Exercise 6: TPM Key Management and Key Replication Mechanisms

1 Introduction

In this exercise we provide an overview of the TPM key management architecture and key replication procedures, in particular the TPM key backup, key migration and key maintenance mechanisms. As practical work we will develop a simple application which goal is to move a TPM generated migratable key to another platform.

Before starting with practical work, please read followed theoretical information relevant to targeted work. More information can be found in slides from Trusted Computing lectures and in number of documents from Trusted Computing Group: TPM Main specification, Part 1, Design Principles [7], TPM Main specification, Part 2, Structures [4], TPM Main specification, Part 3, Commands [5], TCG Specification, Architecture Overview [6], Interoperability Specification for Backup and Migration Services [3]. As support for practical work it is recommended "A Practical Guide to Trusted Computing" book.

1.1 Key attributes

All keys managed by the TPM have an attribute designation of migratable or non-migratable. The key attribute determines whether a key may be transferred from one TPM to another. Attribute value is established at the time the key is created and cannot be changed.

1. Non-migratable keys (NMK) are bound to a single TPM. There are keys that are unique to a single TPM and can't be migrated or exported from the TPM. A non-migratable key is guaranteed to reside in a TPM-shielded location. A TPM can create a certificate stating that a key is a NMK.

2. Migratable keys (MK) are cryptographic keys which are not bound to a specific TPM. They can be generated either inside or outside of a TPM and with suitable authorization, can be integrated inside a TPM or moved from one TPM to another one. MKs are only trusted by the party who generated them (e.g., the user of the platform). A third party has no guarantee that such a key has indeed been generated on a TPM.

3. Certified Migratable key (CMK): This type of encryption key was introduced in version 1.2 of the TCG specification. It is a kind of key that is halfway between a non-migratable key and a migratable key. They are generated inside of a TPM, but can be migrated to another TPM. At the time they are created, the creator has to pick up a Migration Authority (MA) or Migration Selection Authority (MSA), which will have the authority to migrate the key. CMKs can both be migrated and also be considered secure (if you trust the MSA and MAs to migrate the key).

1.2 Key types

TCG defines 7 key types. Each type carries with it a set of restrictions that limits its use. 6 types are asymmetric cryptographic keys and only one key type classified as Authentication key is symmetric.

Signing keys are asymmetric general purpose keys used to sign application data and messages. Signing keys can be migratable or non-migratable. Migratable keys may be exported / imported between TPM devices. The TPM can sign application data and enforce migration restrictions.

Storage keys are asymmetric general purpose keys used to encrypt data or other keys. Storage keys are used for wrapping keys and data managed externally (outside a TPM).

Endorsement key (EK). At the end of TPM chip fabrication (after final testing), the manufacturer generates a 2048 bit private/public Endorsement key pair in the TPM. This is stored in such a way that the private key (PK) can no longer be read out, but can only be used internally in the TPM. Endorsement Key is used to decrypt owner authorization data at the time a platform owner is established and to decrypt messages associated with AIK creation.

Attestation identity keys (AIK): These signature keys are exclusively used to sign data originated by the TPM (such as TPM capabilities and PCR register values). AIKs provide anonymity of platforms including a TPM. They are locally created by the TPM. The public part is certified by an issuer, a Privacy Certification Authority (Privacy CA) stating that this signature key is really under control of a secure TPM.

Bind keys may be used to encrypt small amounts of data (such as a symmetric key) on one platform and decrypt it on another.

Legacy keys are keys which can be exported to another TPM after creation and may be used for signing and encryption operations. They are referenced to specific users and their application fields (e.g. user specific mail signing and encryption, user specific SW authentication or service access keys and similar).

Authentication keys are symmetric keys used to protect transport sessions involving the TPM.

1.3 TPM Key Management

The complete trust and security functionality of a Trusted Computing Platform is based on the capabilities of the TPM to protect these keys and certificates.

A TPM contains a Root of Trust of Storage (RTS) which protects data and keys entrusted to the TPM. The RTS manages a small amount of volatile storage inside the TPM device that is used to hold currently used keys (key slots). Unused keys may be

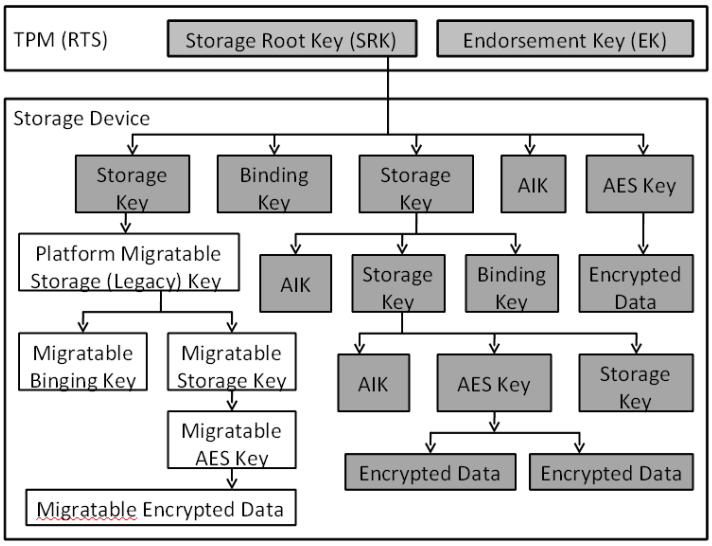

Figure 1: Key Hierarchy

encrypted with a storage key and moved off the TPM chip, e.g., to a hard disk drive. The key slots of the TPM are managed by a trusted service outside the TPM which is called Key Cache Manager (KCM). The storage key might be encrypted with another storage key which leads to a key hierarchy as shown in figure 1 with the Storage Root Key (SRK) being the root. The SRK is generated during the process of taking logical ownership of the platform and is embedded into the TPM. It can be re-generated by creating a new platform owner which destroys the previous key hierarchy and all the data and keys it contains.

As shown at the architecture in Figure 1, the migratable key chain is designed so that only one key, the Legacy key (or Platform Migratable Storage Key) needs to be migrated in order to take all the keys below in to a new TPM.

1.4 Key replication mechanisms

1.4.1 Migration

It could be necessary that keys are required on different platforms, which are handled by a user alternatively. Sharing the same key across multiple platforms may be achieved by using key migration mechanisms. Key migration mechanisms allow the private keys from TPM-protected key hierarchy to be attached to other TPM-protected storage trees.

Migratable Keys (MK) can be moved to another TPM by using either rewrap (`TPM_MS_REWRAP`)

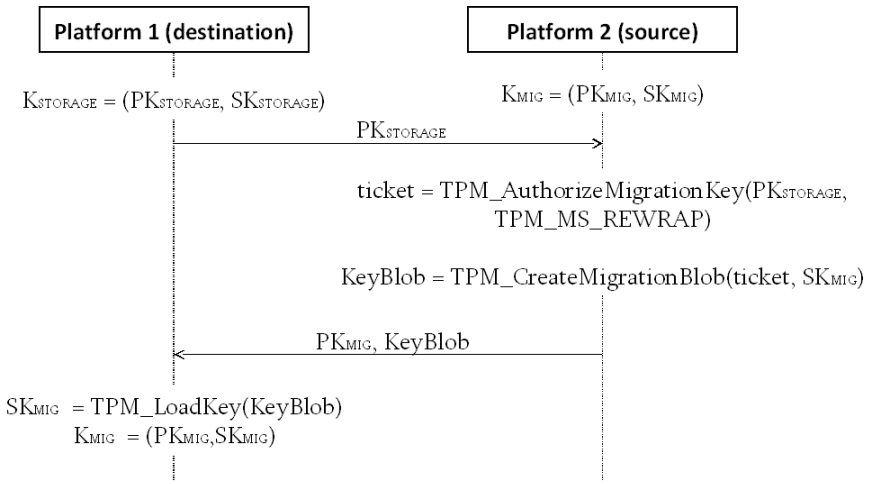

Figure 2: *Migration according to rewrap scheme*

or migration scheme (`TPM_MS_MIGRATE`).

Rewrap scheme. To migrate a key with a `TPM_MS_REWRAP` scheme 2, a destination TPM selects a storage key which will be used as a parent for the migratable key and sends it's public part to a source TPM. The source TPM rewraps the migratable key under the destination public key. The using of destination public key should be authorized by owner with `TPM_AuthorizeMigrationKey` command and rewrap procedure can be done with command `TPM_CreateMigrationBlob`. Resulting blob is then forwarded to the destination TPM in conjunction with a plaintext object describing the public key from the key pair to be migrated. The destination TPM can load blob with `TPM_LoadKey` command.

Scheme with participation of MA. The migration mode `TPM_MS_MIGRATE` involves the use of an intermediary (see Figure 3), Migration Authority (MA). In this scenario the source TPM encrypts the migratable key under the public key of the MA. There is no assurance that the public key does actually represent the MA public key, therefore the owner of source platform must authorize the correct destination with a command `TPM_AuthtorizeMigrationKey` before creates migratable blob with a command `TPM_CreateMigrationBlob`. Blob is passed to MA, where it is unwrapped and rewrapped under the public key of the destination TPM (MA runs `TPM_MigrateKey` command to do that). To prevent the intermediary from getting unauthorised access to the migrated key, the key is additionally protected with XOR encryption with randomly generated string. This string is sent directly to a destination TPM omitting the Migration Authority. The destination TPM can install the migratable blob with the TPM command `TPM_ConvertMigrationBlob`. Output of this command is a key blob including migratable key encrypted with a storage key. Then the key blob can be loaded into the

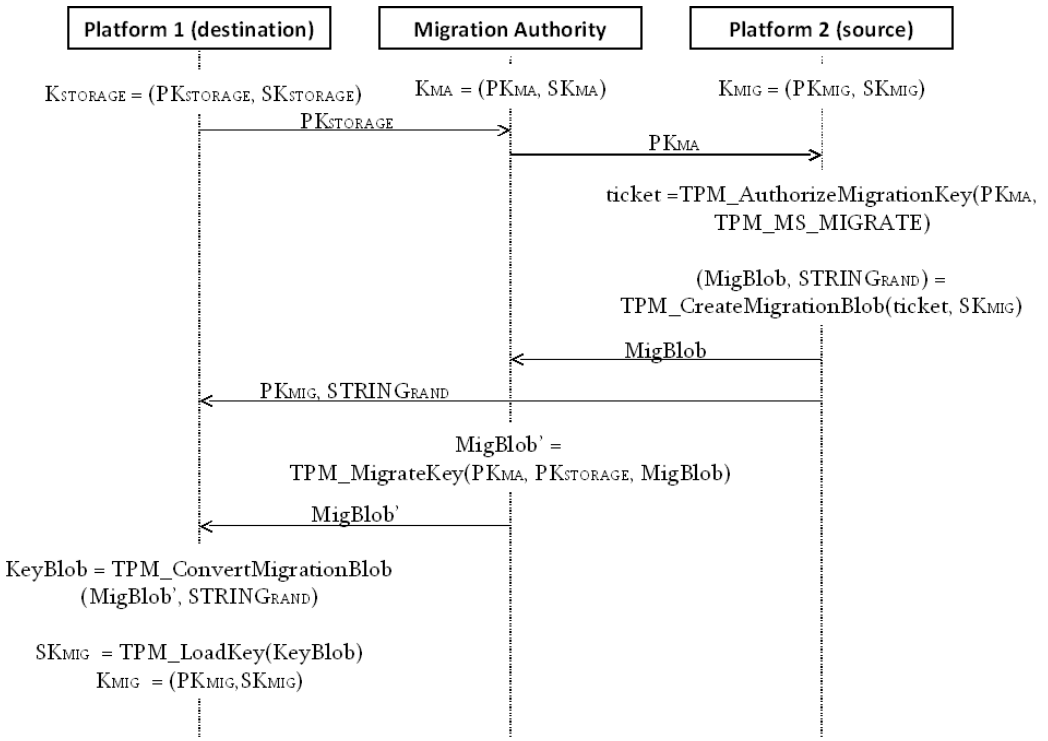

Figure 3: *Migration with a mediation of Migration Authority*

TPM with a command `TPM_LoadKey`.

Certified Migratable Keys (CMKs) can be migrated only with help of MA (`TSS_MS_MIGRATE scheme`), the migration with `TPM_MS_REWRAP` scheme is restricted. To enforce authorised control for CMK destination, either owner authorization or a Migration Selection Authority (MSA) can be used. If MSA was selected as intermediary during CMK creation, it will be then responsible for selection allowed MAs. Otherwise the owner should use a command `TPM_CMK_ApproveMA` to authorize a public key of intermediating MA.

1.4.2 Backup

Backup mechanism serves to eliminate lost of data in case of TPM failure or computer theft. It allows to do a secure backup and to store the backup data on an external storage device. The archive contains encrypted copies of migratable key chains and duplicates of data files encrypted with these migratable keys. In case of disaster the data encrypted by migratable keys can be restored via restore operation. Note that during backup operation non-migratable keys are not copied. It means that all non-migratable key material and data encrypted with non-migratable keys will not be accessable in case of TPM failure.

1.4.3 Maintenance

To recover all TPM-shielded material including non-migratable keys in case of any disaster such as TPM failure the Maintenance mechanism (`TPM_MS_MAINT`) is recommended by TCG specification. Maintenance procedure defines the process of secure moving non-migratable keys from one TPM to another and it can only be completed with the co-operation of the TPM owner and the platform manufacturer. Maintenance procedure must only be performed between two platforms of the same manufacturer and model. After successful completion of this procedure the source TPM must be destroyed to prevent existing of two TPM modules with the same credentials. Supporting of maintenance features is not mandatory and is not implemented yet by any manufacturer. Maintenance process is explained in details in lectures of Trusted Computing, see chapter 4.

2 Theoretical Assignments

Please prepare the theoretical assignments **at home**. You have to hand in your answers at the beginning of each practical assignment!

1. Answer some questions related to keys:
   a) Is Attestation Identity Key (AIK) migratable or non-migratable key?
   b) Can migratable keys be trusted?
   c) Do public part of migratable key needs to be migrated during migration procedure?

   Explain your statements.

2. Which mechanisms are provided for key migration? Give a short description.

3. Imagine that once things go wrong and one of the numbered below disasters happen:
   a) Your laptop is stolen
   b) Motherboard of your computer is damaged
   c) The hard drive of your computer is corrupt

   Which mechanism do you need to use in listed cases to restore your platform functionality? Explain your statements.

4. Which mechanism is used to migrate TPM keys during making system backup? Which keys are migrated then? Explain your answer.

3 Practical Assignments

Your general task for this extercise is to create a migration key and migrate it to another TPM using two schemes: `TPM_MS_REWRAP` and `TPM_MS_MIGRATE`.

3.1 Implementing Migration Scheme TPM_MS_REWRAP

All migration proccess in wrap mode illustrated on Figure 2 can be devided into four steps:

1. Source platform creates migratable key.

   Input: None. Output: Public part of migratable key (`PKmig`), encrypted private part of migratable key (`KeyBlobLocal`).

2. Destination platform creates migration storage key.

   Input: None. Output: A migration storage key which will be used to migrate a migratable key (`PKstorage`).

3. Source platform wraps created migratable key under migration storage key of destination platform.

   Input: Encrypted private part of a migratable key (`KeyBlobLocal`), migration storage key of destination platform (`PKstorage`). Output: Encrypted private part of a migratable key (`KeyBlob`).

4. Destination platform loads migratable key wrapped with own storage key.

   Input: An encrypted pritvate part of a migratable key (`KeyBlob`), public part of a migratable key (`PKmig`). Output: None.

Here we denoted as input required parameters to implement each step. Output parameters indicate data created as a result of performing these steps.

You have to implement these steps by extending some soucre code files given to you as a start point. Three source code files are available on *Exercise Data*: `platform1.c`, `platform2.c` and `utilities.c`.

The file `platform1.c` represents a destination platform where migratable key will be migrated. The file `platform2.c` represents a source platform which will create and then migrate a migratable key. The file `utilities.c` contains some utility functions used by both platforms. You have to extend `platform1.c` and `platform2.c` files while `utilities.c` file is complete and doesn't need any extention.

As it was already mentioned above, the destination platform is represented by `platform1.c` file. You can compile this file with `gcc -o platform1 -ltspi -lgcrypt utilities.c platform1.c`. Then compiled, you will get a binary file. If you run it as following: `./platform1`, you will get information about usage:

```
Usage:  platform1 [option]
--genMigrationKey Generate a migration key
```

```
--installMigBlob Install a migratable key to a platform
```

Source code of source platform is represented in a file platform2.c. You can compile it with `gcc -o platform2 -ltspi -lgcrypt utilities.c platform2.c`. Execution `./platform2` will print to you the following help message:

```
Usage:  platform2 [option]
--genMigratableKey Generate a migratable key
--createMigBlob Migrate a migratable key
```

Union of these options represents four steps required for implementation overall migration process. All required input/output parameters for these steps should be read /written from/to files. Reading/writing parameters from/to files is already included into given source codes and don't have to be implemented. Filenames for parameters are hardcoded.

3.1.1 Step 1. Source Platform: Creation of Migratable Key

To create migratable key on the source platform, extend a function `MyFunc_genMigratableKey` of the file `platform2.c`. The parent key of the new key should be SRK of the source platform (or SRK2). To create a new key, you should perform following operations:

1. Create a key object `TSS_OBJECT_TYPE_RSAKEY` with `Tspi_Context_CreateObject` function. Specifty the next flags for the new key object: `TSS_KEY_TYPE_LEGACY`, `TSS_KEY_MIGRATABLE`, `TSS_KEY_SIZE_2048`, `TSS_KEY_AUTHORIZATION`. As result, the `TPM_KEY` structure will be created described in table 1 (more details you can find in page 88 of [4].

2. Specify the `keyUsage` field of `TPM_KEY` structure to define operations permitted with this key. For that you should use function `Tspi_GetPolicyObject` with parameter `TSS_POLICY_USAGE` to get handler to this field, then the field can be changed with function `Tspi_Policy_Set_Secret`.

3. Define migration policy for a new key: Create migrate policy object `TSS_POLICY_MIGRATION` with function `Tspi_Context_CreateObject`, specify value with a function `Tspi_Policy_Set_Secret` and assign a new policy object to the key with a `Tspi_PolicyAssignToObject` call.

4. Set encryption scheme to `TSS_ES_RSAESPKCSV15` with function `Tspi_SetAttribUint32` (with parameter `TSS_TSPATTRIB_KEYINFO_ENCSCHEME`).

5. Generate a key pair by calling a function `Tspi_Key_CreateKey`.

Hint: an example of code for a key creation is available in [2]. It just need to be adopted for key type `TSS_KEY_MIGRATABLE`.

After the migratable key is created, you have to get it's public and encrypted private parts (to save them into files). To be able to work with created key, you should first load it with `Tspi_Key_LoadKey` function. Then you can get public modulus with

Type	Name	Description
TPM_STRUCT_VER	ver	Always 1.1.0.0
TPM_KEY_USAGE	keyUsage	The TPM key usage that determines the operations permitted with this key
TPM_KEY_FLAGS	keyFlags	The indication of migration, redirection etc.
TPM_AUTH_DATA_USAGE	authDataUsage	The indication of the conditions where it is required that authorization be presented.
TPM_KEY_PARMS	algorithmParms	The information regarding the algorithm for this key
UINT32	PCRInfoSize	The length of the pcrInfo parameter. If the key is not bound to a PCR this value should be 0
BYTE*	PCRInfo	A structure of type TPM_PCR_INFO, or an empty array if the key is not bound to PCRs
TPM_STORE_PUBKEY	pubKey	The public portion of the key
UINT32	encDataSize	The size of the encData parameter
BYTE*	encData	An encrypted TPM_STORE_ASYMKEY structure or TPM_MIGRATE_ASYMKEY structure

Table 1: Parameters for TPM_KEY structure created with `Tspi_Context_CreateObject`

`Tspi_Key_GetPubKey` function, and private part can be accessed with `Tspi_GetAttribData` function with `TSS_TSPAATRIB_KEYBLOB_BLOB` parameter.

How do you think, can we migrate private key blob extracted here on the new platform? Justify your answer!

3.1.2 Step 2. Destination Platform: Creation of Storage Migration Key

To simplify the exercise, we will use SRK of the destination platform (SRK1) as a migration storage key. Then you don't need to create it, just get public part of SRK!

Extend function `MyFunc_genMigKey` of the file `platform1.c`. Use the `Tspi_TPM_OwnerGetSRKPubKey` function to get public key of SRK.

3.1.3 Step 3. Source Platform: Wrapping the Migratable Key under Migration Storage Key of Destination Platform

You have to extend a function `MyFunc_createMigBlob` of the file `platform2.c` to implement this step.

Fist of all, you should load public part of migration storage key (or SRK1 in our case) into the TPM. For that you need to:

1. Create a key object for migration storage key. Repeat operation 1 from section 3.1.1 to do this. Specify the following flags for the key object: `TSS_KEY_TYPE_STORAGE`, `TSS_KEY_SIZE_2048`, `TSS_KEY_VOLATILE`, `TSS_KEY_AUTHORIZATION`, `TSS_KEY_NOT_MIGRATABLE`.

2. Specify `keyUsage` of migration storage key. This operation is similar to operation 2 from 3.1.1 section.

3. Set encryption scheme to `TSS_ES_RSAESOAEP_SHA1_MGF1` volume with function `Tspi_SetAttribUint3` (use parameter `TSS_TSPATTRIB_KEYINFO_ENCSCHEME`).

4. Load public modulus of migration storage key into the key object structure. Use `Tspi_SetAttribData` function with paramener `TSS_TSPAATRIB_KEYBLOB_PUBLIC_KEY`.

Next, you should load a migratable key into the TPM an authorize it's usage.

1. Load the key with function `Tspi_Context_LoadKeyByBlob`.

2. Repeat opeartions 2-3 from the proccess of migratable key creation (section 3.1.1).

Next, you have to authorize usage of migration storage key for migration. You can do it by creation an authorization ticket with function `Tspi_TPM_AuthorizeMigrationTicket`. Use `TSS_MS_REWRAP` parameter here.

And, at the end, create the migration blob by calling `Tpsi_Key_CreateMigrationBlob` function.

3.1.4 Step 4. Destination Platform: Loading the Migratable Key

You should modify a function `MyFunc_installMigBlob` of the file `platform1.c` to implement this step.

Create the object for migratable key first and set object's parameters. For that you just repeat steps 1-4 from 3.1.1. Additionally load public key into the key object (as it was done in section 3.1.3, 4th operation). Then finished, you should call function `Tspi_Context_LoadKeyByBlob` to load migratable key into the TPM.

How do you think, which key will be a parent key for a migratable key on this platform? Justify your answer!

3.2 Implementing Migration Scheme TPM_MS_MIGRATION

Due to in reality no Migration Authority has existed yet, we will simplify basic scheme by excluding Migration Authority and authorizing the public key of destination platform instead of migration authority public key. Resulting scheme is illustrated on the Figure 4. Such a scheme includes also 4 steps:

1. Source platform creates migratable key.

   Input: None. Output: Public part of migratable key (`PKmig`), encrypted private part of migratable key (`KeyBlobLocal`).

2. Destination platform creates migration storage key.

   Input: None. Output: A migration storage key which will be used to migrate a migratable key (`PKstorage`).

3. Source platform wraps created migratable key under migration storage key of destination platform.

   Input: Encrypted private part of a migratable key (`KeyBlobLocal`), migration storage key of destination platform (`PKstorage`). Output: A blob with a migration key (`MigBlob`), random string (`STRINGrand`).

4. Destination platform loads migratable key wrapped with own storage key.

   Input: random string `STRINGrand`, a migration blob `MigBlob`, public part of a migratable key (`PKmig`). Output: encrypted private part of a migratable key (`KeyBlob`).

Firts two steps are similar to corresponding steps of TPM_MS_WRAP migration scheme. It means you have to modify only third and fourth ones to implement simplified TPM_MS_MIGRATION scheme.

3.2.1 Step 3. Source Platform: Wrapping the Migratable Key under Migration Storage Key of Destination Platform

You have just to change the parameter of the function `Tspi_TPM_AuthorizeMigrationTicket` from `TSS_MS_REWRAP` to `TSS_MS_MIGRATION`.

3.2.2 Step 4. Destination Platform: Loading the Migratable Key

This step needs a little more changes. Open the file `platform1.c` and find a place where `TPM_MS_REWRAP` scheme migratable key should be loaded. Here remove call `Tspi_Context_LoadKeyByBlob` and use `Tspi_Key_ConvertMigrationBlob` instead to install a migratable blob. Then get private part of a migratable key to be able to write it to the output file.

How do you think, could such simplified scheme be useful in a real life? Which benefits gives us mediation of MA in full scheme `TPM_MS_MIGRATE`? Justify your answer!

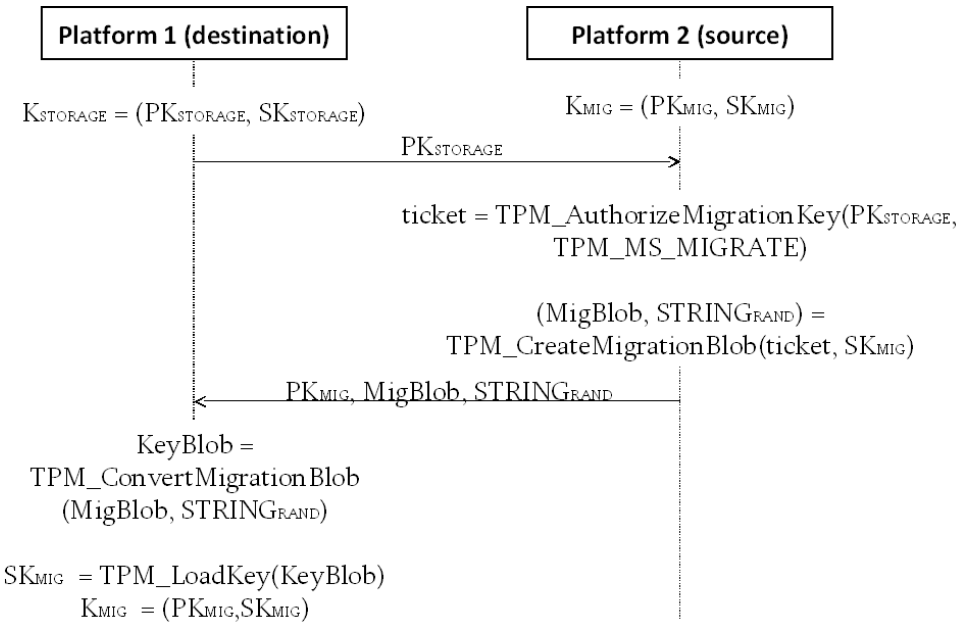

Figure 4: *Simplified migration scheme TPM_MS_MIGRATION without Migration Authority*

References

[1] Official website of lecture "trusted computing". 2009. doi: http://www.ei.rub.de/studierende/lehrveranstaltungen/231/.

[2] David Challener, Kent Yoder, Ryan Catherman, David Safford, and Leendert Van Doorn. *A practical guide to trusted computing*. IBM Press, 2007. ISBN 9780132398428. doi: http://my.safaribooksonline.com/9780132398428?tocview=true.

[6] Trusted Computing Group. Tcg architecture overview, version 1.4. 2007. doi: http://www.trustedcomputinggroup.org/.

[3] Trusted Computing Group. Infrastructure work group interoperability specifications for backup and migration services specification, version 1.0. 2005. doi: http://www.trustedcomputinggroup.org/.

[7] Trusted Computing Group. Tpm specification version 1.2 revision 103: Part 1 - design principles. 2007. doi: http://www.trustedcomputinggroup.org/.

[4] Trusted Computing Group. Tpm specification version 1.2 revision 103: Part 2 - structures. 2006. doi: http://www.trustedcomputinggroup.org/.

[5] Trusted Computing Group. Tpm specificationversion 1.2 revision 103: Part 3 - commands. 2006. doi: http://www.trustedcomputinggroup.org/.

[8] Chris Mitchell, editor. *Trusted Computing*. Professional Applications of Computing Series 6. 2005. doi: http://books.google.de/books?id=-298zQJnJSwC&pg=PA73&lpg=PA73&dq=rewrap+migration+mode+TCG&source=bl&ots=hfrAd4IsSX&sig=zjJRwuFg5U1rW78ehBQQOL4Ywi8&hl=de&ei=2aPrSY6WHc6S_Qb4rtDvAw&sa=X&oi=book_result&ct=result&resnum=2#PPP1,M1.