Secure Generic Sub-System (SGSS)

Cryptographic Module

Security Policy

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

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

Document History

Version	Date	Description
001	13 October 1999	FIPS 140-1 level 4 Cryptographic Module Security Policy
002	29 October 1999	Updated to incorporate the formal aspects of CMSP
003	9 November 1999	Modified following comments by Cygnacom
004	23 November 1999	Modified to more closely relate to FSM
005	9 December 1999	Modified following further comments by Cygnacom
006	1 February 2000	Updated in an attempt to make the whole document set more generic
007	2 May, 2000	Modified following comments from CygnaCom
008	10 May, 2000	Copyright banner removed
009	29 August, 2000	Updated to reflect responses to NIST questions
010	14 May, 2003	Company's name and address changed
011	11 March, 2005	Change SafeGuard Security Subsystem to Secure Generic Sub-System

Distribution List

Text	Location
Master Copy	Project log.

Contents

1. Glossary

CA	Certificate Authority
DSA	Digital Signature Algorithm
EDC	Error Detection Code
KAT	Known Answer Test
LED	Light Emitting Diode
SGSS	Secure Generic Sub-System
SHA-1	Secure Hashing Algorithm

2. Related documents

FIPS140-1	Federal Information Processing Standards Publication 140-1, Security Requirements for Cryptographic Modules
0550A109	Key Management Specification
0562A195	Secure Generic Sub-System (SGSS) FIPS 140-1 Supporting Documentation

3. Introduction

The purpose of this document is to provide the information required in order to satisfy the FIPS 140-1 requirements for a formal model of the Cryptographic Module Security Policy for submission of the Secure Generic Sub-System (SGSS) at level 4.

The SGSS is a multi-chip embedded module, as described in Secure Generic Sub-System (SGSS) FIPS 140-1 Supporting Documentation (0562A195).

The SGSS provides the functionality necessary to start an installed application, and it provides functionality to securely upgrade this application.

The SGSS is a level four multi-chip embedded module. The module is designed to lie inside a Datacryptor 2000 or similar device where the module will provide a highly secure security subsystem. In this capacity, the module provides the Datacryptor 2000 (or other such devices) a bootstrap capable of securely loading an application while in the field. Applications support services provided by the Datacryptor 2000. The bootstrap provides system initialization and transfer of control to the application. The bootstrap may also be used to load a new bootstrap or new application. The code that is loaded is signed with the Digital Signature Algorithm. Once the signature is verified, the new code becomes operational. The purpose of the SGSS is to securely validate the digital signature.

The private and the public key pair that is used to sign and verify the bootstrap of application is generated by the factory Certificate Authority (CA). The cryptographic officer loads the public key certificate into the module at the factory and the factory retains the private key.

The SGSS is a Level 4 module for the secure loading of applications. Once an application is loaded the SGSS functions as a component of the Datacryptor 2000 and may no longer function independently as a Level 4 module.

In normal operation, the user need not be aware of the existence of the bootstrap. Its use is restricted to configuration and maintenance tasks such as reading and updating configuration information and erasing and updating the loaded application.

When a user application is present, the bootstrap will provide basic system initialisation and transfer control to the application. The application is provided with an interface to the bootstrap via the trap 15 instruction.

To allow for erasure of a malfunctioning application, at system boot time a specific data exchange sequence is attempted using the bootstrap command protocol. If this succeeds, the application is erased and control returned to the bootstrap.

When no application is present the bootstrap operates in a command mode to allow loading of a new application.

The SGSS guarantees the integrity of any application loaded within the SGSS.

This document is a mixture of informal descriptive text and formal notation. The descriptive text is included to make the formal Z definitions more accessible to the reader.

4. Formal Aspects of the Security Policy

Only those aspects of the security policy that directly relate to FIPS 140-1 relevant aspects of the SGSS are specified using the Z notation. These aspects are:

- Crypto-officer role
- Show status
- Self-tests
- Cryptographic services
- Physical security

There are sections within this document relating to each of these aspects in the security policy. The formalisation of these aspects builds directly on their informal specification. In fact, this exercise simply reiterates the informal specification using the formal notation of Z, for these aspects.

5. Roles and Services

5.1 Roles

5.1.1 Crypto-Officer Role

The SGSS is required to contain a DSA public key. The purpose of this key is to enable the SGSS to verify the signature of any application that it is requested to load. The manufacturing plant requires a crypto-officer to load the SGSS public key certificate. Once the SGSS is fielded, new applications may be generated at the plant and signed using the private key. In the field, the user, acting on behalf of the factory, may load the signed application into the SGSS where the signature is verified using the public key.

5.1.2 User Role

An individual performing the user role, acting on behalf of the factory, is responsible for loading new application images into the SGSS using the commands specified in Section 11.

See section 5.2.2 for the formal specification of this role.

5.2 Services

The SGSS offers a number of services. The only cryptographic service is invoked when a certified application is loaded into the SGSS. However the other services operate in such a way as to maintain the SGSS in a secure state.

5.2.1 Self-Tests

At start-up the SGSS validates its application by checking it using an EDC algorithm. It also performs a known answer test (KAT) on the signature checking algorithms (DSA with SHA-1).

This is described formally in section 11.2.

5.2.2 Cryptographic services

Loading a new application is the only cryptographic service offered by the SGSS software. The SGSS checks the signature of an application before loading it. It uses DSA with SHA-1 and its own DSA certificate. It rejects any application whose signature fails to verify.

This is formally described in section 11.3.3.

5.2.3 Other

The user role may perform the following other non sensitive services:

- Echo (Echoes back an input string)
- Erase application (Erases application)
- Get version (Provides SGSS version number)
- Get CA name (Provides name of CA (factory) which generated public/private key pair)
- Read/Write configuration (Provides/selects current configuration)
- Reboot unit (Resets the unit)

5.3 Authentication

The module provides two roles (Crypto-officer and User), and only one identity, that of the factory. The Cryptographic Officer, acting on behalf of the factory, initializes the module by inserting the public key certificate into the module. The user, acting on behalf of the factory, loads the application. The identification and authentication of the factory is performed by the fact that the application is validated by means of the digital signature. Only the factory could have created the application that is successfully loaded into the module. Thus, the SGSS permits the factory to authenticate and load secure applications.

6. Physical Security

The SGSS is enclosed in a tamper resistant system that surrounds the secure area. This is called the alarm circuit. The circuit consists of an electronic wire grid, which is encased in a hard opaque epoxy. Breaking the wire grid will trigger the alarm circuit that will erase the contents of the RAM and cipher FPGA. Superficial tampering would mar the epoxy and could be detected by inspection. Any penetration significant enough to disturb the wire grid would erase the critical security parameters.

The alarm circuit is powered from the main power supply when this is available, but if the unit is not powered up then a battery powers it. If this battery is disconnected of fails the alarm triggers. Similarly, if the power levels surge or are actively driven above of below the normal levels, then the alarm circuit is triggered. The voltage protection is on the VCCC pin (+5V). The effects of triggering this are the same as for any other of the alarm circuit triggers. The alarm circuit is described in section 5.2.1.6 of the "Secure Generic Sub-System (SGSS) FIPS 140-1 supporting documentation". The alarm will trigger at some point between 6.5 and 7 volts on the VCC line. If this line drops below 4.5 volts, the microprocessor is not powered. In this case a battery powers the alarm circuit. If the battery line drops below 2.3 volts, then the alarm circuit is triggered. Additionally, a temperature sensor causes the alarm circuit to be triggered at temperatures above 60°C or below -5°C.

The effect of triggering the alarm is to erase the RAM and the FPGA, and isolate the interface lines of the SGSS.

Once an alarm has been triggered, the unit must be returned to the factory for the alarm to be reset.

PhysicalSecurity $\widehat{=}$ AlarmState

ALARM ::= alarmed | notAlarmed
POWER ::= acceptable | unacceptable
ENCLOSURE ::= intact | tampered
OperatingTemperature : -5°C . . 60°C

AlarmState

t? : Temperature p? : POWER e? : ENCLOSURE a! : ALARM
a! = (t? $\notin$ ran OperatingTemperature $\Rightarrow$ alarmed) $\vee$ (p? = unacceptable $\Rightarrow$ alarmed) $\vee$ (e? = tampered $\Rightarrow$ alarmed) $\vee$ notAlarmed

7. Software Security

See sections 5.2.2, 8, and 9.

8. Cryptographic Key Management

There is no cryptographic key management performed by the SGSS software.

The SGSS contains the public key component of its CA.

The SGSS CA can be changed only if a new certified SGSS application that contains a new CA is loaded to replace the existing SGSS that contains the existing CA public key.

9. Cryptographic Algorithms

The only cryptographic algorithm used by the SGSS is DSA with SHA-1 used to validate the signatures on any prospective application before loading it.

10. Self-test

See section 5.2.1.

11. Formal Description of the System Software

This section contains a complete Z specification of the software that makes up the SGSS product. Attempts have been made to group the information according to function and operational state.

Types:

OPERATIONAL_STATE ::= POS | TEST | CMD | LOAD | RUN | TRAP
PERSON ::= an individual
UNIT ::= SGSS product

CA_NAME ::= ASCII string

String ::= ASCII string

errorCheckResult ::= validEDC | invalidEDC

dsaKatResult ::= validKAT | invalidKAT

powerOnEraseRequest ::= ErasureRequested | RequestTimedOut

Trap15Command ::= GetSgssVersion | RebootUnit | EraseApplication | ReadAppConfig | WriteAppConfig | GetDramSize | CacheControl

CacheSetting ::= enable | disable | invalidate

booleanFlag ::= TRUE | FALSE

BAUD_RATE ::= 110 | 300 | 600 | 1200 | 2400 | 4800 | 9600 | 14400 | 19200 | 28800 | 38400 | 56000 | 57600 | 115200

Sets:

vApp = = set of all valid candidateApplication

invApp = = set of all invalid candidateApplication

App = = vApp ∪ invApp

unitPool = = set of all UNIT

cryptoOfficerPool = = set of all PERSON

Data:

CA_name : CA_NAME

currentBaudRate : BAUD_RATE

EraseApplicationRequest : powerOnEraseRequest

Version : String

CacheState : CacheSetting

DRAM : ℕ

System

cryptoOfficer : cryptoOfficerPool appInUse : vApp sgssApp : vApp opState : OPERATIONAL_STATE s : unitPool
#cryptoOfficer = 1 #s = 1

AssignCryptoOfficer

m? : cryptoOfficerPool
CryptoOfficer ≠ ∅ ⇒ cryptoOfficer = m

11.1 Initial State

InitialState

Δ System s? : unitPool
appInUse = ∅ cryptoOfficer? = ∅ s′ = s? CA_name = "Racal Manufacture" currentBaudRate = 38400

Version = //version specific string
CacheState = disabled
DRAM = amount of DRAM on mainboard

PowerOnState

Δ System
s? : unitPool
opState? : OPERATIONAL_STATE

if opState? = POS
    ((EraseApplicationRequest = ErasureRequested) ⇒ appInUse = ∅)
    ∨ opState! = TEST
fi

11.2 Self-Test

poSelfTest

Ξ System opState? : OPERATIONAL_STATE
if opState? = TEST ((errorCheckSGSS ∧ posSuccess) ∧ (dsaKat ∧ posSuccess) ∧ (errorCheckApp ∧ posSuccess) ⇒ opState! = RUN) ∨ ((errorCheckSGSS ∧ posSuccess) ∧ (dsaKat ∧ posSuccess) ∧ (errorCheckApp ∧ errorCheckAppErr) ⇒ opState! = CMD) ∨ ((errorCheckSGSS ∧ posSuccess) ∧ (dsaKat ∧ dsaKatErr) ⇒ opState! = POS) fi

errorCheckSGSS

Ξ System result! : errorCheckResult checksum_a, checksum_b, checksum, j : ℕ
checksum_a = 0xff checksum_b = 0xff j = sgssFlashStart do j ≤ sgssFlashEnd checksum_a = checksum_a + *j & 0xff checksum_b = checksum_b + checksum_a & 0xff od checksum = ((checksum_a << 8) | checksum_b) & 0xffff ((checksum = storedChecksum) ⇒ result! = validEDC) ∨ ((checksum ≠ storedChecksum) ⇒ result! = invalidEDC)

errorCheckApp

Ξ System result! : errorCheckResult checksum_a, checksum_b, checksum, j : ℕ
checksum_a = 0xff checksum_b = 0xff j = appFlashStart do j ≤ appFlashEnd checksum_a = checksum_a + *j & 0xff checksum_b = checksum_b + checksum_a & 0xff od checksum = ((checksum_a << 8) | checksum_b) & 0xffff ((checksum = storedChecksum) ⇒ result! = validEDC)

∨ ((checksum ≠ storedChecksum) ⇒ result! = invalidEDC)

dsaKat

Ξ System result! : dsaKatResult
validKAT ⇒ result! = posSuccess ∨ invalidKAT ⇒ result! = dsaKatErr

posSuccess

Ξ System r! : Rep
r! = OK

errorCheckSGSSErr

Ξ System r! : Rep
r! = errorCheckSGSSFailure

errorCheckAppErr

Ξ System r! : Rep
r! = errorCheckAppFailure

dsaKatErr

Ξ System r! : Rep
r! = dsaKatFailure

11.3 Load Application

Types:

block ::= 512 byte memory block

candidateApplication ::= array of 512 byte blocks

configuration_area ::= area of memory immediately before application

header_block ::= twenty bytes of data divided into five four-byte parameters

Data:

upload_in_progress : booleanFlag

currentConfig : configuration_area

11.3.1 Start upload

SetupConfigurationArea

params? : header_block config? : configuration_area
//implementation specific, sets up currentConfig to match config?

StartUpload

Δ System opState? : OPERATIONAL_STATE
if opState? = CMD ((GetApplicationSize < GetAvailableSpace) ∧ InsufficientStorageErr) ∨ (SetupConfigurationArea ⇒ (upload_in_progress = TRUE ∧ opState! = LOAD)) fi

11.3.2 Upload block

UploadBlock

Δ System opState? : OPERATIONAL_STATE index : ℕ appBlock : block
if opState? = LOAD candidateApplication[index] = appBlock fi

11.3.3 Complete upload

TotalLoadApplication ≙ (LoadApplication ∧ Success) ∨ InvalidSignatureErr
∨ InsufficientStorageErr

LoadApplication

Δ System opState? : OPERATIONAL_STATE a? : candidateApplication
if opState? = LOAD (a? ∈ vApp ⇒ (Success ∧ appInUse = a?)) ∨ (a? ∈ invApp ⇒ InvalidSignatureErr) upload_in_progress = FALSE opState! = CMD fi

Success

Ξ System a? : candidateApplication r! : Rep
a? ∈ vApp r! = OK

GetApplicationSize

size! : ℕ a? : candidateApplication //implementation specific

GetAvailableSpace

size! : ℕ s? : unitPool
//implementation specific

InvalidSignatureErr

Ξ System a? : candidateApplication r! : Rep
a? ∉ vApp r! = SignatureInvalid

InsufficientStorageErr

Ξ System a? : APPLICATION r! : Rep
r! = NotEnoughSpace

11.3.4 Cancel upload

CancelUpload

opState? : OPERATIONAL_STATE
if opState = LOAD upload_in_progress = FALSE opState! = CMD fi

11.4 Other Services

11.4.1 Echo

Echo

Ξ System in? : String out! : String

if opState? = CMD out! = in? fi

11.4.2 Reboot

Reboot
Δ System s? : unitPool opState? : OPERATIONAL_STATE
opState! = POS

11.4.3 Read Application Configuration

ReadApplicationConfig
Ξ System config! : configuration_area
if (opState? = CMD ∨ opState? = TRAP) config! = currentConfig fi

11.4.4 Write Application Configuration

WriteApplicationConfig
Δ System config? : configuration_area opState? : OPERATIONAL_STATE
if (opState? = CMD ∨ opState? = TRAP) currentConfig = config? fi

11.4.5 Set Comms Baud Rate

SetCommsBaudRate
Δ System baudrate? : BAUD_RATE opState? : OPERATIONAL_STATE
if opState? = CMD currentBaudRate = baudrate? fi

11.4.6 Get CA Name

GetCAname
Ξ System

name! : CA_NAME
if opState? = CMD   name! = CA_name fi

11.4.7 SGSSversion

SGSSversion

Ξ System opState? : OPERATIONAL_STATE ver! : String
if (opState? = CMD ∨ opState? = TRAP)   ver! = Version fi

11.5 Trap 15

Trap15

Ξ System s? : unitPool opState? : OPERATIONAL_STATE
if opState? = CMD   opState! = TRAP fi

HandleTrap15

opState? : OPERATIONAL_STATE command? : Trap15Command

if opState? = TRAP
  (command? = GetSgssVersion ⇒ (SGSSversion ∧ opState! = RUN))
  ∨ (command? = RebootUnit ⇒ opState! = POS)
  ∨ (command? = EraseApplication ⇒ (appInUse = ∅ ∧ opState! = POS))
  ∨ (command? = ReadAppConfig ⇒ (ReadApplicationConfig ∧ opState! = RUN))
  ∨ (command? = WriteAppConfig ⇒ (WriteApplicationConfig ∧ opState! = RUN))
  ∨ (command? = GetDramSize ⇒ (DRAMsize ∧ opState! = RUN))
  ∨ (command? = CacheControl ⇒ (Cache ∧ opState! = RUN))
fi

Cache

Δ System opState? : OPERATIONAL_STATE newSetting? : CacheSetting
if opState? = TRAP   CacheState = newSetting? fi

DRAMsize

Ξ System opState? : OPERATIONAL_STATE size! : ℕ
if opState? = TRAP size! = DRAM fi