connection is made from the management centre to the unit's Ethernet management port. Otherwise it is assumed that no other connections are made to the unit's Ethernet or serial management ports.

6.1.3.4 Personnel

It is assumed that:

- Administrative personnel who install the unit in accordance with the specified installation, commissioning and configuration procedure are trusted appropriately.
- Access to keyed units is only provided to trusted administrative personnel.
- Access to key material is only provided to trusted administrative personnel.
- Access to sensitive algorithms is only provided to trusted administrative personnel.

In addition it is assumed that administrative personnel have the necessary skill to operate a standard Windows application.

Note that in the context of this TOE, basic "users" of the product are those people whose data is protected by it. Since the DataCryptor 2000 is a network encryptor, the user does not have any direct interaction with the TOE – instead, administrative personnel control all product configuration and operation. As such, there are no requirements or assumptions placed on users themselves.

6.2 Assumptions for the Security Environment of the SGSS

This section describes assumptions concerning the security environment of the SGSS.

6.2.1 Physical protection measures (SGSS)

It is assumed that physical security measures are applied to:

- The secure domain in which unprotected data resides
- Sensitive key material that is stored externally to the SGSS
- Sensitive algorithms that are stored externally to the SGSS
- SGSSs that have been commissioned with key material.

6.3 Threats to the DataCryptor

This section describes threats to the DC2K. Note that the threats to the SGSS described in section 6.4 are also threats to the DC2K since the SGSS is a component of the DC2K.

6.3.1 Extraction of Data from Within the Secure Domain

Inter-document reference T_DC2K_extract_data_from_secure_domain

The DC2K does not provide any security services to data whilst it resides within the secure domain. Instead it operates by encrypting any data passed into its host port from the host network, machine or switch within the secure domain, prior to passing the encrypted data out into the insecure domain.

A threat exists whereby a threat agent could gain access to unencrypted data whilst it resides within this area.

6.3.1.1 Attack

The attack is to gain physical access to the secure domain and record data from within that domain.

6.3.1.2 Asset

Inter-document Reference – A_user_data

The asset under threat is the user's data.

6.3.1.3 Threat Agent

Expertise:	A low level of expertise is required. Data within this domain is not subject to any technical protection, but the threat agent will require the capability to extract the data and decode it from standard network protocols. Such decoding is within the capability of any individual with a basic understanding of common, public domain network protocols.
Resource:	Limited resource is required.

	Some means of recording and decoding the data from the secure domain is required. However, such tools are available, and, depending on the protocol, are relatively cheap. IP recording and decoding tools for example are primarily simple software Windows applications.
Motivation:	Since the value of the asset is relatively high, it must be assumed that motivation to mount this attack is high.
Vulnerabilities Exploited:	Within the definition of the threat, the asset is not subject to the protection of the TOE. Therefore no TOE vulnerabilities need be exploited in order to mount a successful attack.
Opportunity:	If the threat agent has obtained access to the secure domain (either by virtue of authorisation, or by breaching physical security measures), opportunity to mount the attack is high. Otherwise, opportunity is extremely limited.

6.3.2 Recording of Plaintext Data Leaked into Insecure Domain

Inter-document reference T_DC2K_record_plaintext_data_from_insecure_domain

Unencrypted data may be present in the insecure domain for two reasons:

- The data has not been sent via a channel that passes through the DC2K prior to entering the insecure domain.
- The data has been sent through a channel that passes through the DC2K, but the DC2K's encryption capability has not been applied to the data. i.e. the unit's line mode has been set to an insecure mode.

A threat exists whereby a threat agent records such unencrypted data from the insecure domain.

6.3.2.1 Attack

The attack is to record any unencrypted data from the insecure domain.

6.3.2.2 Asset

Inter-document Reference – A_user_data

The asset under threat is the user's data.

6.3.2.3 Threat Agent

Expertise:	A low level of expertise is required.

	Unencrypted data within this domain is not subject to any technical protection, but the threat agent will require the capability to extract the data and decode it from standard network protocols. Such decoding is within the capability of any individual with a basic understanding of common, public domain network protocols.
Resource:	Limited resource is required. Some means of recording and decoding the data from the insecure domain is required. However, such tools are available, and, depending on the protocol, are relatively cheap. IP recording and decoding tools for example are primarily simple software Windows applications.
Motivation:	Since the value of the asset is relatively high, it must be assumed that motivation to mount this attack is high.
Vulnerabilities Exploited:	Within the definition of the threat, the asset has not been subject to the protection of the TOE. Therefore no TOE vulnerabilities need be exploited in order to mount a successful attack.
Opportunity:	Where such unencrypted data is present in the insecure domain, the opportunity for the threat agent to mount this attack is high.

6.3.3 Cryptanalysis of data in the insecure domain

Inter-document Reference – T_DC2K_cryptanalyse_data_within_insecure_domain

In normal usage, it is anticipated that sensitive data residing within the secure domain will be subjected to the TOE's data confidentiality measures prior to it being passed into an insecure domain.

A threat exists for an attacker to record encrypted data sent across the insecure domain and subject it to cryptanalysis in an attempt to discover the underlying plaintext data.

6.3.3.1 Attack

The attack is to record encrypted data from the insecure domain, decode it and perform cryptanalysis to reveal the underlying plaintext data.

6.3.3.2 Asset

Inter-document Reference – A_user_data

The asset under threat is the user's data.

6.3.3.3 Threat Agent

Expertise:	Assuming that an appropriate encryption algorithm is used, its implementation is not flawed, and that key material has not been leaked, a high level of expertise is required to successfully gain plaintext from encrypted data.
Resource:	The resource requirements to mount an attack of this type are high – a very large amount of computing power, either distributed or within one unit would be required.
Motivation:	Since the value of the asset is relatively high, it must be assumed that motivation to mount this attack is high.
Vulnerabilities Exploited:	If a vulnerability were present in the TOE's encryption algorithm or in its implementation, this may be exploited to decrease the level of expertise or resource required for success.
Opportunity:	Where such encrypted data is present in the insecure domain, the opportunity for the threat agent to mount this attack is high.

6.3.4 Exposure of Secret Authentication Key

Inter-document Reference – T_DC2K_access_to_secret_authentication_key

Secret authentication keys (although not loaded into the unit) are used to generate signed key exchange certificates.

If it were possible for an attacker to gain access to such material as it exists externally to the unit, it may be possible for him to ultimately determine the key used to encrypt the user's data, and then use the key to decrypt encrypted data.

There are two means by which access may be provided to a threat agent:

- An individual with authorised access to the material may leak the data intentionally or unintentionally
- An unauthorised individual may breach physical measures to gain access to the material.

6.3.4.1 Attack

A threat agent who has gained access to the secret authentication key may forge signed key exchange certificates. This would allow an active "man-in-the-middle" attack to be mounted between two units whereby both units are spoofed into establishing a shared key encryption key with the threat agent rather than the other unit. A similarly shared data encryption key could then be generated using the rogue key encryption key, and then used to decrypt the user's data.

6.3.4.2 Asset

Inter-document Reference – A_user_key
A_user_data

The asset under threat is the user group secret authentication key, exposure of which may ultimately lead to exposure of the user's data.

6.3.4.3 Threat Agent

Expertise:	A high level of expertise is required. Even assuming successful access to the secret authentication key, the attack is a very sophisticated real-time active attack. It requires insertion and deletion of data from the line between two units without either unit "noticing" the presence of the third party.
Resource:	A high level of resource is required. Equipment to insert and remove traffic from the line in real-time is required, as is equipment which can spoof the entire key exchange protocol, and subsequently react appropriately to any peer-unit requests i.e. subsequent data encryption key updates in a timely fashion.
Motivation:	Since the value of the asset is relatively high, it must be assumed that motivation to mount this attack is high. In particular, repeated application of this attack could give rise to the successful decryption of traffic within the entire lifetime of the secret authentication key.
Vulnerabilities Exploited:	Within the definition of the threat, the asset has not been subject to the protection of the TOE. Therefore no TOE vulnerabilities need be exploited in order to mount a successful attack.
Opportunity:	If the threat agent has obtained access to the secure domain (either by virtue of authorisation, or by breaching physical security measures), opportunity to mount the attack is high. Otherwise, opportunity is extremely limited.

6.3.5 Exposure of Secret Keys Used in the Key Exchange Algorithm

Inter-document Reference – T_DC2K_access_to_secret_key_exchange_alg_keys

In some modes of use, secret key exchange algorithm keys are loaded into the unit from an external source.

If it were possible for an attacker to gain access to such material as it exists externally to the unit, it may be possible for him to ultimately determine the key used to encrypt the user's data, and then use the key to decrypt encrypted data.

There are three means by which access may be provided to a threat agent:

- An individual with authorised access to the material may leak the data intentionally or unintentionally
- An unauthorised individual may breach physical measures to gain access to the material
- The unit may be commissioned with secret key material over an unprotected link or network, to which a threat agent may have access.

6.3.5.1 Attack

Long term secret keys are input by both units participating in the KEK derivation algorithm, and in some modes of operation, these are loaded from an external source. However, the algorithm also utilises relatively substantial quantities of one-time random data generated by the unit's themselves. This random data is never exposed outside the unit. Assuming that the units' random number generator is operating properly, a threat agent would have to guess (or exhaust over) these random values to be able to determine the key encryption key and subsequently the data encryption key used.

6.3.5.2 Asset

Inter-document Reference – A_user_key
A_user_data

The asset under threat is the user's secret key exchange algorithm keys, exposure of which may subsequently lead to exposure of the user's data.

6.3.5.3 Threat Agent

Expertise:	A high level of expertise is required. The threat agent would have to efficiently exhaust over all possible values of one-time random input to the key exchange algorithm to be able to determine the key encryption key established between the two units, and subsequently determine the data encryption key.
Resource:	An extremely high level of resource is required. Assuming that at least one of the unit's random number generators is operating correctly, a huge amount of computing resource would be required to exhaust over all possible one-time random inputs to the algorithm.

Motivation:	Since the value of the asset is relatively high, it must be assumed that motivation to mount this attack is high. In particular, repeated application of this attack could give rise to the successful decryption of traffic within the entire lifetime of the secret key exchange algorithm key.
Vulnerabilities Exploited:	Within the definition of the threat, the asset has not been subject to the protection of the TOE. Therefore no TOE vulnerabilities need be exploited in order to mount a successful attack.
Opportunity:	If the threat agent has obtained access to the secure domain (either by virtue of authorisation, or by breaching physical security measures), or to an insecure commissioning session, opportunity to mount the attack is high. Otherwise, opportunity is extremely limited.

6.3.6 Compromise of Sensitive Cryptographic Algorithm when external to unit

Inter-document Reference – T_DC2K_access_to_algorithm_outside_unit

In some cases, the cryptographic algorithms used to provide data and key confidentiality services to the user are sensitive. Exposure of such a sensitive cryptographic algorithm when it is stored externally to the unit may be undesirable for political reasons, and possibly assists future cryptanalysis.

6.3.6.1 Asset

Inter-document Reference – A_user_algorithm

The asset under threat is the user's sensitive cryptographic algorithms.

6.3.6.2 Attack

The attack is to gain access to the user's sensitive cryptographic algorithms, wherever those may be stored, with intent to gain expertise in cryptanalysis of the user's communications.

6.3.6.3 Threat Agent

Expertise:	A low level of expertise is required. When stored externally to the unit, the algorithm is not subject to any technical protection
Resource:	Limited resource is required.

Motivation:	Since the value of the asset is high, it must be assumed that motivation to mount this attack is high. In particular, information gained from this attack could be used to determine information about protective measures applied by other secure applications owned by the user.
Vulnerabilities Exploited:	Within the definition of the threat, the asset has not been subject to the protection of the TOE. Therefore no TOE vulnerabilities need be exploited in order to mount a successful attack.
Opportunity:	If the threat agent has obtained access to the secure domain (either by virtue of authorisation, or by breaching physical security measures), opportunity to mount the attack is high. Otherwise, opportunity is extremely limited.

6.3.7 Cryptanalysis of Encrypted Keys in the insecure domain

Inter-document Reference – T_DC2K_cryptanalyse_keys_within_insecure_domain

Two communicating DataCryptor 2000s must undertake a key exchange protocol prior to exchanging encrypted data. Full details of the key exchange protocols are provided in sections 4.2.2 (steps 1 – 8) and 4.1.1 (steps 1 – 5) of [5]; the use of public key cryptography allows two commissioned units operating within the same user group to negotiate a shared key encryption key, and subsequently a data encryption key. These protocols operate in such a way that there is no requirement for any secret data to be transmitted from either unit.

A threat exists whereby an attacker may perform cryptanalysis on the key exchange protocols to determine the key encryption keys and or data encryption keys subsequently used by the unit. An alternative attack may be to force the re-use of a key, possibly leading to easier cryptanalysis of encrypted data.

6.3.7.1 Attack

The attack is to perform cryptanalysis on the DC2K's key exchange protocols with intent to use key information to decrypt traffic transmitted between the units.

6.3.7.2 Asset

Inter-document Reference – A_user_key
A_user_data

The asset under threat is the user's key encryption keys and data encryption keys.

6.3.7.3 Threat Agent

Expertise:	Assuming that an appropriate key exchange and key encryption algorithms are used, and their implementations are not flawed, a high level of expertise is required to successfully gain keys from the key exchange protocols.
Resource:	The resource requirements to mount an attack of this type are high – a very large amount of computing power, either distributed or within one unit would be required.
Motivation:	Since the value of the asset is relatively high, it must be assumed that motivation to mount this attack is high. Access to the unit's keys could potentially lead plaintext for the lifetime of the key exposed.
Vulnerabilities Exploited:	If a vulnerability were present in the TOE's key exchange or key encryption algorithms or implementations, this may be exploited to decrease the level of expertise or resource required for success.
Opportunity:	Where such encrypted keys are present in the insecure domain, the opportunity for the threat agent to mount this attack is high.

6.3.8 Unit theft or loss

Inter-document Reference – T_DC2K_loss_of_commissioned_unit

A commissioned unit that is subsequently lost or stolen has all the necessary keys in place to engage in an encrypted session with another unit that may still be encrypting legitimate user data. If the unit's disappearance goes unnoticed for a period of time, the user may unknowingly be sending his information to an attacker.

6.3.8.1 Attack

The attack consists of a threat agent gaining access to a commissioned unit an using the unit to decrypt traffic sent to it by a unit still within the user's possession.

6.3.8.2 Asset

Inter-document Reference – A_user_data

The asset under threat is the user's data.

6.3.8.3 Threat Agent

Expertise:	A low level of expertise is required to steal the box, and to subsequently use it to decrypt traffic transmitted by another unit within the same user group.

Resource:	The resource requirement to mount an attack of this type is low. No specialist equipment is required.
Motivation:	Since the value of the asset is relatively high, it must be assumed that motivation to mount this attack is high.
Vulnerabilities Exploited:	Within the definition of the threat, the asset has not been subject to the protection of the TOE. Therefore no TOE vulnerabilities need be exploited in order to mount a successful attack.
Opportunity:	If the threat agent has obtained access to the secure domain (either by virtue of authorisation, or by breaching physical security measures), or a user has simply lost a unit within the insecure domain, opportunity to mount the attack is high. Otherwise, opportunity is extremely limited.

6.3.9 Unit tampering

Inter-document Reference – T_DC2K_tamper_with_unit

It may be possible to tamper with a unit that is transmitting encrypted data in such a way that the security provided by the unit is undermined. This would be achieved by forcing the box to output plaintext directly from the host port into the insecure domain. If such tampering were to go unnoticed, a large amount of data could be leaked.

6.3.9.1 Attack

The attack consists of a threat agent gaining access to a unit and tampering with it so as to cause plaintext data to be leaked into the insecure domain. Such unprotected data could then be recorded from within the insecure domain.

6.3.9.2 Asset

Inter-document Reference – A_user_data

The asset under threat is the user's data.

6.3.9.3 Threat Agent

Expertise:	A moderate level of expertise is required to alter the box in such a way that it causes insecure operation and the tampering goes unnoticed by the user.
Resource:	The resource requirement for such an attack is moderate – specialist equipment may be required to alter the box in an unnoticeable fashion.

Motivation:	Since the value of the asset is relatively high it must be assumed that motivation to mount this attack is high.
Vulnerabilities Exploited:	Within the definition of the threat, the asset has not been subject to the protection of the TOE. Therefore no TOE vulnerabilities need be exploited in order to mount a successful attack.
Opportunity:	If the threat agent has obtained access to the secure domain (either by virtue of authorisation, or by breaching physical security measures), opportunity to mount the attack is high. Otherwise, opportunity is extremely limited.

6.3.10 Loading of Malicious Encryption or Key Exchange Algorithm

Inter-document Reference T_DC2K_algorithm_replacement

The DataCryptor 2000's encryption and key exchange algorithms are soft-loaded under the cryptographic control of the application. If those algorithms were replaced by rogue algorithms that performed poor (or non-existent) data or key encryption, the user's keys and data may be exposed.

6.3.10.1 Asset

Inter-document Reference: *A_user_data*
A_user_key

The assets at threat from this attack are the user's data and keys.

6.3.10.2 Attack

The attack requires the generation of substitute algorithms that induces insecurity into the system. In addition, the algorithms must be formatted and digitally signed by the secret authentication key corresponding to the public key embedded in the DataCryptor's application code. Having generated such algorithms, the threat agent also needs to be able to load them into the unit.

6.3.10.3 Threat Agent

Expertise:	Assuming that the data authentication algorithm and its implementations are not flawed, and that the secret algorithm authentication key is unavailable, a high level of expertise is required to successfully generate an algorithm that will be verified by the DC2K application.
Resource:	The resource requirements to mount an attack of this type are extremely high – a very large amount of computing power, either distributed or within one unit would be required to generate an

	algorithm whose authenticity would be verified by the DC2K application. Furthermore, the only way to achieve the attack is by iteratively generating *and attempting to load* the algorithms into a unit.
Motivation:	Since the value of the asset is relatively high, it must be assumed that motivation to mount this attack is high. A sufficiently insecure algorithm might yield plaintext and keys to the threat agent.
Vulnerabilities Exploited:	If a vulnerability were present in the DC2K's algorithm authentication implementation, or in the data authentication algorithm used, this may be exploited to decrease the level of expertise or resource required for success.
Opportunity:	Where an attacker has access to the unit into which to load a rogue algorithm (either by virtue of being provided with authorised access, or by breaching physical security measures), the opportunity for the threat agent to mount this attack is present. Otherwise, opportunity is extremely limited.

6.3.11 Loading of Certificate Authorities Known to the Attacker

Inter-document Reference T_DC2K_certificate_authority_replacement

The DataCryptor 2000's Certificate Authorities are signed and loaded under the cryptographic control of the application. If this key material were replaced with equivalent key material known to the threat agent *in two units*, it may be possible for him subsequently to load consistent known or degenerate key exchange keysets into both units. Ultimately this attack might lead to the threat agent being able to determine the key used to encrypt the user's data, and use this key to decrypt encrypted data.

6.3.11.1 Attack

Long term public and secret keys are input by both units participating in the KEK derivation algorithm. However, the algorithm also utilises relatively substantial quantities of one-time random data generated by the unit's themselves. This random data is never exposed outside the unit. Assuming that the units' random number generator is operating properly, a threat agent would have to guess (or exhaust over) these random values to be able to determine the key encryption key and subsequently the data encryption key used.

6.3.11.2 Asset

Inter-document Reference – A_user_key

The asset directly under threat is the integrity of the user's Certificate Authority, alteration of which may subsequently lead to the exposure of key encryption keys, data encryption keys, and finally exposure of the user's data.

6.3.11.3 Threat Agent

Expertise:	A high level of expertise is required. Firstly, the threat agent would have to generate a signed Certificate Authority whose authenticity would be verified by the DC2K application, and load it into the unit, together with known key exchange keysets authorised by that CA. Secondly, even having achieved this, he would have to efficiently exhaust over all possible values of one-time random input to the key exchange algorithm to be able to determine the key encryption key established between the two units, and subsequently determine the data encryption key.
Resource:	An extremely high level of resource is required both to generate the signed CA and to determine the one-time random input to the key encryption key generation process.
Motivation:	Since the value of the asset is relatively high, it must be assumed that motivation to mount this attack is also high. In particular, repeated application of this attack could give rise to the successful decryption of traffic within the entire lifetime of the CA.
Vulnerabilities Exploited:	If a vulnerability were present in the DC2K's certificate authentication implementation, or in the data authentication algorithm used, this may be exploited to decrease the level of expertise or resource required for success.
Opportunity:	Where an attacker has access to the unit into which to load a replacement CA and keyset (either by virtue of being provided with authorised access, or by breaching physical security measures), the opportunity for the threat agent to mount this attack is present. Otherwise, opportunity is extremely limited.

6.3.12 Loading of Key Exchange Certificates Known to the Attacker

Inter-document Reference T_DC2K_key_exchange_certificate_replacement

The DataCryptor 2000's Key Exchange public key certificates (and in some instances the corresponding secret key) are signed and loaded under the cryptographic control of the application. If this keyset were replaced with equivalent key material known to the threat agent, it may be

possible for him to determine the key encryption key and data encryption key, and finally use this key to decrypt encrypted data.

6.3.12.1 Attack

Long term public and secret keys are input by both units participating in the KEK derivation algorithm. However, the algorithm also utilises relatively substantial quantities of one-time random data generated by the units themselves. This random data is never exposed outside the unit. Assuming that the unit's random number generator is operating properly, a threat agent would have to guess (or exhaust over) these random values to be able to determine the key encryption key and subsequently the data encryption key used.

6.3.12.2 Asset

Inter-document Reference – A_user_key

The asset directly under threat is the user's key exchange algorithm keys, exposure of which may subsequently lead to key encryption keys, data encryption keys, and finally exposure of the user's data.

6.3.12.3 Threat Agent

Expertise:	A high level of expertise is required. Firstly, the threat agent would have to generate a signed key exchange keyset whose authenticity would be verified by the DC2K application, and load it into the unit. Secondly, even having achieved this, he would have to efficiently exhaust over all possible values of one-time random input to the key exchange algorithm to be able to determine the key encryption key established between the two units, and subsequently determine the data encryption key.
Resource:	An extremely high level of resource is required both to generate the signed keyset and to determine the one-time random input to the key encryption key generation process.
Motivation:	Since the value of the asset is relatively high, it must be assumed that motivation to mount this attack is also high. In particular, repeated application of this attack could give rise to the successful decryption of traffic within the entire lifetime of the key exchange certificates.
Vulnerabilities Exploited:	If a vulnerability were present in the DC2K's key exchange keyset authentication implementation, or in the data authentication algorithm used itself, this may be exploited to decrease the level of expertise or resource required for success.

Opportunity:	Where an attacker has access to the unit into which to load a replacement keyset (either by virtue of being provided with authorised access, or by breaching physical security measures), the opportunity for the threat agent to mount this attack is present. Otherwise, opportunity is extremely limited.

6.4 Threats to the SGSS

This section describes threats to the SGSS. As already noted threats to the SGSS are also threats to the DC2K.

6.4.1 Discovery or Substitution of any Key Material Stored within Unit

Inter-document Reference – T_SGSS_access_to_keys_within_unit

Most secret key material (i.e. key encryption keys and data encryption keys) is generated and stored internally by the DC2K unit. If it were possible for a threat agent to discover such key values or substitute them for values known to him, it may be possible for that information to be used in the decryption of user data.

6.4.1.1 Attack

The attack is to gain access to the unit's sensitive storage areas and extract some or all of their contents without triggering an alarm (which would cause the sensitive contents to be erased).

6.4.1.2 Asset

Inter-document Reference – A_user_key
A_user_data

The asset under threat is potentially all of the user's secret key material, exposure of which may subsequently lead to exposure of the user's data.

6.4.1.3 Threat Agent

Expertise:	A high level of expertise is required. The threat agent would have to extract keys from the unit without triggering an alarm.
Resource:	A high level of resource is required. The threat agent would require sophisticated and specialised equipment to breach the unit's physical protection mechanisms.

	Such equipment might include X-ray capability, extremely fine drills, chemical solvents etc.
Motivation:	Since the value of the asset is relatively high, it must be assumed that motivation to mount this attack is high. In particular, information gained from this attack could be used to determine keys and hence traffic for the lifetime of the key or keys extracted.
Vulnerabilities Exploited:	If a vulnerability were present in the TOE's physical protection design or implementation, this may be exploited to decrease the level of expertise or resource required for success.
Opportunity:	Opportunity to undertake this attack is limited by the availability of access to the unit itself.

6.4.2 Extraction of Sensitive Cryptographic Algorithm From Unit

Inter-document Reference – T_SGSS_access_to_algorithm_within_unit

In some cases, the cryptographic algorithms used to provide data and key confidentiality services to the user are sensitive. Extraction of such a sensitive cryptographic algorithm from the unit may be undesirable for political reasons, and possibly assists future cryptanalysis.

6.4.2.1 Attack

The attack is to gain access to the unit's sensitive storage areas and extract some or all of their contents without triggering an alarm (which would cause the sensitive contents to be erased).

6.4.2.2 Asset

Inter-document Reference – A_user_algorithm

The asset under threat is the user's sensitive cryptographic algorithms.

6.4.2.3 Threat Agent

Expertise:	A high level of expertise is required. The threat agent would have to extract the algorithm from the unit without triggering an alarm.
Resource:	A high level of resource is required. The threat agent would require sophisticated and specialised equipment to breach the unit's physical protection mechanisms. Such equipment might include X-ray capability, extremely fine drills, chemical solvents etc.

Motivation:	Since the value of the asset is high, it must be assumed that motivation to mount this attack is high. In particular, information gained from this attack could be used to determine information about protective measures applied by other secure applications owned by the user.
Vulnerabilities Exploited:	If a vulnerability were present in the TOE's physical protection design or implementation, this may be exploited to decrease the level of expertise or resource required for success.
Opportunity:	Opportunity to undertake this attack is limited by the availability of access to the unit itself.

6.4.3 Loading of Malicious Application Code

Inter-document Reference T_SGSS_application_replacement

Amongst other security critical functions, the DataCryptor 2000's application code controls the cryptographic protection measures that are provided to the user's data. If that application were replaced by a rogue application that subverted the security provided by the application, it is possible that data, keys and algorithms could all be exposed.

6.4.3.1 Asset

Inter-document Reference: *A_user_data*
A_user_key
A_user_algorithm

The assets at threat from this attack are the user's data, keys and algorithms

6.4.3.2 Attack

The attack requires the generation of a substitute application that induces insecurity into the system. In addition, the application must be formatted and digitally signed by the secret authentication key corresponding to the public key embedded in the SGSS's secure bootstrap code. Having generated such an application, the threat agent also needs to load it into the unit.

6.4.3.3 Threat Agent

Expertise:	Assuming that the data authentication algorithm and its implementations are not flawed, and that the code secret authentication key is unavailable, a high level of expertise is required to successfully generate an application that will be verified by the SGSS's bootstrap.

Resource:	The resource requirements to mount an attack of this type are extremely high – a very large amount of computing power, either distributed or within one unit would be required to generate an application whose authenticity would be verified by the SGSS. Furthermore, the only way to achieve the attack is by iteratively generating *and attempting to load* the application into a unit.
Motivation:	Since the value of the asset is relatively high, it must be assumed that motivation to mount this attack is high. A sufficiently insecure application might yield plaintext, keys and algorithms to the threat agent.
Vulnerabilities Exploited:	If a vulnerability were present in the SGSS's secure bootstrap implementation, or in the data authentication algorithm used, this may be exploited to decrease the level of expertise or resource required for success.
Opportunity:	Where an attacker has access to the unit into which to load a rogue application (either by virtue of being provided with authorised access, or by breaching physical security measures), the opportunity for the threat agent to mount this attack is present. Otherwise, opportunity is extremely limited.

6.5 Organisational Security Policies

No claims are made regarding the TOE's compliance with specific organisational security policies.

7 Security Objectives

7.1 Security Objectives for the Target of Evaluation

Sections 7.1.1 and 7.1.2 list the security objectives for the DC2K and SGSS respectively.

Note that a security objective for the SGSS is also a security objective for the DC2K since the SGSS is a component of the DC2K.

Note also that section 9.1 specifies aspects of the TOE security functions for which FIPS or CAPS results are quoted and which are thus excluded from the scope of CC evaluation. The scoping specifications of section 9.1 thus also qualify the TOE objectives below.

7.1.1 DC2K Security Objectives

7.1.1.1 Provision of Data Confidentiality Service

Inter-document Reference OBT_DC2K_provide_data_confidentiality

The DC2K shall provide the option of a confidentiality service to all data that is transmitted through it.

7.1.1.2 Provision of Secure Key Management Service

Inter-document Reference OBT_DC2K_provide_secure_key_management

The DC2K shall provide a means of securely exchanging key material for use in the provision of data confidentiality.

7.1.1.3 Provision of Secure Algorithm Loading Capability

Inter-document Reference OBT_DC2K_provide_secure_algorithm_load

The DC2K shall provide a means by which the authenticity of a cryptographic algorithm may be itself cryptographically verified prior to its loading and usage.

7.1.1.4 Provision of Secure Certificate Authority Loading Capability

Inter-document Reference OBT_DC2K_provide_secure_CA_load

The DC2K shall provide a means by which the authenticity of a Certificate Authority may be cryptographically verified prior to its loading and usage.

7.1.1.5 Provision of Secure Key Exchange Keyset Loading Capability

Inter-document Reference OBT_DC2K_provide_secure_key_exchange_keyset_load

The DC2K shall provide a means by which the authenticity of a Key Exchange Keyset may be cryptographically verified prior to its loading and usage.

7.1.2 SGSS Security Objectives

7.1.2.1 Provision of Physical Security Measures to Sensitive Data Stored Within TOE

Inter-document Reference OBT_SGSS_provide_resistance_to_physical_attack

The SGSS shall provide physical resistance to direct technical attack aimed at the extraction of sensitive data from within the unit.

7.1.2.2 Provision of Secure Application Loading Capability

Inter-document Reference OBT_SGSS_provide_secure_application_load

The SGSS shall provide a means by which the authenticity of a DataCryptor 2000 application may be cryptographically verified prior to its loading and storage.

7.2 Security Objectives for the Environment

This section lists the security objectives for the environment of the DC2K and the SGSS.

Note that a security objective for the environment of the SGSS is also a security objective for the environment of the DC2K.

7.2.1 Security Objectives for the Environment of the DC2K

7.2.1.1 Application of TOE to all sensitive data transmitted

Inter-document Reference - OBE_DC2K_transmit_data_through_TOE

All sensitive data held within the secure domain must be passed through the TOE prior to it reaching the insecure domain.

7.2.1.2 Check For Signs of Unit Tampering

Inter-document Reference - OBE_DC2K_check_for_unit_tamper

Units should be checked periodically for signs of tampering. If tampering is deemed to have taken place, this should be reported immediately to the appropriate authority.

7.2.1.3 Select Appropriate Unit Settings

Inter-document Reference – OBE_DC2K_select_appropriate_settings

A policy should exist concerning the choice of key lifetimes.

7.2.1.4 Unit Management

Inter-document Reference – OBE_DC2K_management_centre

Management of the unit will be restricted to the use of the specified installation, commissioning and configuration procedure [44], using the Element Manager management software (supplied with the DC2K) running on a management centre PC. The management centre will be operated by trusted personnel, and both the environment in which it is operated and its connection to the unit's Ethernet management port will be physically secure. No other use will be made of the unit's Ethernet or serial management ports.

7.2.2 Security Objectives for the Environment of the SGSS

7.2.2.1 Application of Physical Protection to the Secure Domain

Inter-document Reference - OBE_SGSS_protect_secure_domain

Physical protection measures i.e. securely locked premises, guards etc. must be applied as necessary to the secure domain in which sensitive and otherwise unprotected data resides. The value of the assets protected by these measures in the secure domain must not exceed £500,000.

7.2.2.2 Application of Physical Security to External Key Material

Inter-document Reference - OBE_SGSS_protect_key_material

Physical protection measures i.e. securely locked premises, guards etc. must be applied as necessary to sensitive key material where this is stored externally to the unit. Such key material should only be handled by trusted personnel.

7.2.2.3 Application of Physical Security to External Sensitive Algorithms

Inter-document Reference - OBE_SGSS_protect_algorithms

Physical protection measures i.e. securely locked premises, guards etc. must be applied as necessary to sensitive algorithms where these are stored externally to the unit. Such algorithms should only be handled by trusted personnel.

7.2.2.4 Application of Physical Security to Unit When Keyed

Inter-document Reference - OBE_SGSS_protect_keyed_unit

Physical protection measures i.e. securely locked premises, guards etc. must be applied as necessary to units that have been commissioned. Access to keyed units should only be provided to trusted personnel.

8 IT Security Requirements

8.1 Target of Evaluation Security Requirements

8.1.1 Target of Evaluation Security Functional Requirements

The following security functional requirements, defined in the form of components extracted from [1], are required to fully support the security objectives.

For each security functional requirement it is clearly indicated whether the requirement applies to the DC2K or the SGSS.

The assignment operation on the security requirements is indicated by normal underlined text in square brackets.

Section 11.3.1 lists the TOE security functions that meet each of the security functional requirements. See section 9.1 for a discussion of the TOE security functions provided by the SGSS and DC2K.

Note that section 9.1 specifies aspects of the TOE security functions for which FIPS or CAPS results are quoted and which are thus excluded from the scope of CC evaluation. The scoping specifications of section 9.1 thus also qualify the SFRs below.

Note also that, although assurance component AVA_SOF.1 is included in the TOE Security Assurance requirements, all TOE Security Functions realised by a probabilistic or permutational mechanism are cryptographic. Hence a statement regarding their strength level is outside the scope of this Security Target.

8.1.1.1 FCS_CKM.1 [DC2K] – Cryptographic Key Generation

FCS_CKM.1.1
The TSF shall generate cryptographic keys in accordance with specified cryptographic key generation algorithms [described in [5]] and specified cryptographic key sizes [as specified by the algorithm specifications referenced in [4]] that meet the following: [algorithm specification defined or standards referenced in [4]].

8.1.1.2 FCS_CKM.2 [DC2K] – Cryptographic Key Distribution

FCS_CKM.2.1
The TSF shall distribute cryptographic keys in accordance with a specified cryptographic key distribution method [as defined in [5]] that meets the following: [standards referenced in [4]].

8.1.1.3 FCS_COP.1 [DC2K] – Cryptographic Operation

FCS_COP.1.1

The TSF shall perform [data authentication[1], key exchange protocol, data encryption] in accordance with a specified cryptographic algorithm [listed in [4]] and cryptographic key sizes [as in the algorithm specification] that meet the following: [standards referenced within [4]].

8.1.1.4 FCS_COP.1 [SGSS] – Cryptographic Operation

FCS_COP.1.1
The TSF shall perform [data authentication] in accordance with a specified cryptographic algorithm [listed in [4]] and cryptographic key sizes [as in the algorithm specification] that meet the following: [standards referenced within [4]].

8.1.1.5 FPT_PHP.3 [SGSS] – Resistance to Physical Attack

FPT_PHP.3.1
The TSF shall resist [physical intrusion, high and low voltage attacks, attacks requiring temperature extremes, and attacks requiring physical movement] to the [SGSS component of the TOE] by responding automatically such that the TSP is not violated.

8.1.2 Target of Evaluation Security Assurance Requirements

Assurance requirements are as specified by [39].

8.2 Security Requirements for the IT Environment

There are no security requirements for the TOE's assumed IT environment.

[1] In the context of a cryptographic product, data authentication has a precise meaning: it is a means by which the receiver of data can cryptographically ascertain its origin, such that the sender of the data cannot masquerade as someone else.

9 Target of Evaluation Summary Specification

9.1 Target of Evaluation Security Functions

Reference [36] provides a correlation between Security Functions described in this section and those defined in previous ITSec and CAPS DataCryptor 2000 evaluations (with Security Targets at [37] and [38] respectively).

(Note that although assurance component AVA_SOF.1 is included in the TOE Security Assurance requirements, all TOE Security Functions realised by a probabilistic or permutational mechanism are cryptographic. Hence a statement regarding their strength level
is outside the scope of this Security Target.)

9.1.1 TOE System Architecture

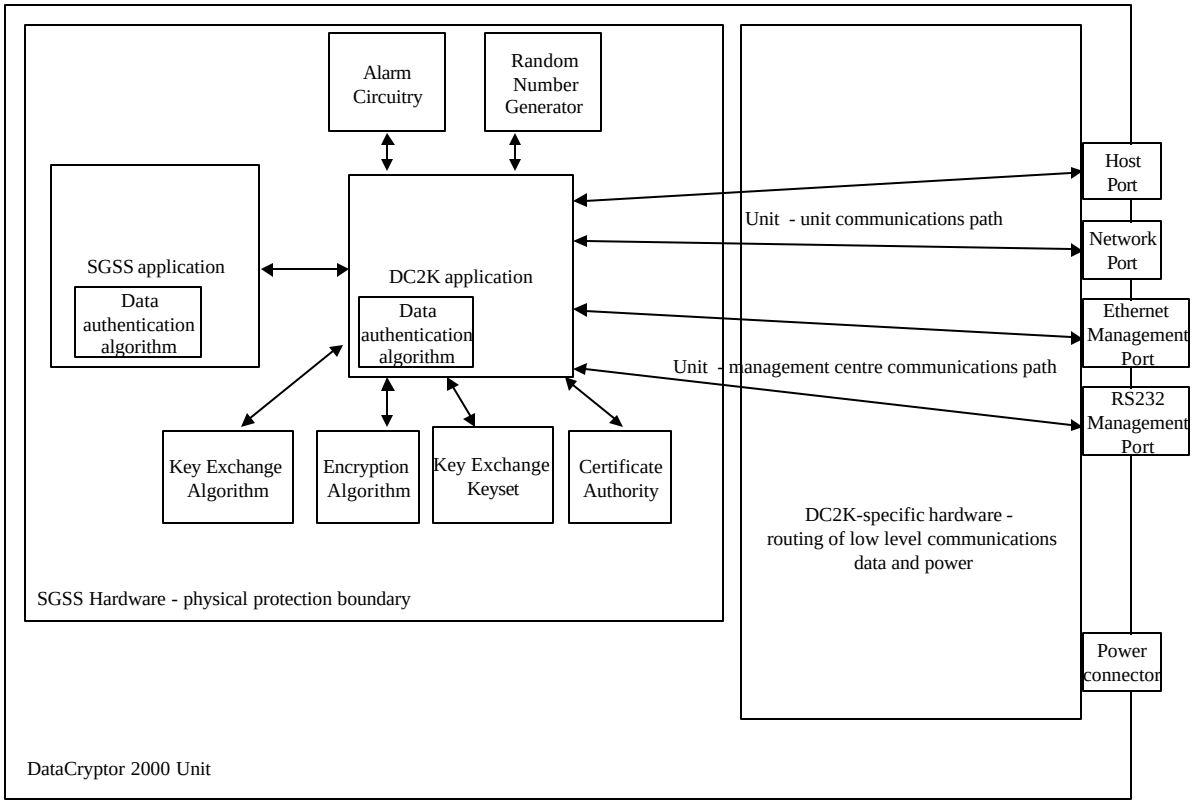


Figure 2 – DataCryptor 2000 and SGSS Architecture

Figure 2 depicts the high level architecture of the DataCryptor 2000 and the SGSS. It demonstrates the organisation of the products at a logical and physical level and provides a context in which to discuss the instantiation of TOE security functions.

9.1.2 SGSS Application

The SGSS application performs cryptographic authentication of any DC2K application to be loaded.

The SGSS contains an implementation of a data authentication algorithm. In addition, at manufacture time, a public key value is embedded in the SGSS application. When the DC2K application is generated, a digital signature is generated over the application using the corresponding secret key, which is held securely at the development site. The application is concatenated with the signature.

On loading the DC2K application, the signature concatenated with the application is verified by the SGSS application's data authentication implementation using the public key embedded within it. If the verification is successful, the application is accepted by the SGSS; otherwise it is rejected.

In this way, only DC2K applications that have been signed by the manufacturer may be run in the unit.

Inter-document Reference SF_SGSS_data_authentication_implementation

SF_SGSS_data_authentication_implementation specification:

> The TOE shall provide a means of verifying a digital signature on an application that is being loaded into the TOE (in accordance with [4], 'Data Authentication Algorithms').
>
> Note that the authentication algorithm is excluded from the scope of the CC TOE. A FIPS validation certificate confirms the correct implementation of this algorithm and thereby complements the CC evaluation.

9.1.3 SGSS Hardware

9.1.3.1 Random Number Generator

The SGSS hardware contains a hardware random number generator that generates high quality random numbers for use in the Key Exchange Protocol (see sections 9.1.5 and 9.1.6.1 below). The random number generator output is subject to frequent background diagnostic statistical testing invoked by the DC2K application, failure of which causes the unit to cease transmission of data. This ensures that all random numbers used for security relevant purposes are of high quality.

Inter-document Reference SF_SGSS_Random_Number_Generator

SF_SGSS_Random_Number_Generator specification:

> The TOE shall provide a means of generating random numbers. The TOE shall check every ten minutes[2] that the random numbers being generated pass the random number generator

[2] Background diagnostic statistical testing is performed by the application. The DC2K does so every 10 minutes.

tests specified in Section 4.11 of [43]. If the tests are failed then the TOE shall cease transmission of network data until such time as the tests are passed.

9.1.3.2 Alarm Circuitry

The physical security provided by the SGSS operates as a protection mechanism for all the DataCryptor 2000's sensitive contents (keys, sensitive algorithms etc.), by providing resistance to physical intrusion and voltage attacks, and temperature and motion sensors.

Intrusion protection is provided by a copper mesh that consists of two circuits - a continuity circuit and a guard circuit. The SGSS is surrounded by the mesh and potted in an opaque resin. An alarm is triggered if the continuity circuit is broken or if the two circuits are bridged. The wires of the circuit are lacquered so that they cannot bridge simply by touching. Any attempt to drill through the resin and mesh should result in an alarm being triggered either by breaking the continuity circuit or by shorting the two circuits. Similarly, attempting to dissolve the resin to gain access to the secure area would dissolve the lacquer on the wires of the mesh, causing the two circuits to short.

The alarm circuit is powered from the main power supply when this is available, or by a battery otherwise. Should the battery fail or become disconnected (i.e. voltage drops), an alarm will be triggered. Similarly, if the voltage levels surge or are actively driven above the normal levels, an alarm is triggered. This prevents both high and low voltage attacks.

A temperature sensor causes the alarm circuit to be triggered at temperatures above 70°C (+/-5°C) or below -15°C (+/-5°C). (Note that the temperature sensor will not respond immediately to changes in the ambient temperature of the DC2K since the sensor is inside the SGSS. Note also, as stated in [14], that the recommended operating temperature range for the DC2K is -5ºC to +40ºC.)

There is a movement sensor that triggers an alarm on detection of movement. (Note that the movement sensor is unlikely to respond to very gentle movement of the unit.)

The effect of triggering an alarm is to force the voltage supply rails to all devices containing sensitive information to ground, causing them to lose their contents. Additionally the interface lines into the DC2K specific hardware are disconnected to prevent an attacker from attempting to prevent device erasure by externally driving in supply voltage.

Inter-document Reference – SF_SGSS_alarm_circuitry

SF_SGSS_alarm_circuitry specification:

The TOE shall provide a means of detecting the following events:

- physical intrusion into the TOE
- TOE's power voltage outside permitted range
- TOE's temperature outside permitted range
- movement of the TOE.

The TOE shall provide a means of destroying sensitive data (i.e. algorithms, keys and Certificate Authorities other than the manufacturer's Certificate Authority[3]) stored within itself should the TOE detect any of the above events.

9.1.4 DC2K Application

The DC2K application is responsible for several security critical functions, discussed below:

- cryptographic authentication of key exchange algorithm
- cryptographic authentication of encryption algorithm
- cryptographic authentication of Certificate Authorities
- cryptographic authentication of Key Exchange Algorithm Keysets.

9.1.4.1 Authentication of Key Exchange Algorithm

The DC2K application contains an implementation of a data authentication algorithm. In addition, at manufacture time, the "DC2K application" public key value is embedded in the DC2K application. When a Key Exchange Algorithm is generated, a digital signature is generated over the algorithm using the corresponding secret key, which is held securely at the development site. The algorithm is concatenated with the signature.

On loading the Key Exchange Algorithm, the signature concatenated with the algorithm is verified by the DC2K application's data authentication implementation using the public key embedded within it. If the verification is successful the algorithm is accepted by the DC2K; otherwise it is rejected.

In this way, only Key Exchange Algorithms that have been signed by an authorised body may be run in the unit.

9.1.4.2 Authentication of Encryption Algorithm

The DC2K application contains an implementation of a data authentication algorithm. In addition, at manufacture time, the "DC2K application" public key value is embedded in the DC2K application. When an Encryption Algorithm is generated, a digital signature is generated over the algorithm using the corresponding secret key, which is held securely at the development site. The algorithm is concatenated with the signature.

On loading the Encryption Algorithm, the signature concatenated with the algorithm is verified by the DC2K application's data authentication implementation using the public key embedded within it. If the verification is successful, the algorithm is accepted by the DC2K; otherwise it is rejected.

In this way, only Encryption Algorithms that have been signed by an authorised body may be run in the unit.

[3] The SGSS erases the contents of its SRAM, DRAM and cipher FPGA.

9.1.4.3 Authentication of Certificate Authorities

Authentication of Certificate Authorities occurs when signed Certificate Authorities are loaded during the unit's commissioning process. The signature on the Certificate Authority is verified by the DC2K's data authentication implementation. If the verification is successful the Certificate Authority is accepted by the DC2K; otherwise it is rejected.

In this way, only Certificate Authorities that have been signed by an authorised body may be used to verify the signature on key exchange keysets.

9.1.4.4 Authentication of Key Exchange Algorithm Keysets

Authentication of Key Exchange Algorithm Keysets occurs at two points in DataCryptor 2000 operation:

9.1.4.4.1 Unit Commissioning

During the unit's commissioning process, Key Exchange Algorithm Keysets signed by a CA are loaded. The signature on the keyset is verified by the DC2K's data authentication implementation. If the verification is successful the keyset is accepted by the DC2K; otherwise it is rejected.

9.1.4.4.2 Key Exchange Protocol

During the key exchange protocol (see [5], section 4.2.1), units exchange signed key exchange certificates. Both units must positively verify that the keyset has been authorised by a CA that they are operating under before proceeding to generate a shared key encryption key. The DC2K application's data authentication implementation is used for this purpose.

Inter-document Reference SF_DC2K_data_authentication_implementation

SF_DC2K_data_authentication_implementation specification:

> *The TOE shall provide a means of verifying a digital signature on a certificate, algorithm or keyset (in accordance with [4], 'Data Authentication Algorithms').*
>
> Note that the authentication algorithm is excluded from the scope of the CC TOE. A FIPS validation certificate confirms the correct implementation of this algorithm and thereby complements the CC evaluation.

9.1.5 DC2K Key Exchange Algorithm

A secure key exchange algorithm allows two units to establish a common Key Encryption Key (KEK) without either party having to transmit any secret data.

An implementation of a secure key exchange algorithm is used for this purpose, which requires the input of both parties' signed public and secret keys. In addition to these values, each unit inputs a random one-time public-secret key pair (using the SGSS random number generator), ensuring that every KEK generated between the two parties is unique.

Inter-document Reference SF_DC2K_key_exchange_algorithm

SF_DC2K_key_exchange_algorithm specification:

> The TOE shall provide a means of secure key exchange in order to establish a KEK with another instance of the TOE (in accordance with [4], 'Key Exchange Algorithms' and 'Data Hashing Algorithms'; and with [5], section 4.2.2, steps 6,7,8 and section 8.2).
>
> Note that the key exchange algorithm is excluded from the scope of the CC TOE. The results of a CESG cryptographic evaluation confirm the correct implementation of this algorithm and thereby complement the CC evaluation.

9.1.6 DC2K Encryption Algorithm

The DataCryptor 2000 uses an encryption algorithm for two purposes – key encryption and data encryption for user and management traffic.

9.1.6.1 Key Encryption

Having agreed a KEK as described in section 9.1.5, the two units must securely derive a data encryption key (DEK). This is achieved by both entities generating random data, encrypting it with the KEK, and sending it to the other party.

The encrypted random data is decrypted by both entities, and combined to generate a shared DEK.

9.1.6.2 Data Encryption

Having agreed a DEK, the encryption algorithm may now be used to encrypt transmitted user data and decrypt received user data.

Inter-document Reference SF_DC2K_encryption_algorithm

SF_DC2K_encryption_algorithm specification:

1. The TOE shall provide a means of establishing a DEK with another instance of the TOE (in accordance with [5], section 4.1.1, steps 4,5).

2. The TOE shall provide a means of transmitting encrypted network traffic to another instance of the TOE (in accordance with [4], 'Data Encryption Algorithms').

   Note that the encryption algorithm is excluded from the scope of the CC TOE. A FIPS validation certificate confirms the correct implementation of this algorithm and thereby complements the CC evaluation.

9.2 Assurance Measures

In the sections that follow, non-italic font is used to state the developer actions of the assurance requirements (extracted directly from [1]), and *italic font* describes the evaluation deliverables that will provide the necessary assurance.

It is anticipated that [39] will not quote an assurance requirement greater than EAL5. The assurance requirements and evaluation deliverables listed below are thus those for EAL5. Where a lower assurance requirement is quoted, this will effectively be a subset of EAL5 and the appropriate subset of deliverables will apply.

9.2.1 ACM_AUT.1 Partial CM automation

Developer action elements
ACM_AUT.1.1D The developer shall use a CM system.
ACM_AUT.1.2D The developer shall provide a CM plan.

TOE development conforms to the Engineering Configuration Management Procedure provided at [6]. The plan describes the use of automated tools for configuration management.

9.2.2 ACM_CAP.4 Generation support and acceptance procedures

Developer action elements
ACM_CAP.4.1D The developer shall provide a reference for the TOE.

A unique TOE reference is provided in section 4.1.

ACM_CAP.4.2D The developer shall use a CM system.
ACM_CAP.4.3D The developer shall provide CM documentation.

TOE development conforms to the Engineering Configuration Management Procedure at [6]. The plan describes the use of automated tools for configuration management. References [7] and [8] provide a configuration list and acceptance plan respectively. Documentation describing the use of an automated version control system is provided at [9].

9.2.3 ACM_SCP.3 Development tools CM coverage

Developer action elements
ACM_SCP.3.1D The developer shall provide CM documentation.

Documentation management conforms to the Configuration Management Procedures described at [6]. In addition, references [10] and [11] discuss the production of documentation and tracking of software tools respectively.

9.2.4 ADO_DEL.2 Detection of modification

Developer action elements
ADO_DEL.2.1D The developer shall document procedures for delivery of the TOE or parts of it to the user.
ADO_DEL.2.2D The developer shall use the delivery procedures.

TOE delivery conforms to the company Packing and Despatch procedures provided at reference [12].

9.2.5 ADO_IGS.1 Installation, generation, and start-up procedures

Developer action elements
ADO_IGS.1.1D The developer shall document procedures necessary for the secure installation, generation, and start-up of the TOE.

The TOE is shipped with an installation guide and a user guide. These are provided at references [13] and [14] respectively. A guide describing how to install the TOE in accordance with the evaluated configuration is provided at reference [44].

9.2.6 ADV_FSP.3 Semiformal functional specification

Developer action elements
ADV_FSP.3.1D The developer shall provide a functional specification.

A functional specification of the TOE Security Functions is provided at [41]. Its style is semi-formal.

9.2.7 ADV_HLD.3 Semiformal high-level design

Developer action elements
ADV_HLD.3.1D The developer shall provide the high-level design of the TSF.

A high-level design of the TOE Security Functions is provided at [35]. Its style is semi-formal.

9.2.8 ADV_IMP.2 Implementation of the TSF

Developer action elements
ADV_IMP.2.1D The developer shall provide the implementation representation for the entire TSF.

The entire TSF implementation representation shall be provided in the form of C source, VHDL (Very High Speed Integrated Circuit Hardware Description Language), and hardware schematics for the version under evaluation.

9.2.9 ADV_INT.1 Modularity

Developer action elements

ADV_INT.1.1D The developer shall design and structure the TSF in a modular fashion that avoids unnecessary interactions between the modules of the design.
ADV_INT.1.2D The developer shall provide an architectural description.

The TSF is designed in a modular fashion in accordance with the architectural design at reference [16].

9.2.10 ADV_LLD.1 Descriptive low-level design

Developer action elements
ADV_LLD.1.1D The developer shall provide the low-level design of the TSF.

A descriptive low-level design of the TOE Security Functions is provided at [34].

9.2.11 ADV_RCR.2 Semiformal correspondence demonstration

Developer action elements
ADV_RCR.2.1D The developer shall provide an analysis of correspondence between all adjacent pairs of TSF representations that are provided.

Reference [17] provides an analysis of the correspondence between all adjacent pairs of the TSF.

9.2.12 ADV_SPM.3 Formal TOE security policy model

Developer action elements
ADV_SPM.3.1D The developer shall provide a TSP model.
ADV_SPM.3.2D The developer shall demonstrate or prove, as appropriate, correspondence between the functional specification and the TSP model.

A formal TOE Security Policy Model, along with a correspondence with the TOE functional specification, is provided in [18].

9.2.13 AGD_ADM.1 Administrator guidance

Developer action elements
AGD_ADM.1.1D The developer shall provide administrator guidance addressed to system administrative personnel.

The TOE is shipped with an installation guide and a user guide. These are provided at references [13] and [14] respectively.

9.2.14 AGD_USR.1 User guidance

Developer action elements
AGD_USR.1.1D The developer shall provide user guidance.

The TOE is shipped with an installation guide and a user guide. These are provided at references [13] and [14] respectively.

9.2.15 ALC_DVS.1 Identification of security measures

Developer action elements
ALC_DVS.1.1D The developer shall produce development security documentation.

Development of the TOE conforms to the standards defined within the UK Government's Manual of Protective Security.

9.2.16 ALC_LCD.2 Standardised life-cycle model

Developer action elements
ALC_LCD.2.1D The developer shall establish a life-cycle model to be used in the development and maintenance of the TOE.
ALC_LCD.2.2D The developer shall provide life-cycle definition documentation.
ALC_LCD.2.3D The developer shall use a standardised life-cycle model to develop and maintain the TOE.

A life-cycle model for the DataCryptor 2000 is defined at [19]. The company's quality assurance standards, including life-cycle procedures, have been examined in support of am ISO-9001 registration. As such, a recognised standards body has approved the life-cycle model.

9.2.17 ALC_TAT.2 Compliance with implementation standards

Developer action elements
ALC_TAT.2.1D The developer shall identify the development tools being used for the TOE.
ALC_TAT.2.2D The developer shall document the selected implementation-dependent options of the development tools.

TOE development tools used are described in the software tools registry, as defined in [11]

ALC_TAT.2.3D The developer shall describe the implementation standards to be applied.

The implementation standards applied to the TOE development are described at references [20] and [21] ('C' and VHDL coding standards respectively).

9.2.18 ATE_COV.2 Analysis of coverage

Developer action elements
ATE_COV.2.1D The developer shall provide an analysis of the test coverage.

An analysis of the DataCryptor 2000 Test Coverage is provided at reference [22].

9.2.19 ATE_DPT.2 Testing: low-level design

Developer action elements
ATE_DPT.2.1D The developer shall provide the analysis of the depth of testing.

An analysis of the depth of testing carried out on the TOE is provided at reference [23]. The analysis includes a description of the correspondence between the tests performed and the high and low level design specifications.

9.2.20 ATE_FUN.1 Functional testing

Developer action elements
ATE_FUN.1.1D The developer shall test the TSF and document the results.
ATE_FUN.1.2D The developer shall provide test documentation.

The DataCryptor 2000 has been tested as defined in references [24] to [28] inclusive and [42], which include test results. In addition independently defined tests have been performed, specified as references [29] – [31] inclusive, the results of which are reported at reference [32].

9.2.21 ATE_IND.2 Independent testing - sample

Developer action elements
ATE_IND.2.1D The developer shall provide the TOE for testing.

The TOE is available for independent testing as required.

9.2.22 AVA_CCA.1 Covert channel analysis

Developer action elements
AVA_CCA.1.1D The developer shall conduct a search for covert channels for each information flow control policy.
AVA_CCA.1.2D The developer shall provide covert channel analysis documentation.

Covert Channel Analysis is not applicable to the DataCryptor 2000 evaluation, since it has no information flow control policies.

9.2.23 AVA_MSU.2 Validation of analysis

Developer action elements
AVA_MSU.2.1D The developer shall provide guidance documentation.
AVA_MSU.2.2D The developer shall document an analysis of the guidance documentation.

Guidance documentation in the form of the DataCryptor 2000 installation and user manuals are provided at references [13] and [14] respectively. An analysis of the guidance documentation is given at reference [33].

9.2.24 AVA_SOF.1 Strength of TOE security function evaluation

Developer action elements
AVA_SOF.1.1D The developer shall perform a strength of TOE security function analysis for each mechanism identified in the ST as having a strength of TOE security function claim.

All TOE Security Functions realised by a probabilistic or permutational mechanism are cryptographic. Hence a statement regarding their strength level is outside the scope of this Security Target.

9.2.25 AVA_VLA.3 Moderately resistant

Developer action elements
AVA_VLA.3.1D The developer shall perform and document an analysis of the TOE deliverables searching for ways in which a user can violate the TSP.
AVA_VLA.3.2D The developer shall document the disposition of identified vulnerabilities.

A Vulnerability Analysis has been carried out on the DataCryptor 2000. This is provided at reference [40].

10 Protection Profile Claims

No claims of conformance to a Protection Profile are made.

11 Rationale

11.1 Security Objectives Rationale

11.1.1 Security Objectives counter Threats

The security objectives are designed to counter the threats in line with the threat descriptions given in sections 6.3 and 6.4.

The table below lists each of the threats identified in sections 6.3 and 6.4, and for each threat applies sufficient security objectives (taken from section 7) to fully counter the threat. Since every threat is countered, the security objectives are sufficient to meet all of the assumed threats.

Note that threats to the DC2K may be countered by SGSS security objectives, but threats to the SGSS cannot be countered by DC2K security objectives.

Threat	Asset	TOE Security Objective	Environment Security Objective
T_DC2K_extract_data_from_secure_domain	*A_user_data*		*OBE_SGSS_protect_secure_domain*
T_DC2K_record_plaintext_data_from_insecure_domain	*A_user_data*		*OBE_DC2K_transmit_data_through_TOE*
T_DC2K_cryptanalyse_data_within_insecure_domain	*A_user_data*	*OBT_DC2K_provide_data_confidentiality*	
T_DC2K_access_to_secret_authentication_key	*A_user_key* *A_user_data*		*OBE_SGSS_protect_key_material*
T_DC2K_access_to_secret_key_exchange_alg_keys	*A_user_key* *A_user_data* *A_user_data*		*OBE_SGSS_protect_key_material*
T_DC2K_access_to_algorithm_outside_unit	*A_user_algorithm*		*OBE_SGSS_protect_algorithms*
T_DC2K_cryptanalyse_keys_within_insecure_domain *T_DC2K_loss_of_commissioned_unit*	*A_user_key* *A_user_data* *A_user_data*	*OBT_DC2K_provide_secure_key_management*	*OBE_SGSS_protect_keyed_unit*
T_DC2K_tamper_with_unit	*A_user_data*		*OBE_DC2K_check_for_unit_tamper*

T_DC2K_algorithm_ replacement	*A_user_key* *A_user_data*	*OBT_DC2K_provide_ secure_algorithm_ load*	
T_DC2K_certificate_ authority_replacement	*A_user_key* *A_user_data*	*OBT_DC2K_provide_ secure_CA_load*	
T_DC2K_key_exchange _certificate_replacement	*A_user_key* *A_user_data*	*OBT_DC2K_provide_ secure_key_exchange _keyset_load*	
T_SGSS_access_to_keys _within_unit	*A_user_key* *A_user_data*	*OBT_SGSS_provide_ resistance_to_ physical_attack*	
T_SGSS_access_to_ algorithm_within_unit	*A_user_algorithm*	*OBT_SGSS_provide_ resistance_to_ physical_attack*	
T_SGSS_application_ replacement	*A_user_algorithm* *A_user_key* *A_user_data*	*OBT_SGSS_provide_ secure_application_ load*	

Table 1 – correlation between threats and security objectives required to fully counteract threat

The following shows that, for each threat, the security objectives applied to it in Table 1 successfully counter that threat.

The threat that an attacker with physical access to the secure domain gains access to data assets residing there (*T_DC2K_extract_data_from_secure_domain*) is countered by *OBE_SGSS_protect_secure_domain*, which applies physical protection measures to the secure domain.

The threat that an attacker is able to access sensitive data because it has been sent unencrypted into the insecure domain (*T_DC2K_record_plaintext_data_from_insecure_domain*) is countered by the security objective *OBE_DC2K_transmit_data_through_TOE*. This ensures that all data transmitted from the secure domain to the insecure domain passes through the TOE. Since the TOE is placed in encrypt mode as part of the installation procedures [44] the threat is removed because sensitive data will not exist in an unencrypted form within the insecure domain.

The threat that an attacker gains access to data by recording it in its encrypted form while in the insecure domain and then employing cryptanalysis (*T_DC2K_cryptanalyse_data_within_insecure_domain*) is countered by the security objective *OBT_DC2K_provide_data_confidentiality*, which implements a confidentiality service by the use of encryption. If correctly implemented, this objective diminishes the threat by requiring greater expertise and resources on the part of the attacker.

The threat of disclosure of the externally held secret authentication key (*T_DC2K_access_to_secret_authentication_key*) makes a man-in-the-middle attack possible. This threat is countered by the security objective *OBE_SGSS_protect_key_material*, which ensures the appropriate physical protection measures are used for key material stored externally to the TOE.

The threat of disclosure of secret key exchange algorithm keys that are loaded into the TOE from an external source (*T_DC2K_access_to_secret_key_exchange_alg_keys*) is also countered by *OBE_SGSS_protect_key_material*, which applies physical security measures to key material stored externally.

The threat of an attacker gaining knowledge of sensitive algorithms while they are external to the TOE (*T_DC2K_access_to_algorithm_outside_unit*) is countered by the objective *OBE_SGSS_protect_algorithms*, which ensures that appropriate physical security measures are applied to algorithms stored externally to the TOE.

The threat of an attacker determining key encryption keys or data encryption keys by cryptanalysis of the key exchange protocol between two instances of the TOE (*T_DC2K_cryptanalyse_keys_within_insecure_domain*) is countered by security objective *OBT_DC2K_provide_secure_key_management*, which provides a means of exchanging key material securely. If this objective is met correctly then the threat is greatly diminished because the method of attack becomes impractical.

The threat of an attacker gaining possession of a commissioned unit (*T_DC2K_loss_of_commissioned_unit*) is countered by security objective *OBE_SGSS_protect_keyed_unit*, which applies physical protection measures to commissioned units. This reduces the opportunity and so diminishes the threat.

The threat of an attacker physically tampering with the TOE so that it did not encrypt transmitted data (*T_DC2K_tamper_with_unit*) is countered by security objective *OBE_DC2K_check_for_unit_tamper*, which ensures that units are checked periodically for signs of tampering. The objective will mitigate the effects of the threat by ensuring that such an attack is detected.

The threat of an attacker installing a rogue encryption or key exchange algorithm (*T_DC2K_algorithm_replacement*) is countered by the TOE security objective *OBT_DC2K_provide_secure_algorithm_load*, which provides a means of cryptographically verifying the authenticity of an algorithm prior to its loading and use. This reduces the likelihood of a successful attack.

If an attacker manages to load their own Certificate Authorities into two communicating instances of the TOE, it may lead to the exposure of key material and ultimately to the exposure of user data. This threat (*T_DC2K_certificate_authority_replacement*) is countered by the TOE security objective *OBT_DC2K_provide_secure_CA_load*, which provides a means of cryptographically verifying the authenticity of a CA before its loading and use. This reduces the likelihood of a successful attack.

Likewise, the threat of an attacker loading known Key Exchange certificates so that keys and user data is exposed (*T_DC2K_key_exchange_certificate_replacement*) is countered by the TOE security objective *OBT_DC2K_provide_secure_key_exchange_keyset_load*, which provides a means of cryptographically verifying the authenticity of a Key Exchange Keyset prior to its loading and use. This reduces the likelihood of a successful attack.

The threat of an attacker gaining knowledge of secret key material stored within the TOE (*T_SGSS_access_to_keys_within_unit*) is countered by

OBT_SGSS_provide_resistance_to_physical_attack, which provides resistance to such forms of direct physical attack.

Similarly, the threat of an attacker gaining knowledge of sensitive algorithms while they are stored within the TOE (*T_SGSS_access_to_algorithm_within_unit*) is countered by *OBT_SGSS_provide_resistance_to_physical_attack* as it provides resistance to such forms of direct physical attack.

The threat of an attacker subverting the security of the TOE by installing their own application (*T_SGSS_application_replacement*) is countered by the TOE security objective *OBT_SGSS_provide_secure_application_load*, which provides a means of cryptographically verifying the authenticity of an application. Implemented correctly, this objective greatly reduces the likelihood of this attack being successful.

It follows that the security objectives are suitable to counter all identified threats.

11.1.2 Security Objectives cover the Environment Assumptions

The following table shows that the security objectives for the environment identified in section 7 are suitable to cover all of the assumptions defined in sections 6.1 and 6.2.

Environment Assumptions defined in sections 6.1 and 6.2	Security Objectives for the Environment
6.1.1 – Assumed Usage	*OBE_DC2K_select_appropriate_settings* *OBE_DC2K_check_for_unit_tamper*
6.1.1.1 – Limitations of Use	*OBE_SGSS_protect_secure_domain* *OBE_DC2K_transmit_data_through_TOE*
6.1.2 – Protection of Assets	*OBE_SGSS_protect_secure_domain*
6.1.3.1 – Management *6.1.3.2* – Physical Protection Measures (DC2K)	*OBE_DC2K_management_centre* *OBE_SGSS_protect_key_material* *OBE_SGSS_protect_algorithms* *OBE_SGSS_protect_keyed_unit*
6.1.3.3 – Connectivity	*OBE_DC2K_transmit_data_through_TOE* *OBE_DC2K_management_centre*
6.1.3.4 – Personnel	*OBE_SGSS_protect_key_material* *OBE_SGSS_protect_algorithms* *OBE_SGSS_protect_keyed_unit*
6.2.1 – Physical Protection Measures (SGSS)	*OBE_SGSS_protect_secure_domain* *OBE_SGSS_protect_key_material* *OBE_SGSS_protect_algorithms* *OBE_SGSS_protect_keyed_unit*

Table 2 – Objectives for the Environment cover all the Environment Assumptions

Table 2 lists each of the environment assumptions identified in sections 6.1 and 6.2 and shows that each assumption is covered by environment objectives (taken from section 7). The table shows that the environment objectives cover all of the environment assumptions.

Assumption 6.1.1 is covered by the two security objectives *OBE_DC2K_select_appropriate_settings* and *OBE_DC2K_check_for_unit_tamper*. This is clear from the statement of the objectives given in sections 7.2.1.3 and 7.2.1.2.

Assumption 6.1.1.1 is covered by the two security objectives *OBE_SGSS_protect_secure_domain* and *OBE_DC2K_transmit_data_through_TOE*. Again it is clear from sections 7.2.2.1 and 7.2.1.1 that the objectives cover the assumption.

Assumption 6.1.2 is covered by the single security objective *OBE_SGSS_protect_secure_domain* (see section 7.2.2.1).

Assumption 6.1.3.1 is covered by the single security objective *OBE_DC2K_management_centre* (see section 7.2.1.4).

Assumption 6.1.3.2 is covered by the three security objectives *OBE_SGSS_protect_key_material, OBE_SGSS_protect_algorithms* and *OBE_SGSS_protect_keyed_unit*. This is clear from the statement of the objectives given in sections 7.2.2.2, 7.2.2.3 and 7.2.2.4.

Assumption 6.1.3.3 is covered by the two security objectives *OBE_DC2K_transmit_data_through_TOE* and *OBE_DC2K_management_centre*. This is clear from the statement of the objectives given in sections 7.2.1.1 and 7.2.1.4.

Assumption 6.1.3.4 is covered by the three security objectives *OBE_SGSS_protect_key_material, OBE_SGSS_protect_algorithms* and *OBE_SGSS_protect_keyed_unit*. This is clear from the statement of the objectives given in sections 7.2.2.2, 7.2.2.3 and 7.2.2.4.

Finally assumption 6.2.1 is covered by the four security objectives *OBE_SGSS_protect_secure_domain, OBE_SGSS_protect_key_material, OBE_SGSS_protect_algorithms* and *OBE_SGSS_protect_keyed_unit*. This is clear from the statement of the objectives given in sections 7.2.2.1, 7.2.2.2, 7.2.2.3 and 7.2.2.4.

11.2 Security Requirements Rationale

11.2.1 Functional Requirements

Security Objective	IT Functional Requirement
OBT_DC2K_provide_data_confidentiality	FCS_COP.1 [DC2K]
*OBT_DC2K_provide_secure_*	FCS_CKM.1 [DC2K], FCS_CKM.2 [DC2K],

key_management *OBT_SGSS_provide_resistance_to_ physical_attack*	FCS_COP.1 [DC2K] FPT_PHP.3 [SGSS]
OBT_SGSS_provide_secure_application_ load	FCS_COP.1 [SGSS]
OBT_DC2K_provide_secure_algorithm_ load	FCS_COP.1 [DC2K]
OBT_DC2K_provide_secure_CA_load *OBT_DC2K_provide_secure_key_ exchange_keyset_load*	FCS_COP.1 [DC2K] FCS_COP.1 [DC2K]

Table 3 - IT functional requirements required to meet each of the TOE's security objectives .

Table 3 lists each of the security objectives identified in section 7, and for each, applies sufficient IT functional requirements (taken from section 8.1.1) to meet the objective. Since every security objective is met, the IT functional requirements are sufficient to meet all of the security objectives.

In addition, where each objective is met by exactly one IT functional requirement, it follows that the functional requirement must be necessary as well as sufficient to meet the objective.

Each security objective from Table 3 is considered below where the indicated IT functional requirements are shown to meet it.

The objective *OBT_DC2K_provide_data_confidentiality* is for the TOE to provide the option of a confidentiality service to all data transmitted through it. Data confidentiality is achieved using encryption and is fully covered by the data encryption component of the functional requirement FCS_COP.1 [DC2K].

The objective *OBT_DC2K_provide_secure_key_management* requires three IT functional requirements to fully meet it, FCS_CKM.1 [DC2K], FCS_CKM.2 [DC2K] and FCS_COP.1 [DC2K]. Public data exchange is achieved by FCS_COP.1 [DC2K] (see [5] section 4.2.1). FCS_CKM.1 [DC2K] and FCS_CKM.2 [DC2K] respectively achieve generation and distribution of keys. Logically, all three requirements are needed to satisfy this objective fully.

The objective *OBT_SGSS_provide_resistance_to_physical_attack* is for the TOE to provide resistance to direct physical attacks aimed at extracting sensitive data. The functional requirement FPT_PHP.3 [SGSS] satisfies this objective by resisting the physical tampering scenarios listed in its definition.

The remaining four objectives (*OBT_SGSS_provide_secure_application_load, OBT_DC2K_provide_secure_algorithm_load, OBT_DC2K_provide_secure_CA_load, OBT_DC2K_provide_secure_key_exchange_keyset_load*) are all concerned with providing a means to cryptographically verify the authenticity of a piece of data. The first objective is satisfied by FCS_COP.1 [SGSS] and the last three are satisfied by the data authentication component of the functional requirement FCS_COP.1 [DC2K].

11.2.2 Dependencies of Functional Requirements for the DC2K

Reference [1] states that some IT functional requirements are dependent on others (and in addition, some dependencies themselves have further dependencies). The table below shows the dependencies for the DC2K IT functional requirements:

DC2K IT functional Requirements	Dependencies
FCS_CKM.1 [DC2K]	FCS_CKM.2
	FCS_CKM.4
	FMT_MSA.2
FCS_CKM.2 [DC2K]	FCS_CKM.1
	FCS_CKM.4
	FMT_MSA.2
FCS_CKM.4	FCS_CKM.1
	FMT_MSA.2
FMT_MSA.2	ADV_SPM.1
	FDP_ACC.1
	FMT_MSA.1
(ADV_SPM.1)	FMT_SMR.1 (see section 11.2.4)
FDP_ACC.1	FDP_ACF.1
FMT_SMR.1	FIA_UID.1
FMT_MSA.1	FDP_ACC.1
	FMT_SMR.1
FDP_ACF.1	FMT_MSA.3
	FDP_ACC.1
FIA_UID.1	No dependency
FMT_MSA.3	FMT_MSA.1
	FMT_SMR.1
FCS_COP.1 [DC2K]	FCS_CKM.1
	FCS_CKM.4
	FMT_MSA.2

Table 4 – DC2K IT Functional Requirement Dependencies (claimed IT functional requirements in **bold type**).

Table 4 shows the dependencies of components (with iterated dependencies of dependencies). It should be noted that only those components in bold, i.e. FCS_CKM.1 [DC2K], FCS_CKM.2 [DC2K] and FCS_COP.1 [DC2K] have been claimed as IT functional requirements. Taking each (unclaimed) dependency in turn, the following sections provide a rationale as to why these dependencies are inappropriate and/or irrelevant in the context of the DataCryptor 2000 evaluation.

11.2.2.1 Cryptographic Key Destruction

FCS_CKM.4

The DataCryptor 2000 disables cryptographic keys as a result of key expiry or a deletion request from the user. However, such disabling does not constitute "key destruction" as such, it simply ensures that the keys are unavailable for subsequent use by the product.

Unlike a standard software system or product, the DataCryptor employs physical protection measures to prevent both unauthorised and authorised access to cryptographic key values (see section 8.1.1.5). This means that FCS_CKM.4 is effectively subsumed by FPT_PHP.3 (Resistance to Physical Attack). Thus dependency FCS_CKM.4 is not relevant in the context of DataCryptor 2000.

11.2.2.2 Access Control, User Identification, Management of Security Attributes and Security Management Roles

FDP_ACC.1, FDP_ACF.1, FIA_UID.1, FMT_MSA.1, FMT_MSA.2, FMT_MSA.3, FMT_SMR.1

In the manner in which they are described in [1], access control, user identification, security management roles and management of security attributes, are functions that are most meaningful in the context of a typical software security product or system. In such a situation, users might log on to accounts using unique user IDs and passwords, and they may be restricted to only performing certain actions (e.g. read, write) on files with certain ownership criteria (e.g. owner, group, all).

In the context of the TOE however, access control, identification and authorisation is applied to the capability to view and alter security attributes such as key lifetimes, unit alarm settings etc., (rather than to the information under protection itself). Such tasks are performed by establishing an encrypted "management session" between a management centre and the unit under management. Individuals performing these tasks are simply considered as authorised or unauthorised, and as stated in section 5.1.5, it is assumed that only authorised individuals have access to the key material. Individuals without access to the appropriate key material (i.e. unauthorised individuals) are unable to manage the box in such a way as to view or alter sensitive information.

In this way, the DataCryptor's claimed IT functional requirements of cryptographic operation, cryptographic key distribution, cryptographic key generation and resistance to physical attack provide all the necessary support for dependencies FDP_ACC.1, FDP_ACF.1, FIA_UID.1, FMT_MSA.1, FMT_MSA.2, FMT_MSA.3, and FMT_SMR.1. In the manner in which they are described in [1], these dependencies are inappropriate for this target of evaluation.

11.2.3 Dependencies of Functional Requirements for the SGSS

The table below shows the dependencies for the SGSS IT functional requirements:

SGSS IT functional requirements	Dependencies
FCS_COP.1 [SGSS]	FCS_CKM.1

	FCS_CKM.4 FMT_MSA.2
FPT_PHP.3 [SGSS]	No dependencies

Table 5 – SGSS IT Functional Requirement Dependencies (claimed IT functional requirements in **bold type**).

Arguing as in sections 11.2.2.1 and 11.2.2.2 the FCS_CKM.4 and FMT_MSA.2 dependencies do not apply.

Furthermore the FCS_CKM.1 dependency does not apply since the public key that is used by the data authentication algorithm is preloaded in the factory and therefore cryptographic key generation techniques are not used by the SGSS.

11.2.4 Assurance Requirements

The assurance requirements specified in this security target are exactly those specified by the Evaluation Assurance Level 5. The actual Evaluation Assurance Level required for the evaluation is specified in reference [39] – it may be any EAL up to and including 5. Where the EAL is 5, this set is consistent and mutually supportive. All dependencies are implicitly met by inclusion of the dependent component itself or a stronger component from the same assurance family within the set. Where the EAL is lower than 5, the actual evaluation assurance requirements are a subset of those at EAL 5, and are therefore at least met, if not exceeded by those specified in this document.

Choice of the assurance component set of the EAL defined in [39] is appropriate to meet the requirements of the company's customers.

11.2.5 Security Requirements are Mutually Supportive and Internally Consistent

The security requirements do not conflict as they apply to distinct but related operations. FCS_CKM.1 and FCS_CKM.2 apply to the generation and distribution of keys respectively. They support each other in the overall objective of secure key management.

FCS_COP.1 applies to the operations of data authentication, data encryption, and key exchange protocol. These do not conflict with FCS_CKM.1 and FCS_CKM.2.

The final security requirement, FPT_PHP.3, is concerned with resistance to physical attack and clearly does not conflict with the other requirements, all of which are related to cryptographic services.

The preceding shows that the set of security requirements is internally consistent. It also shows that they are mutually supportive in that they support each other where necessary.

11.3 Target of Evaluation Summary Specification Rationale

11.3.1 Satisfaction of TOE Security Functional Requirements

IT Functional Requirements	TOE Security Functions
FCS_CKM.1 [DC2K]	*SF_SGSS_Random_Number_Generator* *SF_DC2K_key_exchange_algorithm* *SF_DC2K_encryption_algorithm*
FCS_CKM.2 [DC2K]	*SF_DC2K_key_exchange_algorithm*
FCS_COP.1 [DC2K]	*SF_DC2K_data_authentication_implementation*
	SF_DC2K_encryption_algorithm
FCS_COP.1 [SGSS]	*SF_DC2K_key_exchange_algorithm* *SF_SGSS_data_authentication_implementation*
FPT_PHP.3 [SGSS]	*SF_SGSS_alarm_circuitry*

Table 6 – Use of TOE Security Functions to meet IT functional Requirements

Table 6 lists each of the IT Functional Requirements identified in section 8.1.1, and identifies all the TOE security functions needed to meet that requirement. Since every requirement is met by one or more security functions, the security functions are sufficient to meet all of the IT Functional Requirements.

In addition, where each IT Functional Requirement is met by exactly one security function (and assuming the requirement is valid), it follows that the security functions must be necessary as well as sufficient to counter the threat.

The suitability of the security functions to meet the IT Functional Requirements is shown as follows.

The functional requirement FCS_CKM.1 [DC2K] (Cryptographic Key Generation) is met by the security functions *SF_SGSS_Random_Number_Generator*, *SF_DC2K_key_exchange_algorithm* and *SF_DC2K_encryption_algorithm*. *SF_SGSS_Random_Number_Generator* provides a hardware random number generator for use by *SF_DC2K_key_exchange_algorithm* and *SF_DC2K_encryption_algorithm* in the generation of KEKs and DEKs respectively.

The functional requirement FCS_CKM.2 [DC2K] (Cryptographic Key Distribution) is met by the security function *SF_DC2K_key_exchange_algorithm* which provides an implementation of a secure key exchange algorithm (see 9.1.5).

The functional requirement FCS_COP.1 [DC2K] (Cryptographic Operation) is met by three security functions, namely *SF_DC2K_key_exchange_algorithm*, *SF_DC2K_encryption_algorithm* and *SF_DC2K_data_authentication_implementation*. It is clear from previous sections of this document that all three functions are required to support the

security objectives met by FCS_COP.1. Therefore the three security functions are necessary and sufficient to counter the threat.

The functional requirement FCS_COP.1 [SGSS] (Cryptographic Operation) is met by the security function *SF_SGSS_data_authentication_implementation* as can be seen by comparing sections 8.1.1.4 and 9.1.2.

Finally the functional requirement FPT_PHP.3 (Resistance to Physical Attack) is met by the security function *SF_SGSS_alarm_circuitry* as can be seen by comparing 8.1.1.5 and 9.1.3.2.

It follows that the set of security functions is both necessary and sufficient to support the IT Functional Requirements. Note also that the security functions work together so as to satisfy the Functional Requirements (i.e. the security functions do not conflict with each other and are mutually supportive in satisfying the Functional Requirements).

11.3.2 Compliance of Assurance Measures with Assurance Requirements

Section 9.2 lists every assurance requirement, and separately addresses the assurance measures for each. Since this provides a one-to-one mapping between assurance requirements and assurance measures, (and assuming the suitability of each assurance measure), the set of assurance measures must provide full compliance with the set of assurance requirements.

DataCryptor 2000 Version Under Evaluation (Common Criteria)

Thales e-Security
The Sussex Innovation Centre
Science Park Square
Sussex University
Falmer
Brighton
East Sussex
BN1 9SB

Tel: +44 (0) 1273 384600
Fax: +44 (0) 1273 384601

Document History

Issue	Date	Description
00A	16th October 2001	Initial Release
00B	4th December 2001	Updated following changes in level of CAPS re-use possible
00C	12th January 2004	Updated section 2 to include hardware part numbers of the different variants

Contents

1 Introduction

The DataCryptor 2000 is a range of network encryptors that supports several different network protocols and cryptographic algorithms.

This document should be read in conjunction with 0562a218, the DataCryptor 2000 Security Target (Common Criteria). It defines the DataCryptor 2000 version to be evaluated under the Common Criteria.

2 Version Under Evaluation

Target of Evaluation Title: DataCryptor 2000

Top Level Part Number: 1600X320

Hardware Part Numbers: 1600A321
1600B321
1600E321

Version Number: DataCryptor 2000 Application Software 3.3.

DataCryptor 2000 Protocols Under Evaluation (Common Criteria)

Thales e-Security
The Sussex Innovation Centre
Science Park Square
Sussex University
Falmer
Brighton
East Sussex
BN1 9SB

Tel: +44 (0) 1273 384600
Fax: +44 (0) 1273 384601

Document History

Issue	Date	Description
00A	16th October 2001	Initial Release
00B	4th December 2001	Updated following changes in level of CAPS re-use possible
00C	10th March 2003	Removal of Channelised variant
00D	28th January 2004	Removal of X.25 variant

Contents

1 Introduction

The DataCryptor 2000 is a range of network encryptors that supports several different network protocols and cryptographic algorithms.

This document should be read in conjunction with 0562a218, the DataCryptor 2000 Security Target (Common Criteria). It defines the DataCryptor 2000 network protocols to be evaluated under the Common Criteria.

2 Protocols Under Evaluation

The following protocols should be evaluated under the Common Criteria:

Link
Frame Relay
IP

DataCryptor 2000 Cryptographic Algorithms Under Evaluation (Common Criteria)

Thales e-Security Ltd
4th/5th Floors
149 Preston Road
Brighton
BN1 6BN

Tel: +44 (0) 1273 384600
Fax: +44 (0) 1273 384601

Document History

Issue	Date	Description
00A	16th October 2001	Initial Release
00B	7th March 2003	Revised

Contents

1 Introduction

The DataCryptor 2000 is a range of network encryptors that supports several different network protocols and cryptographic algorithms.

This document should be read in conjunction with 0562a218, the DataCryptor 2000 Security Target (Common Criteria). It defines the DataCryptor 2000 cryptographic algorithms to be evaluated under the Common Criteria.

2 Cryptographic Algorithms Under Evaluation

The following algorithms should be evaluated under the Common Criteria:

Key Exchange Algorithms:

Diffie-Hellman (ANSI X9.42 Hybrid1)

Data Encryption Algorithms:

Triple DES (Data Encryption Standard, as specified in FIPS PUB 46-3)

Data Authentication Algorithms:

DSA (Digital Signature Algorithm, as specified in FIPS PUB 186-2)

Data Hashing Algorithms:

SHA-1 (Secure Hash Algorithm, as specified in FIPS PUB 180-1)

DataCryptor 2000 – Evaluation Assurance Level

Thales e-Security Ltd
4th/5th Floors
149 Preston Road
Brighton
BN1 6AS

Tel: +44 (0) 1273 384600
Fax: +44 (0) 1273 384601

Document History

Issue	Date	Description
00A	4th December 2001	Initial Release
00B	2nd April 2004	Address updated EAL4 added

Contents

1 Introduction

The DataCryptor 2000 is a range of network encryptors that supports several different network protocols and cryptographic algorithms.

This document should be read in conjunction with 0562a218, the DataCryptor 2000 Security Target (Common Criteria). It defines the DataCryptor 2000 Evaluation Assurance Level under the Common Criteria.

2 Evaluation Assurance Level

Target Of Evaluation Title: DataCryptor 2000

Part Number: 1600x320

Evaluation Assurance Level: 5

Note that for purposes of international recognition an Evaluation Assurance Lever of 4 is also defined.

Key Management

Specification

Racal-Airtech Ltd
The Sussex Innovation Centre
Science Park Square
University of Sussex
Falmer
East Sussex
BN1 9SB

Tel: +44 (0) 1273 384600
Fax: +44 (0) 1273 384601

Document History

Issue	Date	Description
00A	22 November 1995	Re-named and formatted - see Issue Authority 1985-1995
00B	23 November 1995	Revision following review
00C	15 December 1995	Re-ordering
00D	24 April 1996	Incorporate 0557A197.00B
00E	23 January 1997	Correct 3.1.4 diagram, and footers
00F	30 April 1998	Added DH parameter support (non-CA) Incorporate SDT implementation links
00G	24 November 1998	Revised to include SafeDial long-term fix
00H	3 February 2000	Explain DEK challenge/response
00I	17 March 2000	Update KEK exchange description to include hashing

Contents

1. Related Documents

NIST FIPS PUB 186

Digital Signature Standard, US Department of Commerce, May 1994
{This publication describes DSA as it is used in any product implementing the KMS}

NIST FIPS PUB 180-1

Secure Hash Standard, April 1995
{This publication describes SHA-1, as used in any product implementing the KMS}

2. Glossary

ACK	Acknowledge
CA	Certification Authority
CBC	Cipher Block Chaining
Certificate	Signed public key information
Challenge	Data to transformed to demonstrate correct key
DEK	Data Encryption Key
DSA	Digital Signature Algorithm
ECB	Electronic Code Book
ECB3, CBC3	Triple encryption using EDE double length key
EDE	Encrypt Decrypt Encrypt
IV	Initial Vector
KEK	Key Encryption Key
KMKEK	Key Management KEK
NAK	Negative ACK
Response	Transformed challenge
SHA	Secure Hash Algorithm
[xxx]yyy	ECB encrypt xxx using key yyy
3[xxx]yyy	ECB3 encrypt xxx using key yyy
{xxx}yyy	CBC MAC xxx using key yyy and IV of all zeroes.
3{xxx}yyy	CBC3 MAC xxx using key yyy and IV of all zeroes.
xxx XOR yyy	xxx Xor combined with yyy
D_{ECB} [xxx]yyy	ECB decrypt xxx using key yyy
D_{CBC} [xxx]yyy	CBC decrypt xxx using key yyy

3. Overview

This document details a key management system to be used as a basis for the majority of future Racal-Airtech encryption products.

The mechanism described is X.509 compliant. All data exchanges are therefore defined using ASN.1, and data structures from X.509 are used where relevant.

The mechanism is largely detailed from a general case point of view. Smaller systems can, however, use a simpler interpretation - e.g. only holding a single key at a particular point in the key hierarchy, rather than the implied key directories.

The rest of this section provides an overview of each of the tiers of the encryption key hierarchy. This overview describes the cryptography from the point of view of a unit at, for example, one end of a communication link.

3.1 User Data Encryption (DEK)

All user data is encrypted using a symmetric algorithm, normally operating in an 8-bit cipher-feedback mode. The running IV is updated after each user data packet, i.e. the user data is cryptographically equivalent to a contiguous stream of data.

The DEK is the key used to encrypt user data. This DEK is either randomly generated or, for algorithms where random generation is not possible, exchanged at connection establishment.

A DEK is normally only valid for a single connection.

3.2 DEK Exchange (KEK)

To protect the exchange of DEKs, a second symmetric key (the KEK) is used. This key is used to encrypt key data in transit across a connection.

The unit imposes lifetime constraints on the KEK. This is normally a number of hours or days after the KEK is loaded. The lifetime is applied independently by, and is a separately configurable item for, each unit.

Under certain circumstances, the KEK may have a zero lifetime, i.e. it is used to exchange a DEK and then immediately discarded. The KEK layer is still present under these circumstances, for consistency.

The KEK tier is present in the key hierarchy for two purposes:

1. Public key techniques are relatively slow, and may impose unacceptable delays if performed every time a connection is established. As it is a symmetric key, DEK data may be exchanged using the KEK quickly.

2. The public key transport provides a random KEK. This would be unsuitable where the data encryption method is not capable of supporting random keys. In such circumstances, the KEK layer can be implemented using an alternative symmetric algorithm that is capable of using random keys, whilst still using the required user data encryption method.

3.3 KEK Exchange

To allow the secure exchange of KEKs, an extended public key exchange process is used.

This extended form allows for a random input into each key exchange, along with a fixed & certifiable input. The random portion is discarded after the key exchange.

The fixed & certifiable portion of the public key data is the units public/secret key data, with the public part being the certifiable portion. This key data has a lifetime defined by the certificate that certifies it.

The public key transport mechanism generates a random KEK. This new KEK replaces any previously exchanged for a particular peer connection.

3.4 Key Certification

The units public key data is certified using a CA public key set. The certificate is generated by a CA.

Before two units can communicate, they must exchange the certified unit public key data. When the certificate is exchanged, it is checked using the public key algorithm and the relevant CA public key. This ensures that the unit & its key data is valid & a member of the appropriate CA group.

Having checked the certificate, the data can be used to exchange a KEK. The data may also optionally be kept for future key exchanges.

Each certificate has a start & end date. The certificate & any data signed within that certificate are not used outside these limits.

3.5 “Commissioning” data

All units within a user group hold the same CA public key data, allowing all members of the user group to exchange keys and thus provide secured communication within that group.

Units are added to a user group by providing the unit with the CA public key for that group, and subsequently providing the unit public key data signed by that CA.

A unit may hold more than one CA public key set, along with appropriately certified key data. This allows a unit to talk to members of more than one user group.

3.6 Symmetric CA Key Management

An extra level is optionally provided in the key hierarchy, where public key techniques are not considered suitable for the top level of the key hierarchy.

A triple length symmetric master key (KMKEK) is present within the unit. New CA public key sets and unit key sets can be loaded, encrypted and authenticated using this master key.

When a new CA key data is loaded using this mechanism, all current CA and unit key information is erased, to be replaced by the new key data.

The KMKEK may also be replaced by supplying a new KMKEK encrypted and authenticated using the current KMKEK.

When a unit is manufactured, a default manufacturers KMKEK is implanted within the unit.

This KMKEK process is only available when a symmetric algorithm is present within the unit.

4. Detail of Operations

This section describes the sequencing & data transfer of key exchanges and other cryptographic operations.

Textual descriptions are given for all operations. Unit - unit and Unit - CA exchanges are also detailed using message sequence charts.

Both the textual descriptions and message sequence charts assume that the update proceeds to completion. If an exchange should fail, a presentation layer NAK will be issued by one of the parties, containing a numeric identification of the error. Such failure paths are omitted from this document for clarity.

The overall scheduling of key update actions, along with error recovery procedures, are not defined in this document. The intent is that systems request and retry key exchanges as required, with no user intervention. For example, if a unit wishes to establish a connection, and it has no KEK with which to exchange the DEK, a KEK update should be automatically instigated. The only point that user intervention should be required is when an unrecoverable failure has occurred, or the failure indicates possible cryptographic attack.

In general, where key data is transferred between units, fixed size data packets are used, with random padding inserted to fill the data out to the fixed size. This avoids information about key length being available to an attacker.

4.1 DEK exchange.

Four types of DEK exchange are defined, depending on the application.

1. Balanced random key exchange
2. Unidirectional random key exchange
3. Pre-generated DEK exchange
4. Unidirectional pre-generated DEK exchange

All DEK exchanges use the current KEK to protect the DEK in transit, and also transfer the IV to use for that connection. They therefore must share a previously exchanged KEK, and also must share a common data encryption algorithm.

4.1.1 Balanced Random DEK Exchange

Balanced random DEK exchange involves both units on a connection having random input into the DEK generation process. This is the preferred method where applicable.

The DEK exchange consists of the following sequence of operations (See also the following message sequence chart):

1. The units exchange their names. Each unit generates a list of currently loaded KEK ids that are shared between both units. This list normally has a single entry.

2. The units exchange this list of KEK ids. Both units generate a list of KEKs that are present in both. This list normally has a single entry. Should more than one KEK be present in both units, the KEK that was most recently updated is selected.

3. The units exchange a list of available data encryption algorithms. The units generate a list of encryption algorithms common to both. The highest priority encryption algorithm is selected for use on this connection (Encryption algorithms have a numeric priority tag as an attribute).

4. Both units generate and exchange random key and IV data. Four items are included in this data:

   Transmit key data (TK)
   Transmit IV data (TIV)
   Receive key data (RK)
   Receive IV data (RIV)

5. Both units derive their working keys as follows:

   $\text{Transmit DEK} = [\text{self.TK XOR } [\text{peer.RK}]_{KEK}]_{KEK}$
   $\text{Transmit IV} = [\text{self.TIV XOR } [\text{peer.RIV}]_{KEK}]_{KEK}$
   $\text{Receive DEK} = [[\text{self.RK}]_{KEK} \text{ XOR peer.TK}]_{KEK}$
   $\text{Receive IV} = [[\text{self.RIV}]_{KEK} \text{ XOR peer.TIV}]_{KEK}$

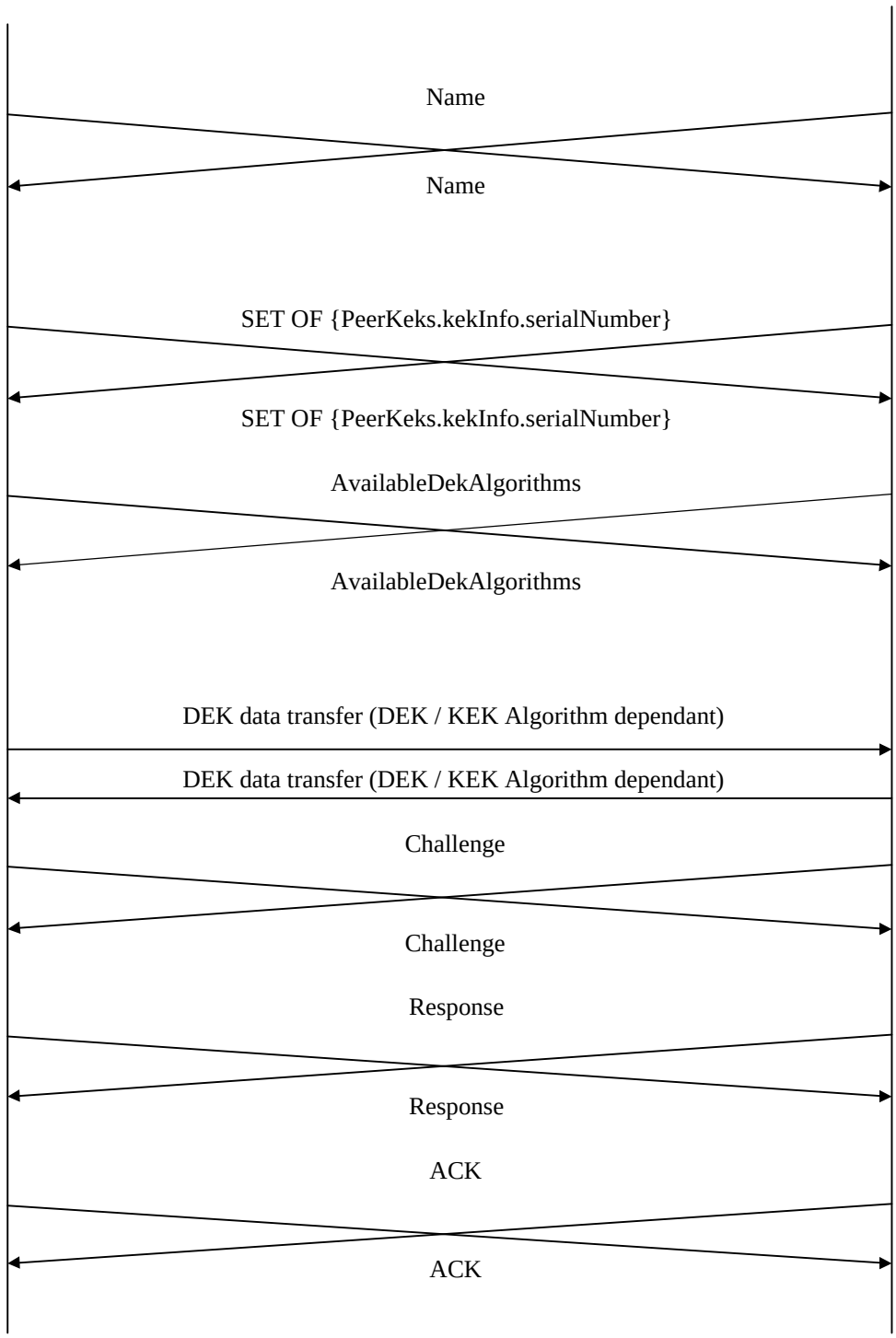
6. To ensure that the same key data is installed at both units, each unit issues a random challenge, which is random data decrypted with then Receive DEK. The peer unit generates the response:

   $\text{Response} = [D_{CBC}[\text{Challenge}_{Peer}]_{\text{Receive DEK}} \text{ XOR Transmit IV}]_{\text{Transmit DEK}}$

7. The unit verifies the response:

   $\text{Check response} = [D_{CBC}\,[\text{Challenge}_{Self}]_{\text{Transmit DEK}} \text{ XOR Receive IV}]_{\text{Receive DEK}}$

8. If all is correct, the unit issues an ACK. On reception of the corresponding peer ACK, the connection is ready to securely transfer user data.


Unit A
Unit B
Name
Name
SET OF {PeerKeks.kekInfo.serialNumber}
SET OF {PeerKeks.kekInfo.serialNumber}
AvailableDekAlgorithms
AvailableDekAlgorithms
DEK data transfer (DEK / KEK Algorithm dependant)
DEK data transfer (DEK / KEK Algorithm dependant)
Challenge
Challenge
Response
Response
ACK
ACK

4.1.2 Unidirectional Random DEK Exchange

Unidirectional random DEK exchange involves only one unit on a connection having random input into the DEK generation process. This method is intended for communication mechanisms where the readily available return data path is either very slow or non-existent, most notably for EMAIL applications. This method still derives transmit & receive keys, however the receive keys are often vestigial, having no use.

The unit generating the key data is the master unit , the other is the slave in this transaction.

The DEK exchange consists of the following sequence of operations (See also the following message sequence chart):

1. The master unit selects the KEK that was most recently updated and the highest priority algorithm it considers available. Identification of both items are sent to the slave.

2. The master unit generates random key and IV data and sends it to the slave. Four items are included in this data:

   Transmit key data (TK)
   Transmit IV data (TIV)
   Receive key data (RK)
   Receive IV data (RIV)

3. The master unit derives its working keys as follows:

   Transmit DEK = $[TK]_{KEK}$
   Transmit IV = $[TIV]_{KEK}$
   Receive DEK = $[RK]_{KEK}$
   Receive IV = $[RIV]_{KEK}$

4. On reception of the data, the slave unit derives its working keys as follows:

   Transmit DEK = $[RK]_{KEK}$
   Transmit IV = $[RIV]_{KEK}$
   Receive DEK = $[TK]_{KEK}$
   Receive IV = $[TIV]_{KEK}$

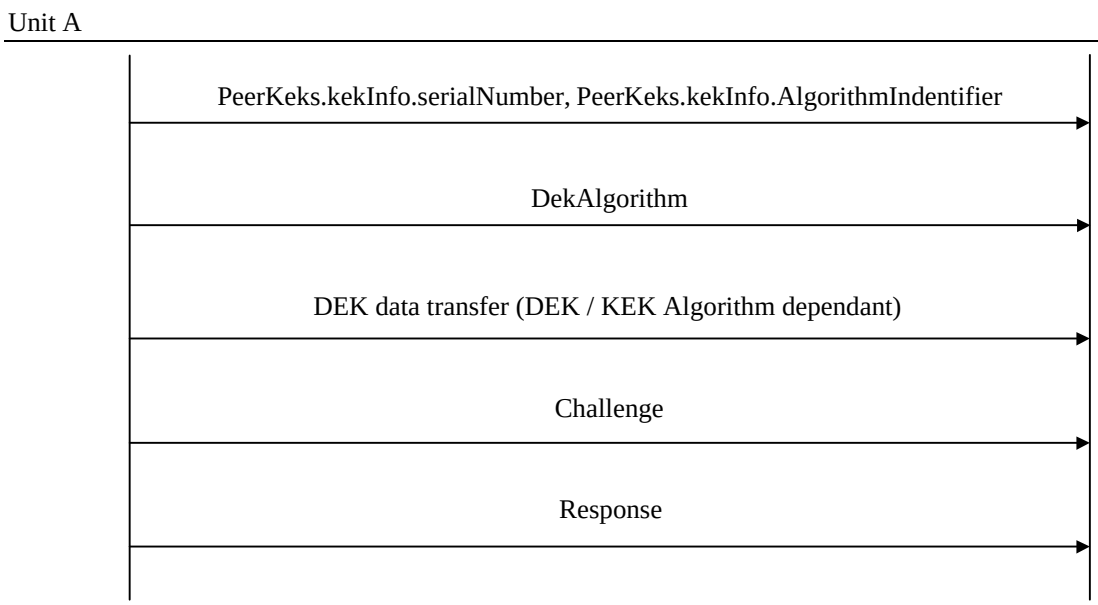
5. To ensure that the same key data is installed at both units, the master unit issues a random challenge along with the required response:

   Response = $[\text{Challenge XOR Transmit IV XOR Receive IV}]_{(\text{Transmit DEK XOR Receive DEK})}$

6. The slave unit verifies the response:

   Response = $[\text{Challenge XOR Transmit IV XOR Receive IV}]_{(\text{Transmit DEK XOR Receive DEK})}$

7. If all is correct, the connection is ready to accept user data.


Unit A
Unit B
PeerKeks.kekInfo.serialNumber, PeerKeks.kekInfo.AlgorithmIndentifier
DekAlgorithm
DEK data transfer (DEK / KEK Algorithm dependant)
Challenge
Response

4.1.3 Pre-generated DEK Exchange

Pre-generated DEK exchange is to be used when random DEKs are not feasible, for example if the raw key contains check data precluding random generation.

The DEK exchange consists of the following sequence of operations (See also the following message sequence chart):

1. The units exchange their names. Each unit generates a list of currently loaded KEK ids that are shared between both units. This list normally has a single entry.

2. The units exchange their currently loaded KEK ids. Both units generate a list of KEKs that are present in both. This list normally has a single entry. Should more than one KEK be present in both units, the KEK that was most recently updated is selected.

3. The units exchange a list of available data encryption algorithms. The units generate a list of encryption algorithms common to both. The highest priority encryption algorithm is selected for use on this connection (Encryption algorithms have a numeric priority tag as an attribute).

4. The units exchange the attributes of the most suitable DEK available at each unit (Note: The trivial case exists where only one unit has any DEKs available). The units use a fixed arbitration algorithm to determine the DEK to use. This arbitration algorithm is for further study, but uses expiry dates of keys and key IDs as its input.

5. The unit that holds the selected DEK transfers the key, encrypted using the KEK, along with IV data:

   Transmit IV data (TIV)
   Receive IV data (RIV)

6. The master unit derives its working IVs as follows:

   Transmit IV = $[TIV]_{KEK}$
   Receive IV = $[RIV]_{KEK}$

7. On reception of the data, the slave unit derives its working IVs as follows:

   Transmit IV = $[RIV]_{KEK}$
   Receive IV = $[TIV]_{KEK}$

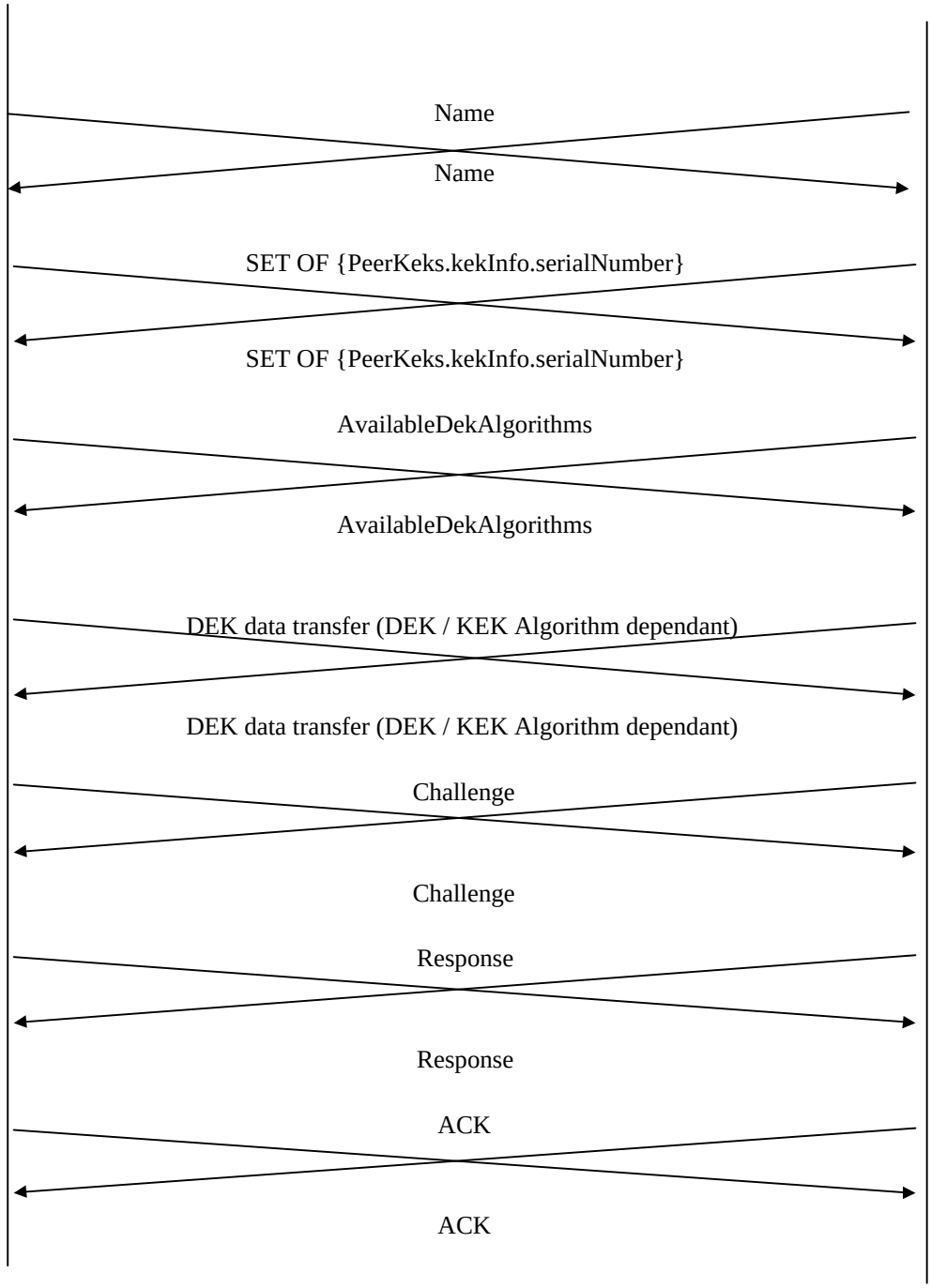
8. To ensure that the same key data is installed at both units, each unit issues a random challenge. The peer unit generates the response:

   Response = $[Challenge_{Peer} \text{ XOR } \text{Transmit IV}]_{DEK}$

9. The unit verifies the response:

   Check response = $[Challenge_{Self} \text{ XOR } \text{Receive IV}]_{DEK}$

10. If all is correct, the unit issues an ACK. On reception of the corresponding peer ACK, the connection is ready to securely transfer user data.


Unit A
Unit B
Name
Name
SET OF {PeerKeks.kekInfo.serialNumber}
SET OF {PeerKeks.kekInfo.serialNumber}
AvailableDekAlgorithms
AvailableDekAlgorithms
DEK data transfer (DEK / KEK Algorithm dependant)
DEK data transfer (DEK / KEK Algorithm dependant)
Challenge
Challenge
Response
Response
ACK
ACK

4.1.4 Unidirectional Pre-generated DEK Exchange

Unidirectional pre-generated DEK exchange is to be used when random DEKs are not feasible and the communication mechanism's readily available return data path is either very slow or non-existent.

The unit supplying the key data is the master unit , the other is the slave in this transaction.

The DEK exchange consists of the following sequence of operations (See also the following message sequence chart):

1. The master unit selects the KEK that was most recently updated and the highest priority algorithm it considers available. Identification of both items are sent to the slave.

2. The master unit determines the required DEK, and transfers the key, encrypted using the KEK, along with IV data:

   Transmit IV data (TIV)
   Receive IV data (RIV)

3. The master unit derives its working IVs as follows:

   Transmit IV = $[TIV]_{KEK}$
   Receive IV = $[RIV]_{KEK}$

4. On reception of the data, the slave unit derives its working IVs as follows:

   Transmit IV = $[RIV]_{KEK}$
   Receive IV = $[TIV]_{KEK}$

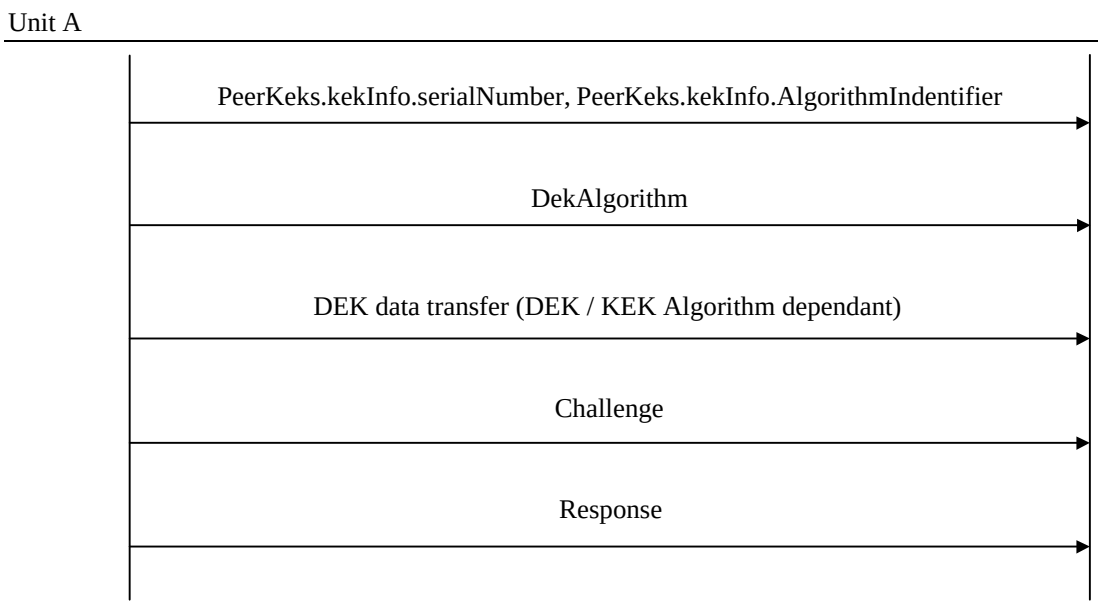
5. To ensure that the same key data is installed at both units, the master unit issues a random challenge along with the required response:

   Response = $[\text{Challenge XOR Transmit IV XOR Receive IV}]_{DEK}$

6. The slave unit verifies the response:

   Response = $[\text{Challenge XOR Transmit IV XOR Receive IV}]_{DEK}$

7. If all is correct, the connection is ready to accept user data.


Unit A
Unit B
PeerKeks.kekInfo.serialNumber, PeerKeks.kekInfo.AlgorithmIndentifier
DekAlgorithm
DEK data transfer (DEK / KEK Algorithm dependant)
Challenge
Response

4.2 KEK Exchange

The KEK exchange requires two elements:

1. A fixed, signed, data set. This includes secret and public parts of a public key set and a signed public copy of the public key.

2. A random data set, generated every time a KEK exchange is performed.

To transfer a KEK, the units must:

1. Both contain key data signed under the same CA.

2. Both share the same CA certification algorithm.

3. Both share the same KEK exchange algorithm.

4. Both share a KEK symmetric data encryption algorithm.

The units exchange the methods used for certification and KEK exchange during the key exchange. Thus multiple algorithms can be supported.

KEK exchange has two distinct component operations:

1. Public data exchange.

2. Random KEK exchange/derivation

4.2.1 Public Data Exchange

Before a KEK can be exchanged, the certified public key data from each unit must be exchanged. Depending on the implementation, this may be performed once, and the certified key data held in a directory for future use, or every time a KEK exchange is performed.

Exchange of the certified data allows each unit to verify the trustworthiness of the peer unit (i.e. that both validly belong to the same CA group).

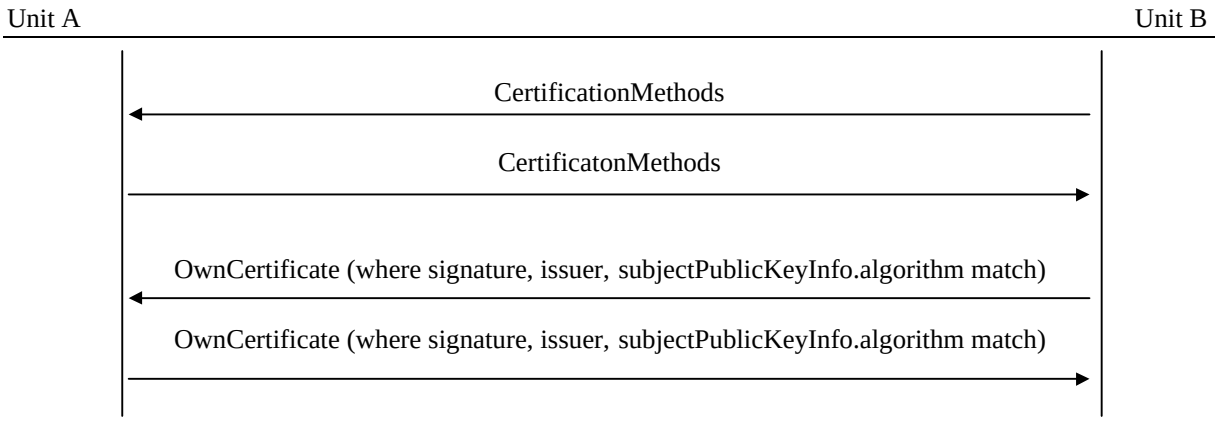
The public key exchange consists of the following sequence of operations (See also the following message sequence chart):

Each unit sends a list of available certification methods. Each entry in the list contains the following items:

- Certification method (Certificate.signature)
- CA identity (Certificate.issuer)
- KEK exchange method(s) Certificate.subjectPublicKeyInfo.algorithm

Each unit selects one of its certified public keys that match one of the entries in the above list supplied by the peer. The unit returns this certificate to the peer unit.

On receiving the peer units certificate, it is validated using the relevant CA and algorithm, and is then available for use in random KEK exchanges.



When both units have no OwnCertificate available to send (and they have no OwnCertificates) at all they can revert to 'straight Diffie-Hellman' as the exchange method see Section 4.2.3 Non-certificate based KEK-Exchange, Page 19.

4.2.2 Random KEK Exchange/Derivation

The KEK exchange consists of the following sequence of operations (See also the following message sequence chart):

1. Each unit sends a list of unit key sets that the unit holds (That is the units own keysets, not certified portions of other units key sets). Each entry of the list is of the type CertificateSerialNumber.

2. On receiving the peer units key set list, each unit creates a list of all certified peer public keys that it holds which appear in the list. Each unit sends this list to its peer.

3. Each unit then creates a list of candidate public keys to use for the key exchange. Each entry in the list contains key set pairs (Units keyset & peers keyset) with the following attributes:
   - Peers selected keyset is within certified unit public key list sent to the peer.
   - Units own selected keyset is within certified peer key list from the peer.
   - Both keyset algorithms match (Certificate.subjectPublicKeyInfo.algorithm)

4. If more than one candidate is found, the units use a fixed arbitration algorithm to determine the PK to use. This arbitration algorithm is as follows: First, select only the candidates(s) with the latest expiry date. If more than one candidate remains, select the certificate with the smallest certificate ID.

5. The units exchange a list of available key encryption algorithms. Each unit generates a list of key encryption algorithms common to both. The highest priority key encryption algorithm is selected for use with this KEK (Key encryption algorithms have a numeric priority tag as an attribute).

6. Each unit generates a random value and encrypts or transforms it (depending on the PK algorithm). This value is sent to the peer.

7. Both units now calculate a common value combining the two random values.

8. The KEK is defined as a portion of KEKdata.

9. A temporary key is extracted for KEK validation - KVAL. This key is a twice the length of the KEK, containing all the KEK bits, along with an equal number of other bits from KEKdata. This double length key is used to allow a challenge - response KEK check, without leaving the opportunity of a known plaintext attack on the single length key. A 32 bit key ID is also extracted from KEKdata, to allow unique identification of the KEK.

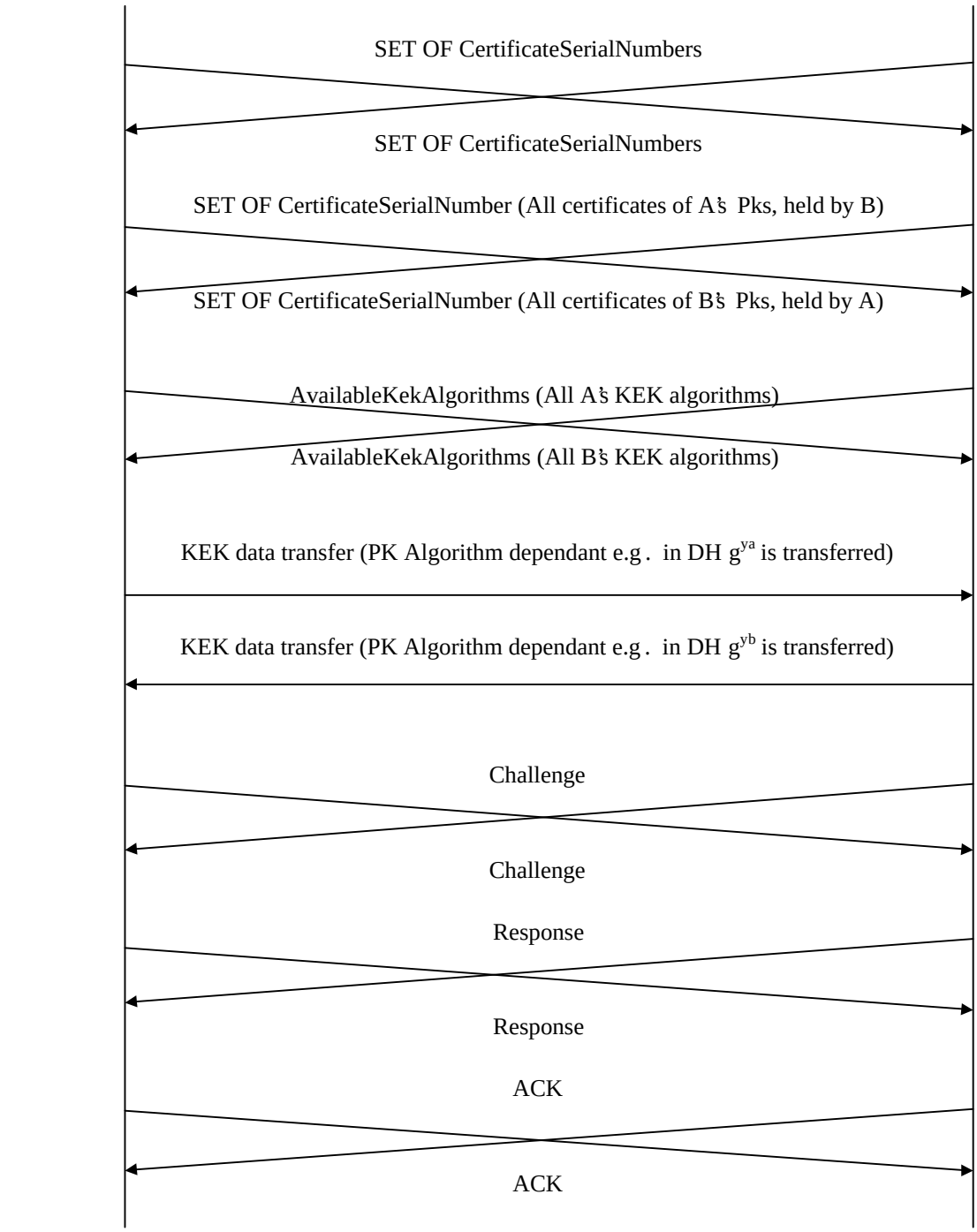
10. To ensure that the same key data is installed at both units, each unit issues a random challenge. The peer unit generates the response:

$$\text{Response} = 3[\text{Challenge}]_{KVAL}$$

11. The unit verifies the response:

$$\text{Response} = 3[\text{Challenge}]_{KVAL}$$

12. If all is correct, the unit issues an ACK and the KEK is available for DEK transfer.



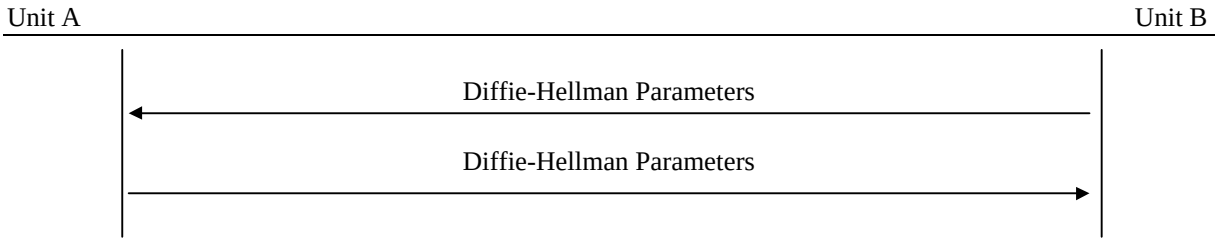
4.2.3 Non-certificate based KEK-Exchange

When the units both contain no OwnCertifiactes at all they can use Diffie-Hellman parameters that were previously loaded into the device signed by a Certificate Authority (i.e. a member of AuthCerts probably Manufacturer). See Section 4.3.6 Diffie-Hellman parameter installation, page 23.
Note: This should only occur when there are no OwnCertificates in either unit - even if they are not yet valid or are for the 'wrong' algorithm.

1. The unit transmits it's Diffie-Hellman parameters and receives the parameters from the Peer unit.

2. If the two sets of Diffie-Hellman parameters are the same continue with the KEK exchange.

3. If the parameter sets a different stop with an error.



4.3 CA Certificate Management

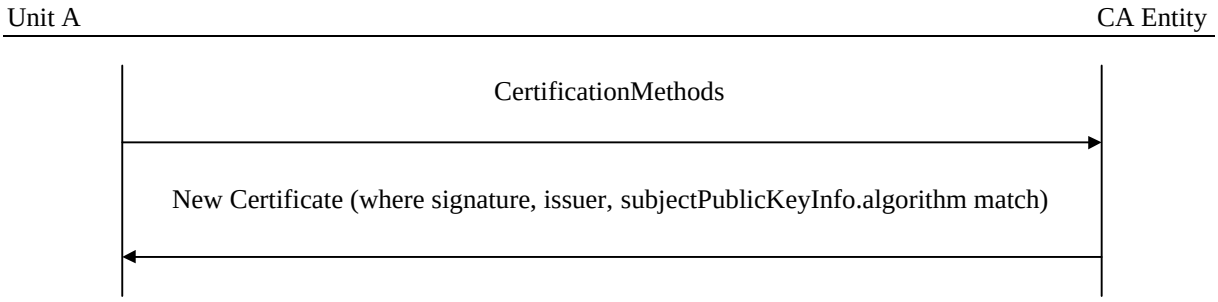
CA key sets & associated certificates & key sets are managed by the following set of functions:

1. CA Transfer
2. Add CA
3. Delete CA
4. Certify Key Set
5. Delete Key Set
6. Diffie-Hellman parameter installation

When a unit is manufactured, or should the situation arise where no CA public key data is present, a default manufacturers CA public key set is implanted within the unit.

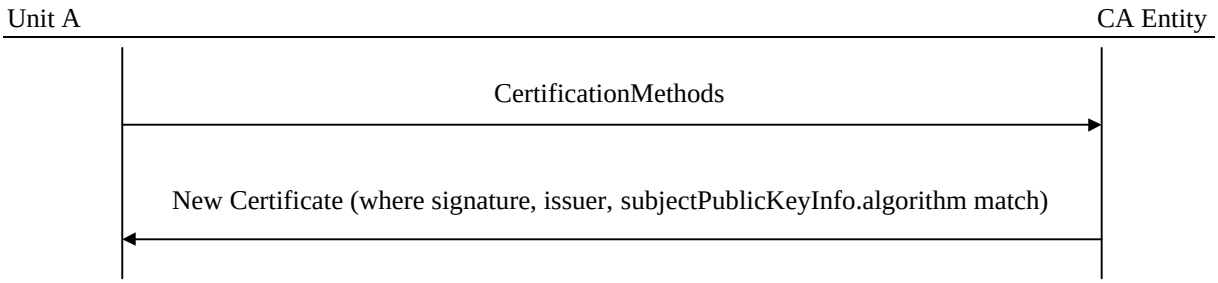
4.3.1 CA Transfer

Transferring from one CA key set to another. The unit supplies a list of the current CA public key sets, each indicating CA name, certificate signature algorithm and public key exchange algorithm. The unit is provided with the new CA public key data signed by a current CA secret key set. The authority that signed the new certificate is replaced by the new certificate.



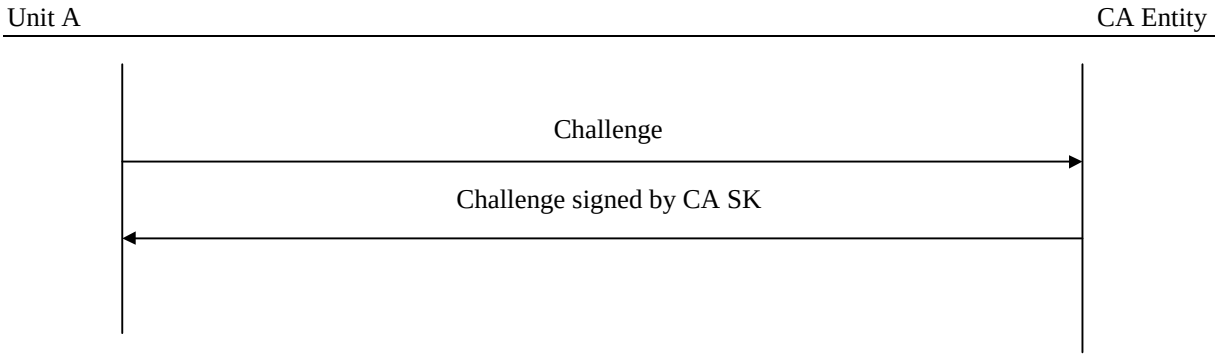
4.3.2 Add CA

Adding a new CA public key set. The unit supplies a list of the current CA public key sets, each indicating CA name, certificate signature algorithm and public key exchange algorithm. The unit is provided with the new CA public key data signed by a current CA secret key set. The new key set is added to the set of current CA key sets. The current CA key set is unaffected by this operation. This operation is not required to be supported by all implementations.



4.3.3 Delete CA

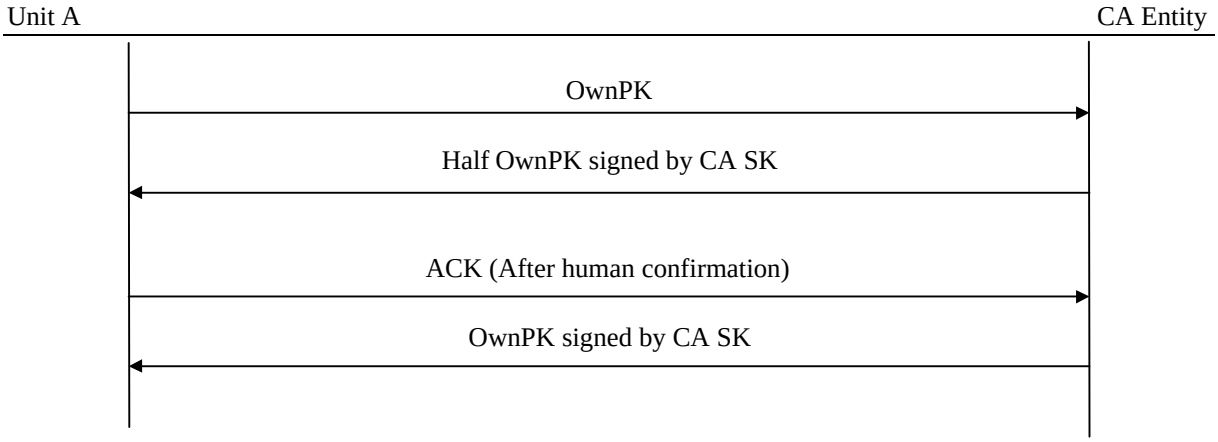
Delete CA public key set. The unit supplies a list of the current CA public key sets, each indicating CA name, certificate signature algorithm and public key exchange algorithm. The unit issues a random challenge, and is provided with the challenge signed by a current CA secret key set. The authority certificate that signed the challenge is then deleted.



4.3.4 Certify Key Set

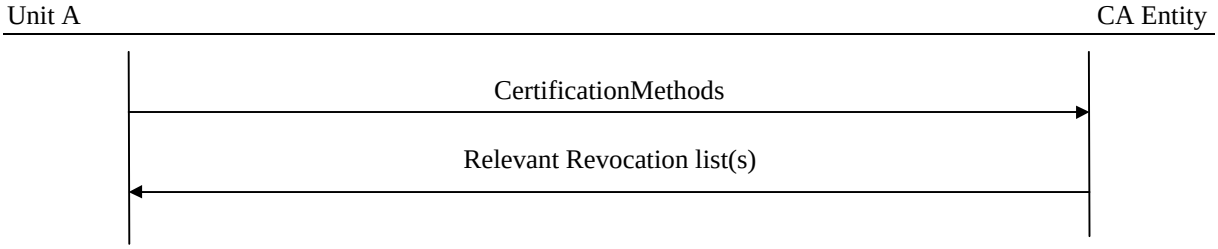
Generate a new random & certified unit key set. Having generated a random unit key set, the unit sends the public key to the CA entity for certification using the CA secret key.

The CA entity initially signs the first half of the key data, and returns this signature to the unit. The unit validates that the signature is correct and the certified data matches the first half of the public key. On successful validation of this, the CA entity then certifies the entire key data, and returns the certificate to the unit. Note that the confirmation of successful validation of the first half of the key set is a manual, not electronic, mechanism. This method allows for secure remote certification (commissioning) of units, as long as a separate trusted human path is available for the confirmation.



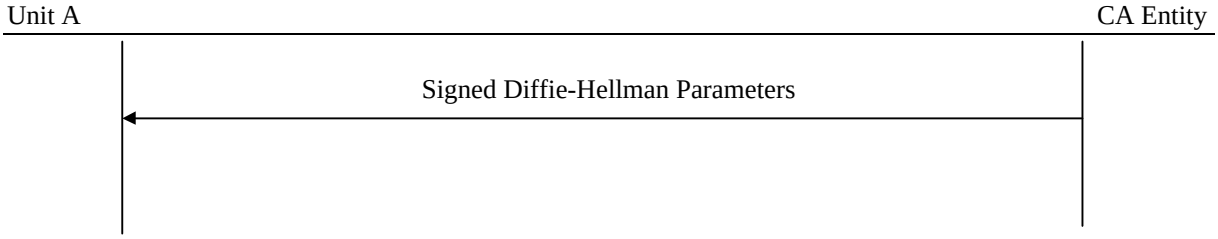
4.3.5 Delete Key Set

Erase unit key set. If the unit receives a valid key set revocation request, signed by the CA secret key, the unit deletes all unit key sets identified in the revocation request.



4.3.6 Diffie-Hellman parameter installation

Loads the Diffie-Hellman parameters to be used when no Certificates are available. If any certificates are installed (even if not valid or the 'wrong' algorithm) this data will not be used and cannot be loaded. The Diffie-Hellman parameters are signed by an Authority Certificate, if the unit successfully verifies the signature the Diffie-Hellman parameter data is stored and is available for use under the correct circumstances. There is only one set of default Diffie-Hellman data - any new set replaces a previously installed set.



5. Key Management Specification in ASN.1

```
KeyManagementSpecification
DEFINITIONS ::=
BEGIN

-- EXPORTS ALL --

IMPORTS
        Name
                FROM InformationFramework informationFramework

        Certificate, Validity, CertificateSerialNumber
                FROM AuthenticationFramework authenticationFramework

-- algorithms --

dsa             ALGORITHM   ::= { DSAParameters IDENTIFIED BY id-dsa}

dsa-with-sha    ALGORITHM   ::= { DSAParameters IDENTIFIED BY id-dsa-with-sha}

sha             ALGORITHM   ::= {NULL IDENTIFIED BY id-sha}

dh              ALGORITHM   ::= {DHParameters  IDENTIFIED BY id-dh}

-- object identifier assignments --

secsig          OBJECT IDENTIFIER   ::= {iso(1) identified-organisation(3) oiw(14) secsig(3)}
algorithm       OBJECT IDENTIFIER   ::= {secsig 2}
id-dsa          OBJECT IDENTIFIER   ::= {algorithm 12}
id-dsa-with-sha OBJECT IDENTIFIER   ::= {algorithm 13}
id-sha          OBJECT IDENTIFIER   ::= {algorithm 18}
id-dh           OBJECT IDENTIFIER   ::= {algorithm 999}

DSAParameters ::=       SEQUENCE {
                                modulusLength   INTEGER,    -- length of p in bits
                                prime1          INTEGER,    -- modulus p
                                prime2          INTEGER,    -- modulus q
                                base            INTEGER     -- base g
                        }

DHParameters ::=        SEQUENCE {
                                modulusLength   INTEGER,    -- length of p in bits
                                prime           INTEGER,    -- modulus p
                                base            INTEGER     -- base g
                        }

-- Certificate Lists --

-- List of  certificates of Certificate Authority certificates --
AuthorityCertificates   ::= SET OF Certificate

-- List of  certificates of own Public keys Certified by different Authorities/Algorithms --
OwnCertificates         ::= SET OF Certificate
```

```
-- List of  certificates of Public Keys of peer units Certified by verifiable Authorities/Algorithms --
PeerCertificates        ::= SET OF Certificate

-- List of  serial numbers of Public Keys cerificates of peer units --
PeerCertificateIDs      ::= SET OF CertificateSerialNumber

-- Kek Certification Methods --
-- A list of the methods of key certification available to a unit --
CertificationMethods    ::= SET OF { KeyCertificationMethod }

KeyCertificationMethod  ::= SEQUENCE {
                                issuer                  Name,
                                signature               AlgorithmIdentifier,
                                publicKeyAlgorithm      AlgorithmIdentifer}

-- KEK Specifications --
-- A list of the KEKs known to the unit --
AllKeks                 ::= SET OF PeerKeks

-- A list of the KEKs between the unit and a single peer --
PeerKeks                ::= SEQUENCE{
        selfSubject             Name,
        peerSubject             Name,
        kekInfo                 SET OF {KekInfo}}

KekInfo                 ::= SEQUENCE{
        serialNumber            KekSerialNumber
        validity                Validity,
        symmetricKekInfo        SymmetricKekInfo}

SymmetricKekInfo        ::= SEQUENCE{
        algorithm               KekAlgorithmIdentifier,
        symmetricKey            BIT STRING}

-- 32 bit  number --
KekSerialNumber         ::= KeySerialNumber

CertificateNames ::= SEQUENCE {
        self            Name,
        issuer          Name}

KekAlgorithmIdentifier  ::= SEQUENCE{
        algorithm       ALGORITHM.&id({SupportedKekAlgorithms}),
        parameters      ALGORITHM.&Type({SupportedKekAlgorithms}{@algorithm}) OPTIONAL}

SupportedKekAlgorithms ALGORITHM  ::= {dsa | dsa-with-sha  | sha  | dh }

AvailableKekAlgorithms  ::= SEQUENCE OF KekAlgorithmIdentifier

AvailableKeks ::= SET OF KekSerialNumber

-- DEK Specifications--
-- A list of the DEKs known to the unit --
AllDeks                 ::= SET OF PeerDeks
```

```
-- A list of the DEKs between the unit and a single peer --
PeerDeks            ::= SEQUENCE{
        selfSubject             Name,
        peerSubject             Name,
        dekInfo                 SET OF {DekInfo}}

DekInfo             ::= SEQUENCE{
        serialNumber                    DekSerialNumber,
        validity                        Validity,
        symmetricDekInfo                SymmetricDekInfo}

-- 32 bit number --
DekSerialNumber     ::= KeySerialNumber

SymmetricDekInfo    ::= SEQUENCE{
        serialNumber            DekSerialNumber
        algorithm               DekAlgorithmIdentifier,
        symmetricKey            BIT STRING}

DekAlgorithmIdentifier  ::= SEQUENCE{
        algorithm       ALGORITHM.&id({SupportedDekAlgorithms}),
        parameters      ALGORITHM.&Type({SupportedDekAlgorithms}{@algorithm}) OPTIONAL}

SupportedDekAlgorithms ALGORITHM ::=  {dsa | dsa-with-sha  | sha  | dh }

AvailableDekAlgorithms  ::= SEQUENCE OF DekAlgorithmIdentifier

AvailableDeks ::= SET OF DekSerialNumber

--General Specifications--

KeySerialNumber     ::= INTEGER (0..4294967295)

Challenge           ::= ENCRYPTED{ SEQUENCE {
                            padding         RandomNumber,
                            keyId           KeySerialNumber}}

RandomNumber        ::= BIT STRING

ChallengeResponse   ::= ENCRYPTED{TRANSFORMED{DECRYPTED{Challenge}}}

KeyData             ::= SEQUENCE {
                            transmitKeyData         BIT STRING,
                            transmitIVData          BIT STRING,
                            receiveKeyData          BIT STRING,
                            receiveIVData           BIT STRING}

Ack     ::= '♠'-- ASCII char 06 --
Nak     ::= SEQUENCE {
                nak             '§',            --ASCII char 15 --
                error           ErrorCode}

ErrorCode ::-   INTEGER (1-256)

END
```

6. X.509 Annex A - Authentication Framework in ASN.1

```
AuthenticationFramework{joint-iso-ccitt ds(5) module(1) authentication Framework(7) 2}
DEFINITIONS ::=
BEGIN

--EXPORTS ALL-

IMPORTS
        id-at, informationFramework, upperBounds, selectedAttributeTypes, basicAccessControl
                FROM UsefulDefinitions {joint-iso-ccitt ds(5) module(1) usefulDefinitions(0) 2}

        Name, ATTRIBUTE
                FROM InformationFramework informationFramework

        ub-user-password
                FROM UpperBounds upperBounds

        AuthenticationLevel
                FROM BasicAccessControl basicAccessControl

        Uniqueldentifier, octetStringMatch
                FROM SelectedAttributeTypes selectedAttributeTypes;

--types--

Certificate                     ::=     SIGNED { SEQUENCE {
        version                 [0]     Version DEFAULT v1,
        serialNumber                    CertificateSerialNumber,
        signature                       AlgorithmIdentifier,
        issuer                          Name,
        validity                        Validity,
        subject                         Name,
        subjectPublicKeyInfo            SubjectPublicKeyInfo,
        issuerUniqueIdenfier    [1]     IMPLICIT UniqueIdentifier OPTIONAL,
                                                -- if present, version must be v2 --
        subjectUniqueIdentifier [2]     IMPLICIT UniqueIdentifier OPTIONAL
                                                -- if present, version must be v2 --}}

Version                 ::=     INTEGER {v1(0), v2(1)}

CertificateSerial Number ::=    INTEGER

AlgorithmIdentifier             ::=     SEQUENCE {
        algorithm               ALGORITHM.&id({SupportedAlgorithms}),
        parameters              ALGORITHM.&Type({SupportedAlgorithms}{@algorithm})
OPTIONAL}
-       Definition of the following information object set is deferred, perhaps to standardised
-       profiles or to protocol implementation conformance statements. The set is required to
-       specify a table constraint on the parameters component of AlgorithmIdentifier.
-       SupportedAlgorithms     ALGORITHM ::= {...|...}

Validity                ::=     SEQUENCE{
```

```
        notBefore       UTCTime,
        notAfter        UTCTime}

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

Certificates            ::=     SEQUENCE
        userCertificate                 Certificate,
        certificationPath               ForwardCertificationPath OPTIONAL}

ForwardCertificationPath ::=    SEQUENCE OF CrossCertificates

CertificationPath       ::=     SEQUENCE{
        userCertificate                 Certificate,
        theCACertificates               SEQUENCE OF CertificatePair OPTIONAL}

CrossCertificates       ::=     SET OF Certificate

CertificateList         ::=     SIGNED { SEQUENCE {
        signature                       AlgorithmIdentifier,
        issuer                          Name,
        thisUpdate                      UTCTime,
        nextUpdate                      UTCTime OPTIONAL,
        revokedCertificates             SEQUENCE OF SEQUENCE {
                userCertificate                 CertificateSerialNumber,
                revocationDate                  UTCTime } OPTIONAL}}

CertificatePair         ::=     SEQUENCE{
        forward [0]     Certificate OPTIONAL,
        reverse [1]     Certificate OPTIONAL
                        -- at least one of the pair shall be present -- }

-- attribute types --

userPassword            ATTRIBUTE       ::= {
        WITH SYNTAX                     OCTET STRING (SIZE(0..ub-user-password))
        EQUALITY MATCHING RULE          octetStringMatch
        ID                              id-at-userPassword}

userCertificate         ATTRIBUTE       ::= {
        WITH SYNTAX             Certificate
        ID                      id-at-userCertificate{

cACertificate           ATTRIBUTE       ::= {
        WITH SYNTAX             Certificate
        ID                      id-at-cAcertificate }

authorityRevocationList ATTRIBUTE       ::= {
        WITH SYNTAX             CertificateList
        ID                      id-at-authorityRevocationList}

certificateRevocationList ATTRIBUTE     ::= {
        WITH SYNTAX             CertificateList
        ID                      id-at-certificateRevocationList}
```

```
crossCertificatePair            ATTRIBUTE    ::=  {
        WITH SYNTAX             CertificatePair
        ID                      id-at-crossCertificatePair}

-- information object classes --

ALGORITHM   ::=     TYPE-IDENTIFIER

-- parameterized types --

HASHED{ToBeHashed}  ::=     OCTET STRING ( CONSTRAINED-BY {
        -- must be the result of applying a hashing procedure to the --
        -- BER-encoded (see 8.7) octets --
        -- of a value of -- ToBeHashed})

ENCRYPTED{ToBeEnciphered}   ::=     BIT STRING ( CONSTRAINED BY {
        -- must be the result of applying an encipherment procedure --
        -- to the BER-encoded octets of a value of -- ToBeEnciphered})

SIGNED{ToBeSigned}  ::=     SEQUENCE{
        toBeSigned          ToBeSigned,
        COMPONENTS OF       SIGNATURE{ToBeSigned}}

SIGNATURE{OfSignature}      ::=     SEQUENCE{
        algorithmIdentifier AlgorithmIdentifier,
        encrypted           ENCRYPTED { HASHED { OfSignature}}}

-- object identifier assignments --

id-at-userPassword              OBJECT IDENTIFIER   ::=     {id-at 35}
id-at-userCertificate           OBJECT IDENTIFIER   ::=     {id-at 36}
id-at-cAcertificate             OBJECT IDENTIFIER   ::=     {id-at 37}
id-at-authorityRevocationList   OBJECT IDENTIFIER   ::=     {id-at 38}
id-at-certificateRevocationList OBJECT IDENTIFIER   ::=     {id-at 39}
id-at-crossCertificatePair      OBJECT IDENTIFIER   ::=     {id-at 40}

END
```

7. X.509 Annex H - Reference definition of algorithm object identifiers

```
AlgorithmObjectIdentifiers {joint-iso-ccitt ds(5) module(1) algorithmObjectIdentifiers(8) 2}
DEFINITIONS ::=
BEGIN

--EXPORTS All--

IMPORTS
      algorithm,authenticationFramework
            FROM UsefulDefinitions {joint-iso-ccitt-ds(5) module(1) usefulDefinitions(0) 2}
      ALGORITHM
            FROM AuthenticationFramework authenticationFramework;

-- categories of object identifiers --

encryptionAlgorithm       OBJECT IDENTIFIER       ::=     {algorithm 1}
hashAlgorithm             OBJECT IDENTIFIER       ::=     {algorithm 2}
signatureAlgorithm        OBJECT IDENTIFIER       ::=     {algorithm 3}

-- synonyms --

id-ea                     OBJECT IDENTIFIER       ::=     encryptionAlgorithm
id_ha                           OBJECT IDENTIFIER         ::=     hashAlgorithm
id_sa                     OBJECT IDENTIFIER       ::=     signatureAlgorithm

-- algorithms --

rsa   ALGORITHM                 ::=           {
      KeySize
      IDENTIFIED BY id-ea-rsa}

KeySize     ::=           INTEGER

-- object identifier assignments --

id-ea-rsa                 OBJECT IDENTIFIER       ::=     {id-ea 1}

-- the following object identifier assignments reserve values assigned to deprecated functions --

id-ha-sqMod-n             OBJECT IDENTIFIER       ::=     {id-ha 1}
id-sa-sqMod-nWithRSA      OBJECT IDENTIFIER       ::=     {id-sa 1}

END
```

8. Recommended Techniques

This section details the techniques that are recommended for use with this specification.

8.1 CA Certification

CA certification is performed using the DSA signature standard. DSA key length is normally 1024/160 bits.

8.2 KEK Exchange

KEK exchange is performed using an extended Diffie-Hellman method. The extended Diffie-Hellman key data contains two elements:

1. A fixed, signed, data set. This includes the standard Diffie-Hellman parameters g & modulus p, along with a random secret exponent x, and a signed public copy of (g^x) mod p.

2. A random data set, generated every time a KEK exchange is performed.

8.2.1 Random KEK Exchange/Derivation

1. Each unit generates a random value y and calculates (g^y) mod p. This value is sent to the peer.

2. Both units now calculate:

$Z_x = (g^{(xpeer * xself)})$ mod p, and
$Z_y = (g^{(ypeer * yself)})$ mod p
for t = 0, … , n {
$SHA_t = SHA(Z_x || Z_y || t)$
}
$KEKdata = SHA_0 || SHA_1 || \ldots ||SHA_n$

where:

g^{xpeer} is the value received in the certificate exchange,
g^{ypeer} is the value received above,
xself is the units private key,
yself is the random number generated above,
t is a 32-bit integer,
n is the number of SHA operations required to provide a sufficient quantity of KEKdata, and
|| denotes standard bit string concatenation.

The above calculation results in the same value being calculated by both units, since:

$$(g^{xpeer} \bmod p)^{xself} = (g^{xself} \bmod p)^{xpeer} = (g^{(xpeer * xself)} \bmod p) = Z_x \text{, and}$$
$$(g^{ypeer} \bmod p)^{yself} = (g^{yself} \bmod p)^{ypeer} = (g^{(ypeer * yself)} \bmod p) = Z_y$$

8.3 Code Loading

To allow for the loading of encryption algorithms, and software updates, a unit may allow a code download process.

New software or encryption algorithms are downloaded either:

1. Plaintext, signed using a currently loaded CA public key set.

2. Encrypted and authenticated using the current KMKEK. Note that this method is only available when a symmetric encryption algorithm has been loaded into the unit & the KMKEK functionality is available.

8.4 Protection of Portable Devices

To prevent the theft of portable devices, such devices are protected using a password. After connection to the relevant host equipment, the password must be submitted before secure communication is allowed. A limit is imposed on unsuccessful password attempts. Exceeding this limit will lock the unit.