Ethemba Trusted Host Environment Mainly Based on Attestation

a framework and demonstrator for TPM applications

Andreas Brett
andreas_brett@ethemba.info

Andreas Leicher
andreas_leicher@ethemba.info

December 24, 2008
This work was supported by
Dr. Andreas U. Schmidt (CREATE-NET)
Nicolai Kuntze (Fraunhofer-SIT)
## Contents

1 apps
   1.1 ClearOwnership .......................... 8
      1.1.1 Description .......................... 8
      1.1.2 PUBLIC main .......................... 8
   1.2 TakeOwnership ........................... 9
      1.2.1 Description .......................... 9
      1.2.2 PUBLIC main .......................... 9
   1.3 ManageKnownHashesList .................... 10
      1.3.1 Description .......................... 10
      1.3.2 PUBLIC main .......................... 10
   1.4 PCAclient ................................ 12
   1.5 PCAserver ................................ 14
   1.6 RAclient ................................ 15
   1.7 RAserver ................................ 17

2 jTPMtools .................................. 19
   2.1 AikUtil .................................. 20
   2.2 Client ................................... 21
   2.3 PrivacyCa ................................ 22

3 modules .................................... 23
   3.1 client.CertDB ............................ 24
      3.1.1 Description .......................... 24
      3.1.2 PUBLIC getCert ....................... 24
      3.1.3 PUBLIC putCert ....................... 24
      3.1.4 PUBLIC removeCert .................... 24
      3.1.5 PUBLIC exportCert .................... 24
   3.2 client.CertifyKey ......................... 25
      3.2.1 Description .......................... 25
      3.2.2 CONSTRUCTOR CertifyKey ............... 25
      3.2.3 PUBLIC run .......................... 25
   3.3 client.CreateKey .......................... 28
      3.3.1 Description .......................... 28
      3.3.2 CONSTRUCTOR CreateKey ............... 28
      3.3.3 PUBLIC run .......................... 28
   3.4 client.DataBinding ....................... 29
| Section                                      | Lines |
|----------------------------------------------|-------|
| 3.4.1 Description                            | 29    |
| 3.4.2 **CONSTRUCTOR** DataBinding            | 29    |
| 3.4.3 **PUBLIC** run                         | 29    |
| 3.5 client.DataUnbinding                     | 30    |
| 3.5.1 Description                            | 30    |
| 3.5.2 **CONSTRUCTOR** DataUnbinding          | 30    |
| 3.5.3 **PUBLIC** run                         | 30    |
| 3.6 client.QuoteRetrieval                    | 31    |
| 3.6.1 Description                            | 31    |
| 3.6.2 **CONSTRUCTOR** QuoteRetrieval         | 31    |
| 3.6.3 **PUBLIC** run                         | 31    |
| 3.7 client.TpmKeyDB                          | 32    |
| 3.7.1 Description                            | 32    |
| 3.7.2 **PUBLIC** getUUID                     | 32    |
| 3.7.3 **PUBLIC** putUUID                     | 32    |
| 3.7.4 **PUBLIC** removeUUID                  | 32    |
| 3.8 server.CertifyKeyValidation              | 34    |
| 3.8.1 Description                            | 34    |
| 3.8.2 **CONSTRUCTOR** CertifyKeyValidation   | 34    |
| 3.8.4 **PUBLIC** run                         | 34    |
| 3.9 server.ExternalDataBinding               | 36    |
| 3.9.1 Description                            | 36    |
| 3.9.2 **CONSTRUCTOR** ExternalDataBinding    | 36    |
| 3.9.3 **PUBLIC** run                         | 36    |
| 3.10 server.KeyStorage                       | 37    |
| 3.10.1 Description                           | 37    |
| 3.10.5 **PUBLIC** putPublicKeyFile           | 38    |
| 3.10.6 **PUBLIC** putPrivateKeyFile          | 38    |
| 3.10.7 **PUBLIC** putPublicKey               | 38    |
| 3.10.8 **PUBLIC** putPrivateKey              | 38    |
| 3.10.9 **PUBLIC** put                        | 39    |
| 3.10.10 **PUBLIC** putPublicKey              | 39    |
| 3.10.11 **PUBLIC** putPrivateKey             | 40    |
| 3.11 server.QuoteValidation                  | 41    |
| 3.11.1 Description                           | 41    |
| 3.11.2 **CONSTRUCTOR** QuoteValidation       | 41    |
| 3.11.5 **PUBLIC** run                        | 42    |
| 4 net                                        | 43    |
| 4.1 NetEntity                                | 44    |
| 4.1.1 Description                            | 44    |
| 4.1.2 **CONSTRUCTOR** NetEntity              | 44    |
| 4.1.3 **PUBLIC** init                        | 44    |
| Section | Description | Page |
|---------|-------------|------|
| 4.1.4   | PUBLIC close | 44   |
| 4.1.5   | PUBLIC sendACK | 44   |
| 4.1.6   | PUBLIC sendNACK | 45   |
| 4.1.7   | PUBLIC sendCommand | 45   |
| 4.1.8   | PUBLIC sendObject | 45   |
| 4.1.9   | PUBLIC receiveACK | 46   |
| 4.1.10  | PUBLIC receiveCommand | 46   |
| 4.1.11  | PUBLIC receiveObject | 46   |
| 4.1.12  | PUBLIC getRemoteIP | 47   |
| 4.1.13  | PUBLIC getRemoteHostname | 47   |
| 4.2     | NetCommand | 48   |
| 4.2.1   | Description | 48   |
| 5       | types      | 49   |
| 5.1     | MeasurementList | 50   |
| 5.1.1   | Description | 50   |
| 5.1.2   | CONSTRUCTOR MeasurementList | 50   |
| 5.1.3   | PUBLIC getMeasurementList | 50   |
| 5.1.4   | PUBLIC getMeasurementListForPCR | 50   |
| 5.1.5   | PUBLIC loadFromFile | 51   |
| 5.2     | KnownHashesList | 52   |
| 5.2.1   | Description | 52   |
| 5.2.2   | CONSTRUCTOR KnownHashesList | 52   |
| 5.2.3   | PUBLIC put | 52   |
| 5.2.4   | PUBLIC get | 52   |
| 5.2.5   | PUBLIC containsTag | 53   |
| 5.2.6   | PUBLIC containsSha1Hash | 53   |
| 5.2.7   | PUBLIC contains | 53   |
| 5.2.8   | PUBLIC saveToFile | 55   |
| 5.2.9   | PUBLIC loadFromFile | 55   |
| 6       | utils      | 55   |
| 6.1     | AES         | 56   |
| 6.1.1   | Description | 56   |
| 6.1.2   | CONSTRUCTOR AES | 56   |
| 6.1.3   | PUBLIC encryptObject | 56   |
| 6.1.4   | PUBLIC encrypt | 56   |
| 6.1.5   | PUBLIC decryptObject | 57   |
| 6.1.6   | PUBLIC decrypt | 57   |
| 6.1.8   | PUBLIC generateKey | 58   |
| 6.1.9   | PUBLIC generateKey | 58   |
| 6.1.10  | PUBLIC generateIV | 58   |
| 6.2     | Helpers    | 59   |
1 apps

DE.FRAUNHOFER.SIT.TC.ETHEMBA.APPS

This package contains applications for TPM maintenance on the one hand and client- and server-applications that implement AIK-Certification and Remote-Attestation on the other hand.

- `TakeOwnership` and `ClearOwnership` permit TPM activation capabilities
- `ManageKnownHashesList` helps managing hash lists, that define trusted applications used in Remote-Attestation on the server-side
- `PCAServer` and `PCAClient` implement AIK-Certification
- `RAServer` and `RAclient` implement Remote-Attestation
1.1 ClearOwnership

1.1.1 Description

ClearOwnership clears the ownership of the TPM. The code is mainly based on the original jTpmtools (see References) application. It can be used in the process of resetting the TPM to its initial state. A reboot might be required for changes to take effect. The current owner password is supplied as command-line parameter.

For convenience and tests, the switch /f provides a fixed mode, reading the globally set owner password.

Note: In our test scenario, routing the TPM-Emulator into a virtualized sub-system, we were able to clear the ownership without supplying the correct owner password. It is still to be clarified, if this is an issue with the emulated environment or the implementation in jTSS.

1.1.2 PUBLIC main

Description

Called when used in command-line.

Settings

OwnerPwd

Parameters

String[] args  [1] owner password or globally set owner password when used in fixed mode ('/f').

Output

If no command line parameters are given, a usage information is displayed.
1.2 TakeOwnership

1.2.1 Description
TakeOwnership allows taking ownership of the TPM. The owner password is set, a new SRK is generated inside the TPM and the SRK password is set. The code is mainly based on the original jTpmtools (see References) application.

The owner and SRK passwords are supplied as command-line parameters. For convenience and tests, the switch ‘/f’ provides a fixed mode, reading the globally set owner and SRK passwords.

1.2.2 PUBLIC main

Description
Called when used in command-line.

Settings
OwnerPwd, SRKPwd

Parameters

| STRING[] args | [1] owner password or ’/f’ for fixed mode |
|              | [2] srk password |

Output

If no command line parameters are given, a usage information is displayed.
1.3 ManageKnownHashesList

1.3.1 Description

ManageKnownHashesList takes an IMA-formatted file as input and converts it to a Known-HashesList. This application can be used to create and maintain a database of known hashes. Used in append-mode (command line parameter /a), the contents of the input will be appended to the database. Used in overwrite-mode (command line parameter /o), the contents of the input will overwrite an existing database. If no command line parameter is given, a management console is presented to the user, allowing for view, search and remove entries of the database.

1.3.2 PUBLIC main

Description
Called when used in command-line.

Settings
RAServer_KnownHashesList

Parameters

| STRING[] args | [1] /a or /o to enable append- or overwrite-mode (optional) | [2] filename of IMA-Measurement-File to be used when [1] is set |

Output
If no command line parameters are given, a management console providing view, search and remove functions is presented to the user. Otherwise the new or updated database of known hashes is stored in RAServer_KnownHashesList.

1.3.3 PRIVATE append

Description
appends contents of given IMA-Measurement-File to existing database.

Settings
RAServer_KnownHashesList

Parameters

| STRING file | IMA-Measurement-File |

1.3.4 PRIVATE overwrite

Description
overwrites existing database with contents of given IMA-Measurement-File.
1.3 ManageKnownHashesList

**Settings**
RAServer_KnownHashesList

**Parameters**

| STRING file | IMA-Measurement-File |

1.3.5 **PRIVATE** view

**Description**

displays the entries contained in the KnownHashesList pagewise (blocks of 10 entries)

**Settings**

none

**Parameters**



1.3.6 **PRIVATE** search

**Description**

displays the entries contained in the KnownHashesList matching the given search string

**Settings**

none

**Parameters**

| STRING s | search string to be used |

1.3.7 **PRIVATE** remove

**Description**

removes entries contained in the KnownHashesList matching the given search string

**Settings**

none

**Parameters**

| STRING s | search string to be used |
1.4 PCAsclient

The PCAsclient can be used to create and certify an AIK. The client creates a new AIK using the TPM_CollateIdentityRequest. The public part of it is sent to a PCAserver including the Endorsement Certificate. They are encrypted using the PCAserver’s public key. Upon receipt, the PCAserver can decrypt the contents and verify the Endorsement Certificate. The PCAserver then creates a random nonce and wraps it for the client. The response contains two parts:

1. a symmetric session key and the hash of the AIK public key, both encrypted with the EK public key

2. the nonce, encrypted with the session key

for details see PrivacyCa state3_sub.

Using the TPM_ActivateIdentity command, the TPM can decrypt the answer and the nonce is revealed to the client. Only the client with the TPM used for creation of the AIK is able to decrypt the nonce. The client then sends the decrypted nonce back to the PCAserver.

Upon receipt and verification, the PCAserver will create a certificate for the AIK. It is then encrypted using an AES key. The AES key is wrapped for the client using the same scheme as for the nonce, thus the response consists of three parts:

1. a symmetric session key and the hash of the AIK public key, both encrypted with the EK public key

2. the AES key encrypted with the session key

3. the AIK-Certificate, encrypted with the AES key

for details see PrivacyCa state5_sub.

The AIK is activated by the client via a TPM_ActivateIdentity call. The AES key is decrypted and can be used to decrypt the AIK-Certificate.

The client then assigns a UUID to the AIK and stores it in the System Persistent Storage of jTSS. For later access, the key is registered in the client.TpmKeyDB using the provided AIK label. As final step the AIK-Certificate is stored in the client.CertDB.

The client is separated into different methods representing each state of the protocol. The methods are called subsequently.

**Parameters**

| Parameter               | Description                                           |
|-------------------------|-------------------------------------------------------|
| owner password          | needed to load the EK certificate / EK public key     |
| SRK password            | needed to store the new AIK                           |
| AIK password            | will be needed when the AIK is accessed later on      |
| AIK label               | a label providing identification of the AIK. This label can later be used to load the AIK from client.TpmKeyDB and will also be included in the certificate. |
| PCA server IP address   | IP address of the PCA server                           |
| PCA server port         | port of the PCA server, see Settings for default value |

All required parameters can be set from the command-line, a fixed mode is provided via the switch '/f'. The required parameters will then be read from Settings.
Figure 1.1: protocol diagram for AIK-Certification
1.5 PCAserver

The server is separated into different methods representing each state of the AIK-Certification protocol. The methods are called subsequently.

→ see PCAclient for a detailed description of the AIK-Certification protocol
1.6 RAclient

The RAclient can be used to perform a Remote-Attestation. Integrated Measurement Architecture (IMA) is used to provide runtime analysis of the client’s system state. PCR-10 is constantly being extended with hashes of newly run programs. A client is being attested by a Remote-Attestation-Server by sending an AIK-signed quote of PCR-10 altogether with the corresponding AIK-Certificate and a MeasurementList to the server.

The quote is protected from replay-attacks by including a nonce, the Remote-Attestation-Server challenged the client with. By letting the TPM quote PCR-10 internally, integrity of PCR-10 is ensured.

Upon receipt, the Remote-Attestation-Server performs several checks to attest the client’s system state. First of all the AIK-Certificate is checked for validity (timestamp, signature and trusted issuer). Then the submitted MeasurementList is validated by checking each run application for known and trusted hashes. This results in a re-calculated PCR-10. At last the submitted quote is being validated by checking its signature, the included nonce (for anti-replay!) and equality of the included PCR-10 and the re-calculated one.

This equality check is highly required to ensure an attacker did not trick the Remote-Attestation-Server by submitting a fake MeasurementList.

After verification the client will receive a certificate from RAserver certifying platform integrity. Note that this certificate has to be invalid after a short period of time, as the client’s state might change very quickly.

The RAclient stores the Attestation-Certificate in the client.CertDB using a new UUID. The UUID for the Attestation-Certificate is based on the UUID of the AIK used in the protocol. It can be accessed for later use via the UUID 00000009-0008-0007-0605-aikUuid.getNode().

The client is separated into different methods representing each state of the protocol. The methods are called subsequently.

**Parameters**

| Parameter       | Description                                      |
|-----------------|--------------------------------------------------|
| SRK password    | needed to load the AIK                          |
| AIK password    | needed to load the AIK                          |
| AIK label       | the label the to be used AIK is stored under     |
| RA server IP address | IP address of the RA server                   |
| RA server port  | port of the RA server, see Settings for default value |

All required parameters can be set from the command-line, a fixed mode is provided via the switch ’/f’. The required parameters will then be read from Settings.
Figure 1.2: protocol diagram for Remote Attestation
1.7 RAserver

The server is separated into different methods representing each state of the Remote-Attestation protocol. The methods are called subsequently.

→ see RAclient for a detailed description of the Remote-Attestation protocol
2 jTPMtools

DE.FRAUNHOFER.SIT.TC.ETHEMBA.JTPMTOOLS

This package contains modified source-code taken from jTPMtools.
2.1 AikUtil

This class was copied and modified from IAIK.TC.APPS.JTT.AIKAIkUtil.

It is used within PCAclient to create EK-Certificates on-the-fly (which are passed to the PCAserver). Additionally it is indirectly used within PCAserver and directly used within PrivacyCa to create AIK-Certificates (which are passed to the PCAclient).

Modifications applied:

• changed AIK-Certificate creation to use ethemba's global settings for certificate attributes
• changed CA-Certificate creation to use ethemba's global settings for certificate attributes
• changed EK-Certificate creation to use ethemba's global settings for certificate attributes
• changed PE-Certificate creation to use ethemba's global settings for certificate attributes
2.2 Client

This class was copied and slightly modified from IAIK.TC.APPS.JTT.AIK.CLIENT.

It is used within PCAclient in state3 to run a TPM_CollateIdentityRequest command and in state4 and state6 to run a TPM_ActivateIdentity command.

Modifications applied:

• changed visibility of activateIdentity from protected to public to have outer access
• changed visibility of collateIdentityReq from protected to public to have outer access
• changed visibility of overrideEkCertificate from protected to public to have outer access
2.3 PrivacyCa

This class was copied and modified from IAIK.TC.APPS.JTT.AIK.PRIVACYCA.

It is used within PCAserver to process the collateIdentity request blob received from the PCAclient and wrapping a server-generated nonce inside a TPM blob that can only be accessed by the TPM itself.

In another state of the AIK-Certification protocol (see PCAclient) the second collateIdentity request blob received from the PCAclient is being processed and the resulting AIK-Certificate is again wrapped inside a TPM blob that can only be accessed by the TPM itself.

See PCAclient for detailed information on the AIK-Certification protocol.

**Modifications applied:** Two new methods were added to implement a handshake in the AIK-Certification protocol. The state3_sub method provides the wrapping of the server-generated nonce for the client instead of transmitting the certificate in the first step. In state3_sub the EK certificate is already validated.

In this design, clients without a valid EK certificate will be rejected at an early stage in the protocol. A later verification of the EK certificate might be advantageous as it decreases the burden on the PCA in case the protocol stops during nonce verification. The AIK-Certificate is not issued in this stage.

The method state5_sub provides a secure wrapping for the AIK-Certificate. First the AIK-Certificate is generated. A new one-time symmetric AES key is being generated and used to encrypt the certificate. The response consists of three parts:

1. a symmetric session key and the hash of the AIK public key, both encrypted asymmetrically with the public EK of the client. Via this indirection, only the client is able to decrypt the session key.

2. the symmetric AES key previously used to encrypt the AIK-Certificate, encrypted with the session key

3. the AIK-Certificate, encrypted with the AES key

The client can then decrypt the session key using the TPM.ActivateIdentity function of the TPM. This reveals the AES key, which can then in turn be used to decrypt the AIK-Certificate.
3 modules

DE.FRAUNHOFER.SIT.TC.ETHEMBA_MODULES

This package contains modules providing storage/mapping methods for keys and certificates on the one hand and modules providing convenient TPM method handling (e.g. quote, bind, certify etc.) on the other hand.
3.1 client.CertDB

3.1.1 Description
CertDB saves X509-Certificates in a database mapping it to given UUIDs.

3.1.2 PUBLIC getCert
Description
returns the corresponding certificate for the given UUID

Settings
CertDBfile
Parameters
| TC| TSS| UUID | uuid | UUID of the certificate |

Output
X509CERTIFICATE corresponding certificate

3.1.3 PUBLIC putCert
Description
puts a given certificate into the CertDB using the given UUID

Settings
CertDBfile
Parameters
| TC| TSS| UUID | uuid | UUID of the certificate |
| X509CERTIFICATE | cert | certificate to be put into the CertDB |

3.1.4 PUBLIC removeCert
Description
removes a certificate from the CertDB

Settings
CertDBfile
Parameters
| TC| TSS| UUID | uuid | UUID of the certificate |

3.1.5 PUBLIC exportCert
Description
exports a certificate to the disk
Settings
CertDBfile

**Parameters**

| TCGSSUID | uuid   | UUID of the certificate |
|---------|--------|-------------------------|
| STRING  | outputFile | location of the exported certificate |

### 3.1.6 PRIVATE loadDB

**Description**
loads the CertDB from the globally defined location

**Settings**
CertDBfile

**Parameters**

| none |

**Output**
HASHTABLE<STRING, BYTE[]>| loaded database |

### 3.1.7 PRIVATE saveDB

**Description**
saves a given CertDB to the globally defined location

**Settings**
CertDBfile

**Parameters**

| HASHTABLE<STRING, BYTE[]> | db | database to be saved |
|---------------------------|----|-----------------------|


3.2 client.CertifyKey

3.2.1 Description

This client module can be used to create and certify a TPM key for binding or sealing. The created key is signed by a given AIK.

![Diagram of key creation process]

Figure 3.1: Creation of a CSK (Certified Signing Key) using a certified AIK

3.2.2 CONSTRUCTOR CertifyKey

Description
takes the given input values and stores them internally for later passing it to the TPM via the run method

Settings

| Parameters       | Description                                      |
|------------------|--------------------------------------------------|
| BOOLEAN isBindingKey | create a binding key (TRUE) or sealing key (FALSE) |
| BOOLEAN isVolatile   | volatile or non-volatile key                      |
| STRING srkPwd | SRK-Password                                       |
| STRING keyPwd | Key-Password                                       |
| STRING keyLabel | Label to store created key under                 |
| INT[] pcrSelection | array to define, to which PCRs the key shall be bound to (optional) |
| BYTE[] nonce | nonce to be included in certification (anti-replay!) |
| STRING aikPw | AIK-Password                                      |
| STRING aikLabel | Label of AIK to be used for signing               |

3.2.3 PUBLIC run

Description
Implements the necessary steps to create and certify a TPM key using the TPM. First the SRK (to store the key under) and AIK (to sign the key) are loaded and key attributes for the newly
created key are being assigned. Then a composite object is created holding the selected PCRs
given via the constructor. This data is passed to the TPM using the certifyKey method provided
by jTSS (see References). After creating/certifying the key, it is stored in the persistent storage
and labeled with the given keyLabel. An OBJECT[] is returned containing everything needed
in server.CertifyKeyValidation:

- TcBlobData public key
- TcTssValidation keyCertification
- X509Certificate AIK-Certificate

**Settings**

pwdEncoding, TPM_KeySize

**Parameters**

| none |

**Output**

OBJECT[] containing [1] TcBlobData public key, [2] TcTssValidation keyCertification, [3]
X509Certificate AIK-Certificate
3.3 client.CreateKey

3.3.1 Description
This client module can be used to create a TPM key for binding or sealing.

3.3.2 CONSTRUCTOR CreateKey

Description
takes the given input values and stores them internally for later passing it to the TPM via the run method

Settings
none

Parameters

| Parameter            | Description                                      |
|----------------------|--------------------------------------------------|
| BOOLEAN isBindingKey | create a binding key (TRUE ) or sealing key (FALSE ) |
| BOOLEAN isVolatile   | volatile or non-volatile key                     |
| BOOLEAN isMigratable | migratable or non-migratable key                 |
| STRING srkPwd        | SRK-Password                                     |
| STRING keyPwd        | Key-Password                                     |
| STRING keyLabel      | Label to store created key under                 |
| INT[] pcrSelection   | array to define, to which PCRs the key shall be bound to (optional) |

3.3.3 PUBLIC run

Description
Implements the necessary steps to create a TPM key using the TPM. First the SRK (to store the key under) is loaded and key attributes for the newly created key are being assigned. Then a composite object is created holding the selected PCRs given via the constructor. This data is passed to the TPM using the createKey method provided by jTSS (see References). After creating the key, it is stored in the persistent storage and labeled with the given keyLabel.

Settings
pwdEncoding, TPM_KeySize

Parameters
none

Output
TcBLOBData public part of created keypair
3.4 client.DataBinding

3.4.1 Description
This client module can be used to bind (i.e. encrypt) data to the TPM. The bound data can only be unbound (i.e. decrypted) inside this TPM.

3.4.2 CONSTRUCTOR DataBinding

Description
takes the given input values and stores them internally for later processing in the run method

Settings
none

Parameters
| Type       | Description                      |
|------------|----------------------------------|
| BYTE[] data| data to be bound/encrypted       |
| STRING srkPwd | SRK-Password                   |
| STRING keyPwd | Key-Password                   |
| STRING keyLabel | Label of the key to be used to bind the data |

3.4.3 PUBLIC run

Description
Implements the necessary steps to bind the given data using the public key belonging to the given keyLabel. First the key is loaded using the SRK. Then data is separated into chunks matching the keysize and passed to the bind method provided by jTSS (see References).

Settings
pwdEncoding, TPM_KeySize

Parameters
none

Output
BYTE[] bound/encrypted data
3.5 client.DataUnbinding

3.5.1 Description

This client module can be used to unbind (i.e. decrypt) once bound data.

3.5.2 CONSTRUCTOR DataUnbinding

Description
takes the given input values and stores them internally for later processing in the run method.

Settings

| Parameters       | Description                                      |
|------------------|--------------------------------------------------|
| BYTE[] data      | data to be bound/encrypted                       |
| STRING srkPwd    | SRK-Password                                     |
| STRING keyPwd    | Key-Password                                     |
| STRING keyLabel  | Label of the key to be used to unbind the data   |

3.5.3 PUBLIC run

Description

Implements the necessary steps to unbind the given data using the public key belonging to the given keyLabel. First the key is loaded using the SRK. Then data is separated into chunks matching the keysize and passed to the unbind method provided by jTSS (see References).

Settings

pwdEncoding

| Parameters       | Description                                      |
|------------------|--------------------------------------------------|
| none             |                                                  |

Output

| BYTE[] unbound/decrypted data |
3.6 client.QuoteRetrieval

3.6.1 Description

This client module can be used to request and retrieve a quote from the TPM. The quote includes the signed hash value of the desired PCRs and a signed nonce, as replay protection. The client must provide a valid certificate for the key used to sign the quote. Normally an AIK is used to sign the quote, so a previously acquired AIK certificate from \texttt{PCAclient}/\texttt{PCAserver} can be used to verify the quote.

3.6.2 \textbf{CONSTRUCTOR} QuoteRetrieval

Description
takes the given input values and stores them internally for later retrieval of the quote via the run method

Settings

\textbf{Parameters}

| Parameter   | Description                                      |
|-------------|--------------------------------------------------|
| \texttt{INT[\[] pcrSelection} | array to define which PCRs shall be included in the quote |
| \texttt{BYTE[\[] nonce}     | nonce to be included in quote (→ replay protection) |
| \texttt{STRING srkPwd}      | SRK password, needed to load the AIK             |
| \texttt{STRING aikPwd}      | AIK password, needed to load the AIK             |
| \texttt{STRING aikLabel}    | label that has been assigned to the AIK in a PCA process |

3.6.3 \textbf{PUBLIC} run

Description
Implements the necessary steps to retrieve the quote from the TPM. First the AIK is loaded using the SRK. Then a composite object is created holding the selected PCRs given via the constructor. This data altogether with the nonce is passed to the TPM using the quote method provided by jTSS (see \texttt{References}). After retrieving the quote, the AIK is unloaded from the TPM and the quote is returned to the calling method.

Settings

\texttt{pwdEncoding}

\textbf{Parameters}

| Parameter   |
|-------------|
| none        |

Output

\texttt{TcTSSValidation} the quote that was retrieved from the TPM
3.7 client.TpmKeyDB

3.7.1 Description

TpmKeyDB maps labels to UUIDs. This is used to allow convenient access to keys generated by the TPM. Those keys are separated by TPM generated UUIDs which are rather difficult to remember. TpmKeyDB allows to assign labels to those UUIDs and to access them in an easy way.

3.7.2 PUBLIC getUUID

Description
returns the corresponding UUID for the given label

Settings
TpmKeyDBfile

Parameters

| STRING label | label to get the UUID for |

Output
TcTssUUID corresponding UUID for the given label

3.7.3 PUBLIC putUUID

Description
puts a label ⇔ UUID mapping to the TpmKeyDB

Settings
TpmKeyDBfile

Parameters

| STRING label | label to be used |
| TcTssUUID uuid | UUID to be used |

3.7.4 PUBLIC removeUUID

Description
removes a label ⇔ UUID mapping from the TpmKeyDB

Settings
TpmKeyDBfile

Parameters

| STRING label | label whose mapping is to be removed |
3.7.5 PRIVATE loadDB

**Description**
loads the TpmKeyDB from the globally defined location

**Settings**
TpmKeyDBfile

**Parameters**
none

**Output**
HASHTABLE<STRING, STRING> loaded database

3.7.6 PRIVATE saveDB

**Description**
saves a given TpmKeyDB to the globally defined location

**Settings**
TpmKeyDBfile

**Parameters**

| Parameters       | Description                  |
|------------------|------------------------------|
| HASHTABLE<STRING, STRING> db | database to be saved |
3.8 server.CertifyKeyValidate

3.8.1 Description

This server module can be used to validate a given output of a \texttt{client.CertifyKey} command. The included keyCertification is equipped with a signature and contains a nonce (→ replay protection). First the signature is checked for validity, then the nonce is matched against the challenged one (see \texttt{client.CertifyKey}). At last the digests of the included public key and the one contained in the output of \texttt{client.CertifyKey} are checked for identicalness.

\textbf{Note:} CertifyKeyValidation only checks the result of a \texttt{client.CertifyKey} command for validity. One should verify the included AIK-Certificate for trustability after successful validation. The included public key may then be used in further steps.

3.8.2 \textbf{CONSTRUCTOR} CertifyKeyValidate

\textbf{Description}

takes the given input values and stores them internally for later validation of the certification via the \texttt{run} method

\textbf{Settings}

none

\textbf{Parameters}

\begin{tabular}{|l|}
\hline
\texttt{OBJECT[]} certifyKeyResult & result of the \texttt{client.CertifyKey} command, contains public key, keyCertification and AIK-Certificate \\
\hline
\texttt{BYTE[]} nonce & nonce that was challenged and to validate the certification against \\
\hline
\end{tabular}

3.8.3 \textbf{PRIVATE} createDigestInfoDER

\textbf{Description}

code from jTPM-Tools. It is used to return the digestInfo properly out of the certification.

\textbf{Settings}

none

\textbf{Parameters}

\begin{tabular}{|l|}
\hline
\texttt{TCBLOBDATA} digest & digest to be properly encoded \\
\hline
\end{tabular}

\textbf{Output}

\texttt{TCBLOBDATA} digestInfo used during verification of the certification signature

3.8.4 \textbf{PUBLIC} run

\textbf{Description}

Implements the necessary steps to validate the given output of a previous \texttt{client.CertifyKey}
3.8 server.CertifyKeyValidation

First the signature of the keyCertification is validated against the AIK-Certificate included in client.CertifyKey’s output. Then the nonce is checked against the given one. Finally the public key included in the output is checked against the one signed in the keyCertification.

**Note:** Please be aware that this method only validates the output without verifying the trustability of the contained AIK-Certificate. This should be done after a successful validation.

**Settings**

none

**Parameters**

none

**Output**

BOOLEAN TRUE if certification is valid, else FALSE
3.9 server.ExternalDataBinding

3.9.1 Description

This server module can be used to bind (i.e. encrypt) data to an external TPM using a public key of a TPM keypair resident inside the TPM. Bound data can only be unbound (i.e. decrypted) inside the TPM it was bound to.

3.9.2 CONSTRUCTOR ExternalDataBinding

Description
takes the given input values and stores them internally for later processing in the run method

Settings
none

| Parameters | Description |
|------------|-------------|
| BYTE[] data | data to be bound/encrypted |
| TCBLOBDATA key | public key to be used to bind data |

3.9.3 PUBLIC run

Description
Implements the necessary steps to bind the given data using the given public key. Data is separated into chunks matching the keysize and passed to the bind method provided by jTSS (see References).

Settings
TPM_KeySize

| Parameters | Description |
|------------|-------------|
| none |

Output
BYTE[] bound/encrypted data to be passed to the corresponding TPM
3.10 server.KeyStorage

3.10.1 Description

The server module `server.KeyStorage` provides easy access to save and load functions for server keys. It can be used to store public/private keypairs and provides an interface to retrieve the keys via tags. It is used by `PCAserver` and `RSAserver` to store their signing keys. If a keypair with the specified tag already exists, the older one will be overwritten.

3.10.2 PRIVATE loadFromFile

Description
loads the mapping file from the predefined filesystem-location.

Settings
KeyStorageBaseDir, KeyStorageDB

Parameters
none

3.10.3 PRIVATE saveToFile

Description
saves the mapping to the predefined filesystem-location

Settings
KeyStorageBaseDir, KeyStorageDB

Parameters
none

3.10.4 PRIVATE getMapping

Description
returns the filename mapped to the given tag

Settings
none

Parameters
STRING tag tag to be searched for

Output
STRING[] filenames of key files mapped to the tag
  [0] public keyfile [1] private keyfile
3.10.5 **PUBLIC** **getPublicKeyFile**

**Description**
returns the filename of the public key

**Settings**
KeyStorageBaseDir

| Parameters | STRING tag | keypair tag |
|------------|------------|-------------|

**Output**
STRING the filename of the public key to the given tag

3.10.6 **PUBLIC** **getPrivateKeyFile**

**Description**
returns the filename of the private key

**Settings**
KeyStorageBaseDir

| Parameters | STRING tag | keypair tag |
|------------|------------|-------------|

**Output**
STRING the filename of the private key to the given tag

3.10.7 **PUBLIC** **getPublicKey**

**Description**
get the public key as **PUBLIC** **KEY** for the given tag

**Settings**
none

| Parameters | STRING tag | keypair tag |
|------------|------------|-------------|

**Output**
**PUBLIC** **KEY** stored under the given tag

3.10.8 **PUBLIC** **getPrivateKey**

**Description**
get the private key as **PRIVATE** **KEY** for the given tag
3.10 server.KeyStorage

Settings
none

Parameters

| Parameter       | Description                           |
|-----------------|---------------------------------------|
| STRING tag      | keypair tag                           |

Output

PRIVATE KEY stored under the given tag

3.10.9 PUBLIC put

Description

put stores the given keypair in the files specified by the user. The user must provide a tag by which the keys can be accessed later. The keys will be stored in Settings.KeyStorageBaseDir, the mapping of tags to keys is stored in Settings.KeyStorageDB.

Settings

KeyStorageBaseDir

Parameters

| Parameter       | Description                           |
|-----------------|---------------------------------------|
| STRING tag      | the user provided tag for the storage of the keypair |
| STRING publicKeyFile | the filename of the public key file       |
| PUBLIC KEY publicKey | the public key to be stored             |
| STRING privateKeyFile | the filename of the private key file    |
| PRIVATE KEY privateKey | the private key to be stored            |

Output

STRING[] of replaced entries, if tag already existed in server.KeyStorage

3.10.10 PUBLIC putPublicKey

Description

stores the given public key in the file specified by the user. The user must provide a tag by which the key can be accessed later. The key will be stored in Settings.KeyStorageBaseDir, the mapping of tags to keys is stored in Settings.KeyStorageDB.

Settings

KeyStorageBaseDir

Parameters

| Parameter       | Description                           |
|-----------------|---------------------------------------|
| STRING tag      | the user provided tag for the storage of the keypair |
| STRING publicKeyFile | the filename of the public key file       |
| PUBLIC KEY publicKey | the public key to be stored             |

Output

STRING[] of replaced entries, if tag already existed in server.KeyStorage
3.10.11 PUBLIC putPrivateKey

**Description**
stores the given private key in the file specified by the user. The user must provide a tag by which the key can be accessed later. The key will be stored in Settings.KeyStorageBaseDir, the mapping of tags to keys is stored in Settings.KeyStorageDB.

**Settings**
KeyStorageBaseDir

| Parameters       | Description                                      |
|------------------|--------------------------------------------------|
| STRING tag       | the user provided tag for the storage of the keypair |
| STRING privateKeyFile | the filename of the public key file              |
| PRIVATEKEY privateKey | the private key to be stored                    |

**Output**
STRING[] of replaced entries, if tag already existed in [server.KeyStorage](/server.KeyStorage)
3.11 server.QuoteValidation

3.11.1 Description
The server module \texttt{server.QuoteValidation} can be used to validate a given quote from a TPM. The quote includes the signed hash value of the desired PCRs and a signed nonce (→ replay protection). The client must provide a valid certificate for the key used to sign the quote. Normally an AIK is used to sign the quote, so a previously acquired AIK certificate from \texttt{[PCA-\texttt{client}] to PCAserver} can be used to verify the quote.

The validation of the quote consists of two major parts:

1. Verification of the signature: the public key used for signing is used to verify the given signature on the nonce and the PCR data.

2. Verification of quote contents: it is verified that the original nonce, supplied by the server, is included in the client's quote. Furthermore, the quote contains the hash value of all quoted PCRs. A pre-calculated PCR value can be supplied to \texttt{server.QuoteValidation}, so that validation will only succeed if the pre-calculated value matches the quoted PCR value.

3.11.2 \texttt{CONSTRUCTOR QuoteValidation}

\texttt{Description} takes the given input values and stores them internally for later validation of the quote via the run method.

\texttt{Settings} none

\texttt{Parameters}

| Type | Description |
|------|-------------|
| \texttt{INT[][] pcrSelection} | array to define which PCRs have been included in the given quote |
| \texttt{TCTssValidation quote} | quote as received from the attesting client |
| \texttt{X509Certificate aikCert} | certificate belonging to the key used for signing. It is provided by the client to retrieve the public key and verify the quote's signature. |
| \texttt{BYTE[][] vPCRs} | pre-calculated vPCR values are provided as \texttt{BYTE[][]}. Multiple vPCRs can be provided. |
| \texttt{BYTE[]} nonce | the anti-replay nonce the server previously provided for the client to be included in the quote |

3.11.3 \texttt{PRIVATE selectPCR}

\texttt{Description} the method is used to return a \texttt{TcBlobData} structure with a \texttt{BYTE[][]} representation of a selected PCR. The bit at the given index \(i\) is set to 1, all other bits are 0.
### 3.11.4 PRIVATE createDigestInfoDER

**Description**

code from jTPM-Tools. It is used to return the digestInfo properly out of the quote.

**Settings**

none

**Parameters**

| Type         | Description                        |
|--------------|------------------------------------|
| LONG index   | index of PCR to select             |

**Output**

TCBLOBDATA containing the BYTE[] representation of the selected PCR

### 3.11.5 PUBLIC run

**Description**

Implements the necessary steps to validate the given quote. First the public key is extracted from the supplied certificate. It is used to verify the quote's signature. A composite object with the given pre-calculated PCR values is created and its hash value is calculated. The hash is checked against the hash value provided by the quote. Finally the quote's nonce is checked. Only if all steps of the validation succeed, the quote will validate. Protection is provided against different attacks. If quote contents are changed after signing, the validation of the signature will fail. If a malicious client provides a modified measurement list, pretending to run only trusted software, the calculated vPCR will not match the quoted and signed PCR value. Anti-replay attack protection is given by the usage of the rolling nonce.

**Settings**

none

**Parameters**

| Type         | Description                        |
|--------------|------------------------------------|
| none         |                                    |

**Output**

BOOLEAN TRUE if quote is valid, else FALSE
This package contains networking classes for sending and receiving data via TCP/IP.
4.1 NetEntity

4.1.1 Description

NetEntity provides an implementation of a networked client-server infrastructure. NetEntity can be initialized as server or as client. Once initialized, it can be used to transfer controls (NetCommand) or Objects.

4.1.2 Constructor NetEntity

Description
Sets up a NetEntity as either server or client. If invoking with serverHostname and serverPort, a client version is instantiated. If only serverPort is given, a server is instantiated.

Settings
none

Parameters

| TYPE   | NAME            | Description                                                  |
|--------|-----------------|--------------------------------------------------------------|
| STRING | serverHostname  | hostname of server the client will connect to (client only)  |
| INT    | serverPort      | port the client will connect to / the server will listen on  |

4.1.3 Public init

Description
Initializes the connection. A socket is created and bound to the specified port. If instance is a server, the socket will start to listen and accept connections.

Settings
none

Parameters

|    |                  |                      |
|----|------------------|----------------------|
| none |                  |                      |

4.1.4 Public close

Description
closes the socket. Returns TRUE, if socket closed successfully, else FALSE.

Settings
none

Parameters

|    |                  |                      |
|----|------------------|----------------------|
| none |                  |                      |

4.1.5 Public sendACK

Description
sends a NetCommand ACK
4.1 NetEntity

Settings
none

Parameters
none

Output
TRUE, if sent successfully, else FALSE

4.1.6 PUBLIC sendNACK

Description
sends a NetCommand NACK

Settings
none

Parameters
none

Output
TRUE, if sent successfully, else FALSE

4.1.7 PUBLIC sendCommand

Description
sends the given NetCommand

Settings
none

Parameters
NetCommand netCommand

Output
TRUE, if sent successfully, else FALSE

4.1.8 PUBLIC sendObject

Description
sends the given object over the connection.

Settings
none
**Parameters**

| OBJECT o | object to send |

**Output**

TRUE if sent successfully, else FALSE

### 4.1.9 PUBLIC receiveACK

**Description**

try to receive a `NetCommand` ACK

**Settings**

none

| Parameters | none |

**Output**

TRUE if ACK received, else FALSE

### 4.1.10 PUBLIC receiveCommand

**Description**

receive a `NetCommand`

**Settings**

none

| Parameters | none |

**Output**

returns the received `NetCommand` if received object is an integer. Else `NetCommand`.UNKNOWN is returned

### 4.1.11 PUBLIC receiveObject

**Description**

waits to receive an Object of the given class.

**Settings**

none

| Parameters | CLASS c | class of the object to be received |
Output
If received object is of specified type, the object is returned, else NULL.

4.1.12 PUBLIC getRemoteIP

Description
returns the IP address of the client, when issued on the server

Settings
none

Parameters
none

Output
STRING IP address of client

4.1.13 PUBLIC getRemoteHostname

Description
returns the hostname of the client, when issued on the server

Settings
none

Parameters
none

Output
STRING hostname of client
4.2 NetCommand

4.2.1 Description

NetCommand provides a mapping between symbolic names and command codes (INT) for convenient use of NetEntity objects.

- UNKNOWN
- ACK
- NACK
- INIT
- CLOSE
- KNOWNHASHES
- STRING
- RA_REQUEST
- RA_RESPONSE
- PCA_REQUEST
- PCA_RESPONSE
5 types

DE.FRAUNHOFER.SIT.TC.ETHEMBA.TYPES

This package contains data types for convenient access to IMA measurements and hash lists.
5.1 MeasurementList

5.1.1 Description
MeasurementList is a data type for convenient conversion and access to IMA measurement data.

5.1.2 CONSTRUCTOR MeasurementList
Description
initializes the MeasurementList with a STRING[][]]

Settings
none

Parameters

| STRING[][] | measurementList |
|------------|-----------------|
| [1] PCR number |
| [2] hash |
| [3] application |

5.1.3 PUBLIC getMeasurementList
Description
returns the MeasurementList as string array representation

Settings
none

Parameters
none

Output
STRING[][] string array representation of the MeasurementList

5.1.4 PUBLIC getMeasurementListForPCR
Description
returns the MeasurementList for a given PCR number as string array representation

Settings
none

Parameters

| INT pcr |
|---------|
| to be filtered PCR number |

Output
STRING[][] string array representation of the MeasurementList for the given PCR number
5.1.5 **PUBLIC** `loadFromFile`

**Description**
loads MeasurementList from a file in IMA-Measurement-Format

**Settings**
none

**Parameters**
| STRING filename | file to load MeasurementList from |

**Output**
`MEASUREMENTLIST`  loaded MeasurementList
5.2 KnownHashesList

5.2.1 Description

KnownHashesList is a data type providing basic database abilities for storing known hash ↔ application mappings.

5.2.2 CONSTRUCTOR KnownHashesList

Description

initializes the KnownHashesList

Settings

none

Parameters

none

5.2.3 PUBLIC put

Description

maps the given hash to the given tag

Settings

none

Parameters

| STRING sha1hash   | hash of a application |
|-------------------|------------------------|
| STRING tag        | path and name of the application that was hashed |

Output

STRING tag/application that was associated to the given tag before, else NULL

5.2.4 PUBLIC get

Description

returns the tag to which the specified hash is mapped to

Settings

none

Parameters

| STRING sha1hash | hash to be searched for |

Output

STRING tag/application that has is associated to, else NULL
5.2.5 **PUBLIC** contains Tag

**Description**
tests if the given tag/application is mapped to one or more hashes

**Settings**
none

**Parameters**

| **STRING tag** | tag/application to be searched for |

**Output**
TRUE if given tag/application is associated with one or hashes, else FALSE

5.2.6 **PUBLIC** containsSha1Hash

**Description**
tests if the given hash is mapped to a tag/application

**Settings**
none

**Parameters**

| **STRING sha1hash** | hash to be searched for |

**Output**
TRUE if given hash is associated with a tag/application, else FALSE

5.2.7 **PUBLIC** contains

**Description**
tests if a given hash is mapped to a given tag/application

**Settings**
none

**Parameters**

| **STRING sha1hash** | hash to be searched for |
| **STRING tag** | tag/application to be associated with |

**Output**
TRUE if given hash is associated with the given tag/application, else FALSE

5.2.8 **PUBLIC** saveToFile

**Description**
saves KnownHashesList to a given file
**Settings**
none

**Parameters**

| Type      | Description                              |
|-----------|------------------------------------------|
| STRING    | filename                                 |

**Description**

loads a KnownHashesList from the disk

**Output**

KNOWNHASHESLIST loaded KnownHashesList or NULL if KnownHashesList could not be loaded
6 utils

DE.FRAUNHOFER.SIT.TC.ETHEMBA.UTILS

This package contains classes and methods for data conversion and standardized algorithms, that may be accessed in a convenient way.
6.1 AES

6.1.1 Description

AES provides access to various AES calculations (encryption/decryption). Each method may also be accessed in a static way. Be aware to pass enough data to the static methods (i.e. AES-Key and IV).

Additionally random and pseudo-random AES-Keys and IVs may be generated using this class.

6.1.2 Constructor AES

Description

constructs an AES object with the given key and initialization vector (IV)

Settings

none

Parameters

| Type   | Description                  |
|--------|-----------------------------|
| SecretKey k | AES key                     |
| BYTE[] IV   | initialization vector (IV)  |

6.1.3 Public encryptObject

Description

encrypts a given Object

Settings

none

Parameters

| Type   | Description                  |
|--------|-----------------------------|
| Object m | object to be encrypted      |

Output

BYTE[] encrypted data

6.1.4 Public encrypt

Description

encrypts a given BYTE[]

Settings

none

Parameters

| Type   | Description                  |
|--------|-----------------------------|
| BYTE[] m | data to be encrypted        |
6.1 AES

Output
BYTE[] encrypted data

6.1.5 PUBLIC decryptObject

Description
decrypts a given BYTE[] back into an OBJECT

Settings
none

Parameters
| BYTE[] c | data to be decrypted |

Output
OBJECT decrypted object

6.1.6 PUBLIC decrypt

Description
decrypts a given BYTE[]

Settings
none

Parameters
| BYTE[] c | data to be decrypted |

Output
BYTE[] decrypted data

6.1.7 PRIVATE crypt

Description
en- or decrypts given BYTE[] with given AES-Key and IV

Settings
aesCipherMode, aesKeySize

Parameters
| INT optMode | operation mode: Cipher.DECRYPT_MODE or Cipher.ENCRYPT_MODE |
|------------|----------------------------------------------------------|
| SECRETKEY k | AES-Key to be used |
| BYTE[] IV | IV to be used |
| BYTE[] input | data to be en- or decrypted |
Output
BYTE[] en- or decrypted data

6.1.8 PUBLIC generateKey
Description
generates a random AES-Key

Settings
none

Parameters

Output
SECRETKEY generated key

6.1.9 PUBLIC generateKey
Description
generates a pseudo-random AES-Key by using an initial seed

Settings
none

Parameters

Output
SECRETKEY generated key

6.1.10 PUBLIC generateIV
Description
generates a pseudo-random IV by using an initial seed

Settings
none

Parameters

Output
BYTE[] generated IV
6.2 Helpers

6.2.1 Description
Helpers provides some useful methods for data conversion and serialization.

6.2.2 **PUBLIC byteToHexString**

**Description**
converts a given BYTE[] into a Hex-String

**Settings**
none

**Parameters**
- BYTE[] data  
  data to be converted

**Output**
STRING Hex-String representation of given data

6.2.3 **PUBLIC hexStringToByteArray**

**Description**
converts a given Hex-String into a BYTE[]

**Settings**
none

**Parameters**
- STRING hexstring  
  string to be converted

**Output**
BYTE[] converted data

6.2.4 **PUBLIC leadingZeros**

**Description**
adds leading zeroes to a given number

**Settings**
none

**Parameters**
- INT num  
  number to be padded with zeroes
- INT length  
  length of result

**Output**
STRING string representation of given number with padded zeroes to match the given length
6.2.5 **PUBLIC** saveObjectToFile

**Description**
saves a given object (needs to be serializable) to the disk

**Settings**
none

**Parameters**
| Type    | Description            |
|---------|------------------------|
| OBJECT  | object to be saved     |
| STRING  | location of saved object |

6.2.6 **PUBLIC** saveBytesToFile

**Description**
saves bytes to a file

**Settings**
none

**Parameters**
| Type   | Description          |
|--------|----------------------|
| BYTE[] | bytes to be saved    |
| STRING | file to be saved to  |

6.2.7 **PUBLIC** loadObjectFromFile

**Description**
loads an object from a given file

**Settings**
none

**Parameters**
| Type    | Description                  |
|---------|-----------------------------|
| STRING  | file to load object from    |

**Output**
OBJECT loaded from given file

6.2.8 **PUBLIC** loadBytesFromFile

**Description**
loads a file byte-wise

**Settings**
none

**Parameters**
| Type    | Description                  |
|---------|-----------------------------|
| STRING  | file to be loaded from      |
Output
BYTE[] loaded bytes

6.2.9 PUBLIC Object2Bytes

Description
converts an object to bytes

Settings
none

Parameters

| Parameter | Description         |
|-----------|---------------------|
| OBJECT o  | object to be converted |

Output
BYTE[] converted bytes

6.2.10 PUBLIC Bytes2Object

Description
converts a bytes back to an object

Settings
none

Parameters

| Parameter | Description         |
|-----------|---------------------|
| BYTE[] b  | bytes to be converted |

Output
OBJECT converted object
6.3 SHA1

6.3.1 Description
SHA1 provides access to various SHA1 calculations.

6.3.2 PUBLIC main

Description
generates SHA1-Hash of given string/hex-string

Settings
none

Parameters

| STRING[] args | [0] String to be hashed
|              | [1] if `-h` the given String is assumed to be a hex representation |

Output
SHA1-Hash of given data

6.3.3 PUBLIC hashByteToByte

Description
calculates SHA1-Hash of given BYTE[]

Settings
none

Parameters

| BYTE[] input | data to be hashed |

Output
BYTE[] SHA1-Hash of given data

6.3.4 PUBLIC hashHex

Description
calculate SHA1-Hash for given Hex-String

Settings
none

Parameters

| STRING input | data to be hashed (Hex-String) |
6.3 SHA1

Output
STRING SHA1-Hash of given data

6.3.5 PUBLIC hash

Description
calculate SHA1-Hash for given String

Settings
none

Parameters
| Parameter | Description |
|-----------|-------------|
| STRING input | data to be hashed |

Output
STRING SHA1-Hash of given data

6.3.6 PUBLIC randomHash

Description
returns SHA1-Hash of a random generated number

Settings
none

Parameters
| Parameter | Description |
|-----------|-------------|
| none | none |

Output
BYTE[] of SHA1-Hash
7 Demonstrators

For demonstration purposes we provide two simple yet useful demonstrators. These demon-
strators will prove that the Remote Attestation Protocol developed and implemented in ethemba
serves as reliable attestation method.

To prove reliability the following demonstrators inject bad code into an application that was
added to the RAserver's KnownHashesList. This KnownHashesList acts as a white-list repre-
senting SHA1 hashes for well-known and valid applications. If an application was run on the
attesting system that either is not included in the KnownHashesList or whose SHA1 hash does
not match the one inside the KnownHashesList, Remote Attestation fails.

To prove this we provide a demonstrator that will compile a C-program out of two sources. The
demonstrators can be found in the subfolder demo and contain the following files:

- demoevil.sh
- demogood.sh
- helloworldevil.c
- helloworldgood.c

7.1 demogood.sh

This demonstrator will compile the code contained in helloworldgood.c to the binary hel-
loword.

```c
#include <stdio.h>

int main()
{
    printf("sawubona!\n");
    printf("means 'hello' in zulu\n");
    return 0;
}
```

Listing 7.1: helloworldgood.c

We assume this code to be “good” and therefore included its SHA hash into the RAserver's
KnownHashesList. Executing this program will not break a successful Remote Attestion.
7.2 demoevil.sh

This demonstrator will compile the code contained in `helloworldevil.c` to the binary `helloworld`.

```c
#include <stdio.h>

int main()
{
    printf("hamba kahle!\n");
    printf("means 'goodbye' in zulu\n");
    return 0;
}
```

Listing 7.2: helloworldevil.c

This can be seen as an attack pretending execution of a well-known program. As its SHA1 hash now differs from the one included in the RAserver's KnownHashesList, Remote Attestion now has to fail.
8 Settings
ethemba uses jTSS as underlying implementation of the TCG Software Stack for Java. jTSS implements all the TSS layers directly in Java by staying very close to the TPM specifications stated by the TCG. It is developed and maintained at the Institute for Applied Information Processing and Communication (Institut für angewandte Informationsverarbeitung und Kommunikation, IAIK) at Graz University of Technology (TU Graz).

IAIK additionally provides a set of command line tools for basic operations on TPMs. These tools include several modules for attesting a platform (e.g. generate AIKs, certificates and handle TPM quotes).

Some code of jTPMtools was re-used in ethemba to allow for convenient and reliable access. These modified modules can be found in a separate package underneath ethemba (jTPM-tools).

For further information on jTSS and the OpenTC Project visit the following websites:

- **Trusted Computing for the Java Platform**
  - [http://trustedjava.sourceforge.net](http://trustedjava.sourceforge.net)

- **OpenTC Project**
  - [http://www.opentc.net](http://www.opentc.net)

- **Institute for Applied Information Processing and Communications**
  - [http://www.iaik.tugraz.at](http://www.iaik.tugraz.at)
...just in case you didn’t know...