GMES Sentinels Master Key Management Facility

A. Sisask[1]
Logica Deutschland GmbH & Co. KG, Darmstadt, Hesse, D-64295, Germany

M. Koller[2]
European Space Agency ESA/ESOC, Darmstadt, Hesse, D-64293, Germany

and

M. Götzelmann[3]
VEGA Space GmbH, Darmstadt, Hesse, D-64293, Germany

Security is becoming ever an ever more important and vital need for every mission operations concept. Together ESA, Logica and Vega have devised an effective and low cost operational solution with simplicity and ease of use to meet this challenge for the GMES Sentinel spacecraft. The authentication of telecommands is based on digital signature using symmetric cryptography. Two types of keys are used – master and session keys. Before the launch of the spacecraft all master keys need to be generated by the MKMF and exported to the on-board authentication unit. Session keys are used for actual telecommand authentication and are encrypted using master keys for upload to the spacecraft. MKMF features easy generation, secure storage and export of master and session keys, flexible key and metadata management, multi-mission support, secure backup and restore functions for mission specific or full data, simple verification of system integrity and non-repudiation. MKMF is a standalone desktop application deployed on a dedicated machine, isolated from any network and stored in restricted area. It uses ID Quantis True Random Number Generator device to generate cryptographically strong keys. The system can be easily used in any mission employing similar symmetric cryptography based concept.

I. Introduction

THE security concept for the Sentinels satellites of the GMES program was studied and designed couple of years ago. This is the first time when systematic security measures are applied to spacecraft operated by ESOC. The main goal was to ensure that whenever spacecraft receives a telecommand (TC) it can verify that the source of the TC is the actual operator and not any other party. Without any authentication in place one could relatively easily record and replay the TC packets sent to the spacecraft.

The authentication of TC-s is based on symmetric cryptography employing the AES standard. The technique supported by the on-board authentication unit (AU) is a "plain-text-with-appended-signature" system. It consists of appending an authentication trailer, i.e. a digital signature at the end of the TC. A logical authentication counter (LAC) is used to associate every TC segment with an authentication sequence number, to ensure that identical TC segments will not produce the same signature (except at large intervals of time), e.g. to ensure that recorded TC cannot be replayed by an attacker.

The system has two “levels” or two types of keys - master keys and session keys. Session keys are used for actual authentication of TC-s and can be uploaded to spacecraft during its operation. Master keys are used to encrypt session keys in order to securely upload those to the spacecraft but also to authenticate AU control commands in exceptional situations, for instance when no session keys are available. Master keys are imported to the on-board authentication unit on ground before the flight and can’t be uploaded to the spacecraft later. Session keys are stored

[1] Project Manager, Rheinstrasse 95, Darmstadt, Hesse, D-64295, Germany
[2] Data Systems Manager, Earth Observation Mission Data Systems Section (HSO-GDE), Robert-Bosch-Str. 5, D-64293 Darmstadt, Germany
[3] Principal Consultant Technology, Europaplatz 5, 64293 Darmstadt, Germany

in a volatile reprogrammable (RAM) memory while the master keys are stored in an external PROM memory and as such cannot be reprogrammed.

Each Sentinels spacecraft embarks two AU-s. Each AU includes its own memory areas for the storage of the master and sessions keys. However, an approach in which the same master keys and session keys are stored on both AU units is considered more robust and operationally sound. In particular, it facilitates the overall key management by ground and simplifies the recovery actions in case of troubleshooting or anomaly occurrence. There are three main facilities which implement this security concept as a whole:

1) The Master Key Management Facility (MKMF) which generates and stores the master and session keys on ground,
2) The Key Management Facility (KMF) which, as part of mission control system (MCS), performs the authentication related activities of TC-s on the ground using session keys and uploads encrypted session key blocks and
3) On-board Authentication Unit which performs the encryption and decryption operations on board of spacecraft.

The following diagram gives high-level overview of the three facilities and the key exchange between them.

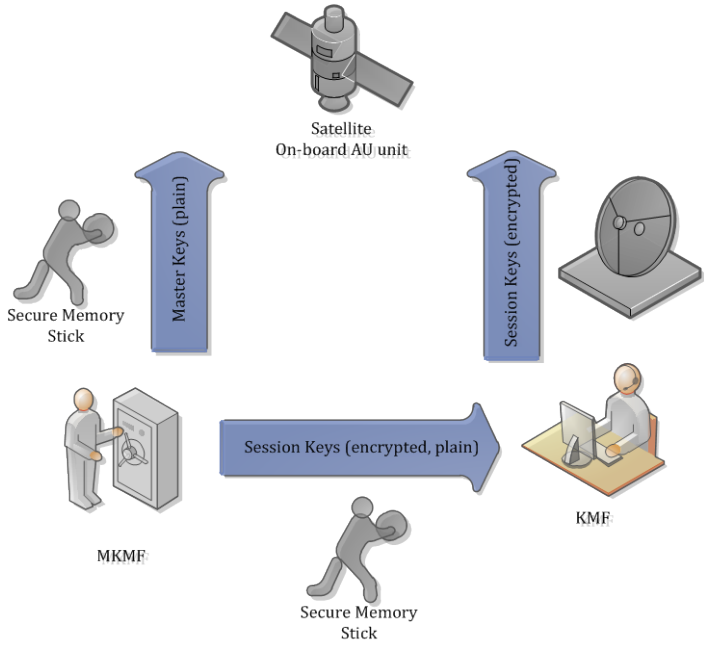

Figure 1 - Overview of Security Facilities

This paper focuses on implementing one of the facilities - the MKMF.

II. Requirements for MKMF

A. Role of MKMF

It is relatively easy to define the role of MKMF. It should generate cryptographically strong AES keys, protect them from being stolen and make securely available to the other parties of the security "chain" - AU and KMF. In theory, MKMF could be completely independent but in practice it still has some links to the specific security concept that it is part of. For instance, it has to organize the keys according to the security concept, know about the ciphers to be used for encryption of session keys, etc.

B. Main Requirements

The fundamental requirements of MKMF are derived from its role - generate and protect master and session keys. However, there are a number of other requirements in order to make the system usable:

1) Metadata and organization of keys - the keys can't be just stored in a single stack, they need to be separated by mission, AU, key type, etc. Metadata such as creation time, creator, export time is needed for tracing when solving potential incidents that may happen;
2) Access control - not all users should access all keys, also not all users should be able to perform all operations in the system, it is needed to have a control of this;
3) There has to be backup / restore function if it is needed to recover from a hardware failure such as hard disk corruption;
4) The system must be configurable (length of AES keys for instance) globally and for each mission;
5) A very clear and straightforward graphical user interface is needed for the users to perform the tasks smoothly.

In addition to those listed above, there are several other detail requirements which are not described here.

III. Major Challenges

The major challenges of MKMF are of course related to security and cryptography. The following sections will describe each of them as well as the principle solution.

A. Generation and Quality of AES Keys

In the context of symmetric cryptography the quality of the keys is defined by their randomness. Any pattern or correlation could lead to the prediction of the key(s). A true random number generator (TRNG) is a must for generating high quality AES keys.

The simplest ways of generating true random numbers are rolling dice or flipping coin, for instance. Unfortunately it takes too much time to produce 1024 keys each 128 bits long using this technique. A device named Quantis produced by a Swiss company IDQ was chosen as TRNG for MKMF. This device uses photons to generate random numbers. The light particles are sent to a semi-transparent mirror and reflection or transmission is associated to 0 or 1 correspondingly. The device is very easy to use (connects to a USB port) and has support for different operating systems and API-s for different programming languages (Java, C++). It can generate 4Mbits of random data in a second which is more than enough for satisfying the performance requirements of MKMF.

However, the random data generated by the device is not directly used to create 128, 192 or 256 bit long AES keys. Instead, the random data received from the device is used to seed a cryptographically strong pseudo random number generator (PRNG). In the current case the PRNG is an implementation of secure random of a JCE provider - either FIPSPRNG of IBMJCEFIPS (if available) or SHA1PRNG of the default SUN provider. One of the motivations to not directly use the output of the TRNG is to be protected against the potential failure of the device. For instance, if the device suddenly starts to generate only 0 or only 1, the actual keys are not entirely made of 0 or 1. Extra checks are applied to the random data generated at least once per session:

1) Using TRNG, 16 bytes of data is generated twice and results are compared, if results are equal the check fails;
2) Using PRNG seeded by TRNG, 16 bytes of data is generated twice and results are compared, if results are equal, the check fails.

Finally, every actual key generated is compared to last N > 0 (N is configurable) keys generated and if a match is found, this event is logged and a new key is generated. The following diagram illustrates the key generation process.

Figure 2 - Key Generation Process

In addition to the generation of master and session keys, the TRNG and PRNG combination is used also in several other cryptographic operations that are used to protect the system itself and its data. For instance, when generating the AES keys for CMS container encryption.

B. Protection of Master and Session Keys

While the metadata of the keys - such as the mission, the key group (AU) it belongs to, the creation time, etc. - and other system data is stored in a relational database (MySQL) the key values which are used for encryption must

not be stored in plain format anywhere. Therefore the key values in MKMF are stored in encrypted format as a regular file on local file system.

In short, the key values are encrypted using public-key (PK) cryptography and stored as a CMS container in a file. Using PK cryptography enables multi-user support and provides also a way to perform the authentication of users. As usually, each user of the system has a RSA key pair. The CMS container is encrypted using the public keys of the users who must have access to the keys. If a user wants to log on to the system, the attempt is made to decrypt the container. If it succeeds, the user is successfully authenticated.

Note that the data in the CMS container is not directly encrypted using public key of each user. Instead, a secret key is generated and the data is encrypted using AES. The secret key is then encrypted for each user using the corresponding public key. All this is part of CMS standard and supported by its implementations.

The key storage for each mission is physically separated because the permissions are always granted on mission basis. User who has access to the keys of Sentinel-1A may not have access to keys of Sentinel-2A. In addition to the encrypted keys, there are other details that need to be stored - all together it forms a mission key store which is implemented as a directory containing fixed set of files. The following diagram illustrates the mission key store.

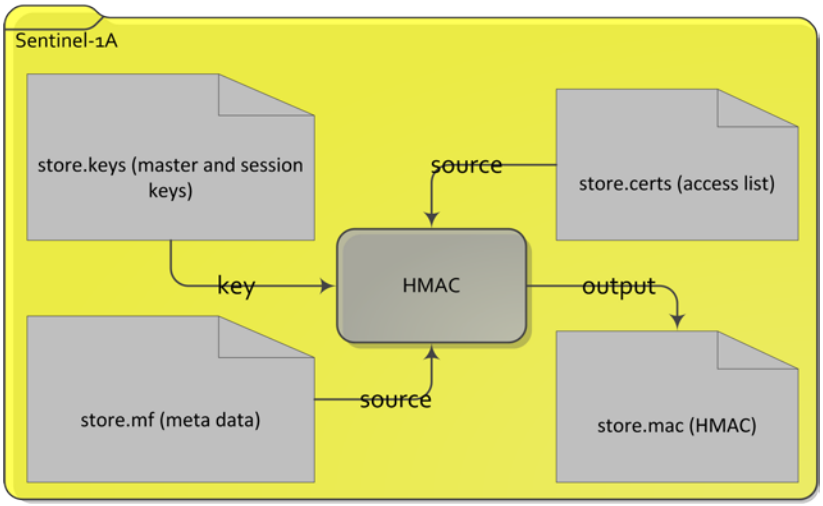

Figure 3 - Mission Key Store Files

The name of the folder normally matches the name of the mission as defined in the system. The store.keys file is a CMS container that is encrypted using all public keys stored in the file store.certs. The content of latter is the certificates of the users who have access to the key store. The certificate in this context is simply used as a well-known container for public key. The file store.mf stores metadata of the key store such as version, date last modified, etc.

The store.mac file contains the message authentication code (MAC) that is calculated using hash-based MAC (HMAC) algorithm. It uses the content of store.certs and store.mf as source and store.keys (as before encryption) as key. The MAC is needed to protect the key store against direct modifications. For instance, one could silently add a certificate to the store.certs file, wait until the store.keys is modified and re-encrypted (including now the added certificate) and then decrypt it. The MAC is re-calculated when the key store is opened and if it does not match the one stored in store.mac file an alarm is raised. Note that only the party knowing the content store.keys in plain can re-calculate the MAC and therefore modify the data in the key store files.

An alternative to software based encrypted file to protect the keys is to use a dedicated hardware security module (HSM) which offers additional physical protection. If one wants to access the keys stored in HSM by physically “opening” the device, the data will be automatically destroyed. In case of MKMF, the computer running the software is kept in special computer-safe that offers good physical protection - it is anchored to the floor, has 65mm thick double layer door made of 3mm steel, secure cable entry, etc. It is true that this safe will not automatically destroy the data on the hard disk when one wants to open it but the price of it is also less than 2% of the price of the HSM which costs approximately 38 000 euros. The other disadvantage of HSM is that it does not come with a suitable user interface (UI). The UI still has to be built and will then become specific to a certain piece of hardware. This limits the options to reuse the software for other missions (making it also quite expensive) and makes it less likely to find replacement hardware if something happens after 10 years of exploitation.

C. User Authentication and Access Control

The user authentication and access control is closely related to the protection of the key store. As discussed before, each user of the system has a RSA key pair. The private key of each user is stored in a passphrase-protected file (using PEM format) and in addition placed on an external secure USB memory stick (IronKey for instance). The USB stick is only connected when the system is being used by a user. It is up to the user to decide if to have a common USB stick for all users, one stick for each mission or even for each user.

The corresponding public keys are stored within X509 self-signed certificates in a file on a local file system. The certificates in the context of MKMF are only used as well-known containers for public keys that can store attributes of the corresponding user (first name, last name, username etc.) and to enforce the "change of password" using the validity dates of the certificate. The change of the password is a creation of a new RSA key pair for the user.

The authorization in MKMF is implemented using classical role-based access control. For each operation such as creating a master key, creating a session key etc. there is a privilege. Privileges can be combined into dynamic roles and roles are assigned to users. The roles are always assigned to user within a specific mission. One user may have different roles assigned in different missions. The authorization and role-based access control has no cryptographic strength - the related data is stored in the relational database and the check of the existence of a privilege is done in application source code. However, this is not an issue as this mechanism is not used to protect the sensitive information like master and session key values but only the data that is in the relational database. Also as described in the section D, it is not possible to make any changes to the system without leaving a trace in the audit log.

The connection between assigning role(s) to user and mission key store is that:

A) If a role is assigned to a user in a mission, the certificate (public key) of this user is added to the list of certificates of the mission key store and the key store is re-encrypted so that the user can decrypt it with his / her private key;

B) If all roles are removed, the certificate is removed from the mission key store and the key store is re-encrypted so that the user can't decrypt it anymore.

The following diagram illustrates the storage of the private keys and certificates.

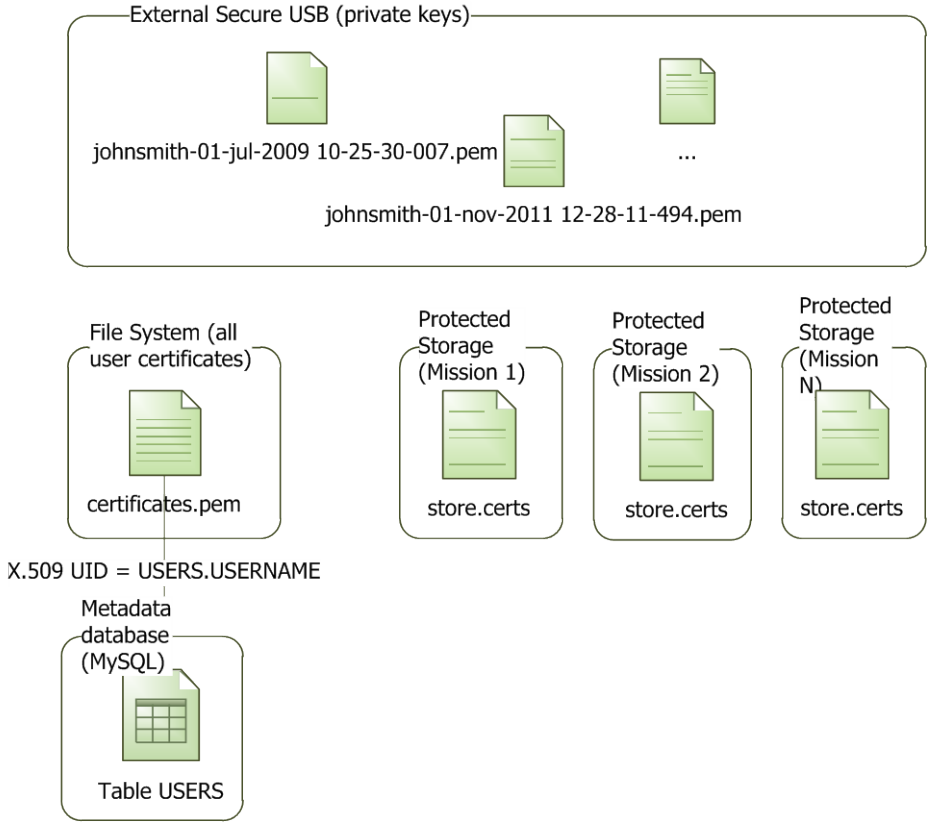

Figure 4 - Private Keys and Certificates

The logon procedure is as follows:

A) User provides location of his / her private key file and passphrase to decrypt it
B) System verifies the signature of the corresponding certificate using the given private key
C) The mission key store is decrypted using the given private key

If all the steps are successful the user can start to work with the mission.

The alternative to store private keys would have been the use of smartcards. The advantage of this is to be able to detect the theft of the private key in which case the physical card is missing. However there are some negative sides - it is difficult to increase the length of the keys and backup keys. In addition it involves specific extra hardware (cards and card readers) which makes the administration and maintenance of the system more complex, especially taking into account that the lifetime of the system is estimated to 25 years.

D. Verification of System Integrity

The system integrity and non-repudiation verification is used to ensure that the system data, especially the session and master keys, have not been changed by an unauthorized party directly modifying the system data; and without any permission to do so.

Reliable verification can only be done, if the "state" of the system is stored externally at regular intervals (for instance once a month). Later the externally stored "state" can be used as a reference when doing verification. It is possible that an unauthorized party can find the way to modify the system data; however, it is very unlikely that he or she can also change all the external copies of the "state" accordingly. The more external copies there are the better.

All important events in the system, such as generation of master keys, session keys, etc. are logged in audit log. For each audit log entry a unique hash (using SHA-512) is calculated using the content (message) of that entry and the hash of previous entry. The hash of the audit log entry can be taken as "state" of the system at that moment. The following table illustrates the audit log.

Message	**…**	**Key Hash**	**Hash**	**Checkpoint No**	**Checkpoint time**	**Checkpoint last verified**
…	…	…	…	…	…	…
User 'a' generated master key 123 in group 'Group1' for mission 'Sentinel 1-A'		ee1e60c7…	dc97775…	1	2011-11-10 15:34:25	2011-11-30 11:12:31
…	…	…	…	…	…	…
…	…	…	…	…	…	…
User 'b' generated master key 124 in group 'Group1' for mission 'Sentinel 1-A'		2e1f992c…	77fbf10…	2	2011-11-21 18:22:01	

Table 1 - Audit Log Sample

The hash of each entry can be always recalculated staring from the first message. Modifying the audit log by changing the message or key hash, adding new entries or deleting entries will always result in a different hash value. When master or session keys are generated, the key value (or actually the hash of it) will be included when calculating the hash of the corresponding audit log entry.

When hash of the (last) entry is stored externally (printed on paper or stored on secure memory flash drive) it can be used any time later as reference in order to check if the audit log has been modified or master / session key values have been modified. The verification process consists of two activities:

A) Setting of checkpoints in audit log and printing hash values on paper (or storing externally otherwise);
B) Verifying the audit log by recalculating the hash values of audit log entries and comparing them to the values printed on paper.

It is up to the user how often to set checkpoints and how often to do verification. It greatly depends on the risk of an unauthorized attempt to change system data.

In addition to the verification of the entire system, it is also possible to check the integrity of a single key or a set of keys. When key is generated a hash is calculated from the key value. The hash of the key value is stored in 3 different locations:

A) Audit log;
B) Relational database (with other meta data of the key);
C) Protected mission key store (together with key value).

In addition the hash can be at any time re-calculated using the current key value. When integrity of the key is checked, these three copies of hash are compared to each other and to the hash calculated from the key value during the check. Normally all the different copies should match. If there are differences it means that someone has manually modified either the stored hashes and/or the key value.

IV. Benefits to the End User

A. Simplicity

The system is very simple to use already from the installation. The installation of the application and bootstrapping (creation of initial mission and user) hardly takes more than half an hour. All functions are available via nice native graphical user interface. The graphical UI follows the design of typical Eclipse RCP application - context menus and wizards make the performing of the operations simple and clear. The user always works on one mission at a time. The application has two perspectives - one for managing keys and one for administration. The following screenshot gives an overview of the application's main window.

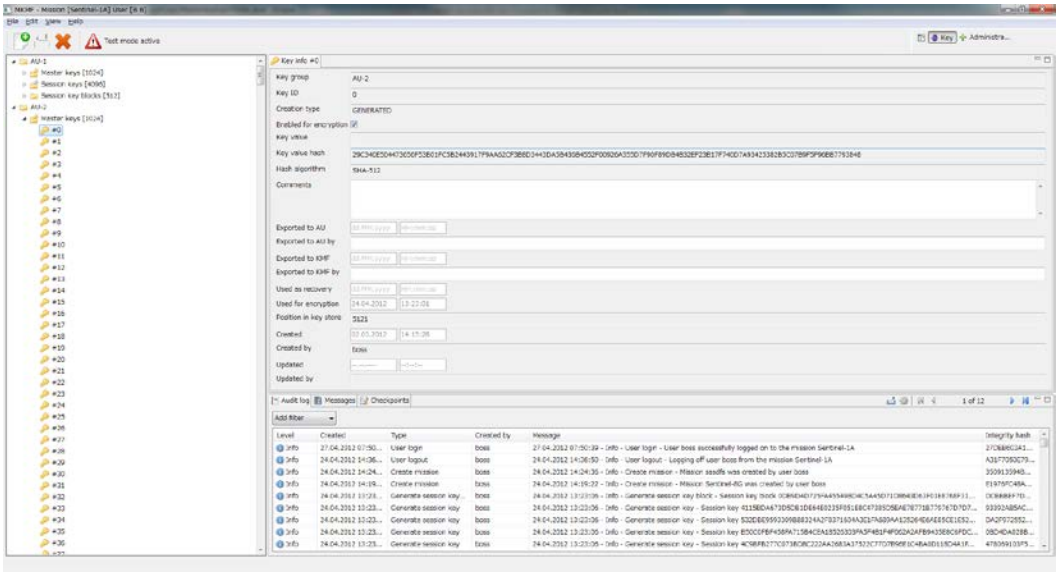

Figure 5 - MKMF Main Window

In addition to the core functions of creating master keys and session keys the following functions are available:

A) Create different key groups depending on the goal of the keys - flight keys for different AU-s, test keys, etc.;
B) Export master keys for AU and session keys for KMF;
C) Set and verify checkpoints in audit log to ensure system integrity;
D) Check integrity of individual key(s);
E) Create and restore backups - either globally or of specific mission;
F) Filter audit log entries in order to review activities and solve potential incidents;
G) Export random data produced by key generator to a file in order to evaluate the quality of it using NIST Test Suite;
H) Dynamic role based access control with multi-mission support.

B. Low-cost

The system has a good balance of cost, quality and security. It is based on a regular PC platform and can run on any hardware supporting Microsoft Windows, Linux or Mac OS operating systems. The only specific hardware pieces required are the Quantis TRNG device and IronKey secure USB memory stick. Also, all software products and libraries used are open source and do not require any licenses to be bought.

C. Security

The system combines and makes use of well-known existing security standards, algorithms, methods and tools in the right way to protect the sensitive information and ensure that the system is not compromised. The hardware used

is cost-effective and allows choosing between alternatives. For instance, one can select a suitable computer-safe without doing any changes to the system. Using a different TRNG device is only matter of changing a few lines of code and can even easily be made configurable by end user.

D. Flexibility

The system has been designed and implemented in the way that it supports easy customization according to the needs of a specific mission. Several things like length of the keys, number keys per key group are configurable per mission basis by end user. In addition the system is easily extendable. For instance, the limitations and details related to export to a certain AU unit or KMF are not hardcoded into the entire application but only to the specific export functionality. Adding a new export format of master or keys is just matter of implementing this export and does not require changes to the entire system.

V. Conclusion

The Sentinels Master Key Management Facility is a lightweight and cost-effective but secure tool to generate and store AES keys. No new security principles, encryption algorithms or ciphers were invented to accomplish the facility. Well known means such as symmetric cryptography, public key cryptography, hash algorithms, hash chains, message authentication code, etc. are used according to best practices in order to secure the system. Existing software frameworks such as JCE and implementations are used to perform cryptographic operations such as AES encryption, creation of CMS containers, calculating hashes, generating cryptographically strong random data. All this has made the MKMF a high quality product. Even currently built specifically for the GMES Sentinels, the system can be used for any other mission that is using symmetric cryptography to secure the communication between ground and space segment. No change is required to use MKMF in the context where in addition to TC authentication, TC and TM data encryption or any other operation is done using the master / session keys.

Appendix A
Acronym List

AES	Advanced Encryption Standard
API	Application Programming Interface
AU	Authentication Unit
CMS	Cryptographic Message Syntax
ESA	European Space Agency
ESOC	European Space Operations Centre
FIPS	Federal Information Processing Standards
GMES	Global Monitoring for Environment and Security
HMAC	Hash-based MAC
JCE	Java Cryptography Extension
JRE	Java Runtime Environment
KMF	Key Management Facility
LAC	Logical Authentication Counter
MAC	Message Authentication Code
MCS	Mission Control System
MKMF	Master Key Management Facility
NIST	National Institute of Standards and Technology
PK	Public-key Cryptography
PRNG	Pseudo Random Number Generator
RAM	Random-Access Memory
PROM	Programmable Read-Only Memory
TC	Telecommand
TRNG	True Random Number Generator

UI	User Interface
USB	Universal Serial Bus

Appendix B
Glossary

IBMJCEFIPS	The IBM Java JCE FIPS Provider for multi-platforms is a scalable, multi-purpose cryptographic module that supports FIPS-approved cryptographic operations through Java APIs.
FIPSPRNG	The secure random implementation in the IBMJCEFIPS provider
RSA	An algorithm for public-key cryptography that is based on the presumed difficulty of factoring large integers, the factoring problem
SUN	The default JCE provider which is always available in JRE
SHA1PRNG	The secure random implementation in the SUN provider
SHA-512	Cryptographic hash function designed by the National Security Agency and published in 2001 by the NIST as a U.S. Federal Information Processing Standard
NIST Test Suite	A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications issued by NIST
X509	X.509 is an ITU-T standard for a public key infrastructure (PKI) and Privilege Management Infrastructure (PMI). X.509 specifies, amongst other things, standard formats for public key certificates, certificate revocation lists, attribute certificates, and a certification path validation algorithm