Audit and backup procedures for Hardware Security Modules

Túlio Cicero Salvaro de Souza
LabSEC – UFSC[*]
Florianópolis – SC – Brasil
salvaro@inf.ufsc.br

Jean Everson Martina[†]
Computer Laboratory - University of Cambridge
Cambridge – United Kingdom
Jean.Martina@cl.cam.ac.uk

Ricardo Felipe Custódio
LabSEC – UFSC[*]
Florianópolis – SC – Brasil
custodio@inf.ufsc.br

ABSTRACT

Hardware Security Modules (HSMs) are an useful tool to deploy public key infrastructure (PKI) and its applications. This paper presents necessary procedures and protocols to perform backup and audit in such devices when deployed in PKIs. These protocols were evaluated in an implementation of a real HSM, enabling it to perform secure backups and to provide an audit trail, two important considerations for a safe PKI operation. It also introduces a ceremony procedure to support the operation of such HSMs in a PKI environment.

Categories and Subject Descriptors

C.2 [**Computer communication network**]: Miscellaneous; D.4.6 [**Security and protection**]: Cryptographic controls; E.3 [**Data Encryption**]: Public key cryptosystems

General Terms

Algorithms, Management, Security

Keywords

Key Management, Public Key Infrastructure, Embedded Cryptographic Hardware, Hardware Security Module, Key Life-cycle, PKI Ceremony

1. INTRODUCTION

Securely managing keys is one of the most important and resource consuming tasks required to guarantee the security on a public key cryptosystem. This is due to a close relationship between security and the proper management of private keys. A public key cryptosystem can be considered secure as long as the private keys are secured. Taking this as a premise, it should be guaranteed that a (private) key is strictly secure during all events in its life cycle. This goal can be achieved by designing systems to securely create, manage and destroy (private) keys, maintaining an audit trail of every operation which was done during their existence. Such systems are known as Hardware Security Modules (HSMs).

HSMs are specialised tamper-proof devices in which cryptographic functions and embedded software have been built to properly manage keys and control their life cycles. They are designed in such a way that if an unauthorised attempt to access them is made, this is considered an attempt to tamper and all critical internal parameters and keys are destroyed.

Although very common in the banking industry, HSMs are also desirable in PKI, but not always implemented. As shown in Table 1, their common usage in the banking industry leads to specialisation of the HSMs to perform tasks such as PIN calculations or payment protocols, that are suitable in such industry.

In a PKI environment, the required characteristics are different from those for banks. In a PKI HSM, it is necessary to establish who can configure, who can access the keys and who can audit the system. For a PKI HSM, it is desirable to have at least three different roles for users.

It is also recommended not to establish these roles for single users since they are easily corruptible, but also to use triggered group mechanisms, such as defined by Shamir[17] and Blakley [2].

Table 1: Comparison between Bank and PKI HSMs

Bank HSMs	PKI HSMs
PIN Calculation	Strong authentication
Role based authentication	Identity based authentication
Dual key entry	Strict key life-cycle control
Payment protocols	Fully auditable operation
Cryptographic speed	Triggered group mechanisms

Normally, it is established groups of administrators (also known as security officers), operators (also known as users), and auditors, who can track all operations done by the previous groups. The administrators are responsible for creating auditors, operators and the keys managed by the latter. The operators are responsible for releasing the keys for use

[*]Computer Security Laboratory at Federal University of Santa Catarina.

[†]Supported by CAPES Foundation/Brazil on grant #4226-05-4

IDtrust '08 March 4-6, 2008, Gaithersburg, MD
Copyright 2008 ACM 1978-1-60558-066-1 ...$5.00.

in an application. The auditors are responsible for analysing the hardware's logs, which register everything that happens inside the HSM - for instance, when a key is used. Responsibility and trust in the keys generated, used and destroyed in HSMs are anchored in these three groups. Moreover, every person who is designated as a member of a group should be carefully selected and all tasks executed by him be registered by automated devices in the PKI context.

Much has been written about HSMs, especially concerning administrators and operators. However, there is no detailed information about audit schemes or backup procedures in HSMs. All developments in this field were done by companies[9, 14, 10, 5, 11], which never published all their internal mechanisms due to industrial secrets concerns and users, and people outside these companies, have no detailed information about their internal protocols.

In this context, "backup" is a way of making a copy of the internal state of the HSM. With a backup, one can prepare new hardware and put it to work as a fully functional copy of the previous HSM. Additionally, the HSMs are products produced by specialised companies and are given some evaluation tests, as described by FIPS 140-2[8].

The HSMs are very expensive devices to use in places such as universities and research centres. There are many of these institutions that would like to deploy a PKI for internal use. Due to the high cost of the HSMs, most of these institutions decide not to use them. Keys are kept in the memory of the host machine which runs the certificate management system. This is a security concern.

Recently, the National Education and Research Network (RNP), a Brazilian social organisation, has created a working group called GT ICPEDU, the main aim of which is to study and develop hardware and software to deploy PKIs for its members. One of its projects was to design an HSM. The hardware architecture of this HSM was conceived by the GT ICPEDU and built by a local company. The software implementing the full keys' life cycle protocol designed for use in PKI environments, called OpenHSM, was the product of an undergraduate final work [18] and a master's thesis [12]. The keys' life cycle protocol passed through several stages of development and improvement. Part of its final version was described by Martina [13]. Two important parts of the protocol, which have not been described yet, are presented by this paper: backup procedures and an audit control mechanism.

This paper considers in section 2, why auditors and backup are important issues in the use of HSMs in PKI deployments and presents related works about these services. Section 3 describes the premises used to state the audit and backup schemes and two basic algorithms used to create and authenticate groups. Section 4 describes the auditing scheme while section 5 describes in detail the backup schemes. Section 6 presents the certification practice statement, in which is described the steps to be followed when a PKI is deployed and briefly introduces the backup-file creation ceremony. The statement includes policies and tasks, and explains how the witnesses can guarantee the backup procedure. Section 7 gives an overview of the main contributions to the field this paper presents and the next step to be undertaken by the working group. Appendix states the primitive functions used in the proposed algorithms. These are session key generation, secret sharing schemes, asymmetric key generation, load and store information, encryption and decryption, and cryptographic token tasks.

2. RELATED WORK

As already exposed in section 1, this work is directly derived from the work by Martina[13] in the effort of establishing an open protocol to run embedded in HSMs. The protocols proposed by Martina in his work are used to create a liable environment to store our private keys for PKI usage. The main characteristics of these protocols are the existence of an internal PKI, where all administrative keys are subject of, the existence of groups to share the control over the roles being used, and the existence of two different roles: the administrators (security-officers) and the operators (users). Another important characteristic of this work is the establishment of policies when enabling keys to use, giving in this way a very strict control over their usage.

Although we see a growing concern about key escrow systems to give availability to (private) keys [16, 3] in a secure way, there is very few specific work relating this with the confinement idea of Hardware Security Modules, where we have a cryptographic perimeter.

Some companies have been working in addressing the problems of make backup copies of operational HSM. Ncipher Corporation, address the problem in their nShield Series[14], by designing what they call a Security World. A security world consists in a cryptographically wrapped environment that is saved in the host machine with all the content of the HSM. This security world will only be unwrapped in an authorised HSM, that belongs to the security world in question. This approach gives very little auditing data, and trusts blindly in the administrators' group.

Other approach includes token replication [5], which involves just copying the private keys to other tokens in a controlled way, leaving administrative and auditing data behind and without backup.

It can look simple, but it is not easy to make a backup of an HSM. First of all, rigid control is needed in the number of keys and their usage. This requirement must be taken into account when the backup procedure is carried out. A basic protocol might be for administrators to copy the internal data of the HSM and transfer it to new hardware. The problem with this and other protocols is that it is hard to control the backup and keep it safe. To improve simple protocols, the backup may remain encrypted by the administrators' public key. If this is done, only the administrators can install the backup onto new hardware. However, doing this means trusting the administrators blindly.

Blind trust issues are always not recommended by best security practices in PKI [4]. It has shown that it is advisable to have an independent group to audit the procedures performed by administrators. To achieve this, we should introduce the auditors role. A protocol should be designed to take into account two different kinds of user profiles — administrators who make the backups and auditors who trace the number of backups created, and the number of them made operational.

Each time the HSM is used, the auditors should analyse the logs produced. To carry this out properly, it is necessary to have previously established a PKI practice statement. This statement is a document which describes different ceremonies, including important ones such as when a backup is created, and when a backup HSM is made operational. We will discuss this issue in Section 6.

Other important and parallel work to be overviewed is the key management in group protocols and communications [15, 1]. Although not directly related, studies in this field show that it is often difficult to assure the behaviour of any involved party in group protocols. So, in this way, this is a reinforcement for the point of applying an auditing strategy and a ceremony design to trace these known problems.

3. BASIC ALGORITHMS

This paper deals with backup and audit procedures of an HSM. Some premises were taken into account to facilitate the description of these procedures. Moreover, there are two basic protocols that are used to create and to authenticate groups. This section lists these premises and protocols.

The premises are:

- HSM when initialised generates a key pair (kr_h, ku_h) and a self signed certificate (c_h). This certificate is used to uniquely identify the HSM and is trusted to issue certificates to the groups' members;
- each type of group has different data storage (DS). N.b. administrators' data storage (ADS) for administrators, auditors' data storage $(AudDS)$ for auditors and operators' data storage (ODS) for operators groups. Although they look like different pieces in the algorithm descriptions, when analysing and formalising the algorithms we consider them part of the owning principals;
- each group has an identifier (id) which uniquely refers to one group of the same type in an HSM. This is not true for administrator groups because an HSM has just one valid group at the same time;
- each group uses the secret sharing scheme[17, 2], owning a symmetric key (ks), that is split in n shares where at least m must be joined to recover the group's key. The thresholds must follow the rule: $1 \leq m \leq n$. A set of shares is represented by Ks;
- each member of each group receives a key pair (kr_i, ku_i) and a certificate (c_i), issued by HSM. Each member owns a cryptographic token (ct_{s_i}) to store this information for authentication;
- A share is assigned to each group's member. This share is encrypted by using the public key of the member's certificate. The shares are used for group authentication;
- every internal certificate is verified before its use;

The first common algorithm to be stated, after the premises presented, is the **createGroup()**. It works to create administrators, operators and auditors groups.

The **createGroup()** algorithm starts creating a session key ks for the group. In step 2, ks is split in n parts where at least m are required to recover it, resulting in Ks, a set of shares. The steps between 3 and 10 are executed n times, a time for each member of the group. Starting the loop, in step 4, a key pair is generated for the current member. Following, in step 5, the HSM loads from the host machine the name of the member id_i. This name will be used in step 6 as the member's distinguished name on the certificate c_i issued by the HSM. After that, the private key (kr_i), the certificate (c_i) and the HSM's certificate (c_h) are stored in his cryptographic token ct_{s_i}. In order to perform future authentication, the member's share is encrypted and, after, stored into the group's data storage (DS) with identification id and his certificate c_i. Finally, in steps 11 and 12, the values of m and n are stored and the group's session key is returned.

Algorithm createGroup(DS, id, m, n, c_h, kr_h) [1]

Creates a group based on secret sharing scheme, generating a key pair (kr_i and ku_i) and a certificate (c_i) for each member (s_i). The members' certificates are issued by HSM using its private key (kr_h) and information entered using the host machine (hm). The issued certificate and corresponding private key are stored into the cryptographic token (ct) belonging to each group's member. The group's session key (ks), which is split in n shares, is also returned as a result of the algorithm run.

```
1: $ks \leftarrow$ genKeySession()
2: $Ks \leftarrow$ splitSecret( $ks$, $m$, $n$ )
3: for $ks_i$ in $Ks$ do
4:     $(kr_i, ku_i) \leftarrow$ genKeyPair()
5:     $id_i \leftarrow$ load( $hm$, $s_i$ )
6:     $c_i \leftarrow$ genCert( $id_i$, $ku_i$, $c_h$, $kr_h$ )
7:     store( $ct_{s_i}$, $kr_i$, $c_i$, $c_h$ )
8:     $eks_i \leftarrow$ encrypt( $ks_i$, $c_i$ )
9:     store( $DS$, $id$, $c_i$, $eks_i$ )
10: end for
11: store( $DS$, $id$, $m$, $n$ )
12: return $ks$
```

Authenticating a group and recovering the group's session key (ks) is another common procedure used in the HSM. To authenticate, the algorithm **authenticateGroup** uses a nonce u in order to avoid Dolev-Yao's replay attack[6] as explained by Martina[13].

Algorithm authenticateGroup(DS[, id]) [1]

Authenticates a group identified by id in the data storage DS. The member's certificates are read from the cryptographic token ct. The process is, basically, to decrypt at least m pieces of shared secret, using members' private key, joining after to recover the group's session key ks.

```
1: $m \leftarrow$ load( $DS$[, $id$] )
2: for i = 1 to m do
3:     $c_i \leftarrow$ load( $ct_{s_i}$ )
4:     $eks_i \leftarrow$ load( $DS$[, $id$], $c_i$ )
5:     $u \leftarrow$ genKeySession()
6:     $eu \leftarrow$ encrypt( $u$, $c_i$ )
7:     $eks_u \leftarrow$ ctDecrypt($ct_{s_i}$, $eks_i$, $eu$ )
8:     $ks_i \leftarrow$ decrypt( $eks_u$, $u$ )
9: end for
10: $ks \leftarrow$ joinSecret( $Ks$ )
11: return $ks$
```

A group's authentication starts loading from DS the information of threshold m identified by id. This information is used for the iteration which will use enough shares to recover the group's session key. In step 3, the current member's certificate (c_i) is loaded from his cryptographic token. In step 4, the encrypted share belonging to the member is loaded from DS. In steps 5 and 6, the nonce u is generated and then encrypted by using the member's public key (which can be found in c_i). In step 7, the encrypted share and the

[1] All descriptions for principals and primitive functions are available in the Appendix under section A

nonce are sent the cryptographic token to be decrypted using the member's private key. The result of this operation will be the shared secret encrypted using u as a key session. In step 8, a piece of the group's secret is decrypted. After at least m shares be decrypted, ks can be recovered as stated in step 10 and, finally, the algorithm returns ks in step 11.

4. AUDITING SCHEME

To meet all the security requirements of the process, audit trails must be created, and a specific group of people, who will act as the HSM auditors, should be defined. Their main purpose is to collect auditing data from the HSM, enabling them to analyse the administrators group behaviour and to report any incident to the PKI's management team. This team should take the necessary procedures in order to guarantee the security of the whole PKI.

The auditors groups in an HSM should be considered as an inspection body. They always act with the purpose of assuring that other groups are acting and behaving as expected by their controlling authority (like PKI policy team).

The auditors group is also an important piece in the whole OpenHSM protocol, because it checks the auditing trail during the entire life of an HSM, and assures that no tampering, physical or logical, happens to the HSM while it is being used or kept in storage.

Our proposal states that the HSM should have at least one auditors group, but this does not limit the possibility of having more than one. Even the auditors group should not be trusted by other groups, and should be inspected by others. No destruction or changes to the auditors groups are planned, because a new group can easily be created if an existing one has been compromised, since they do not perform any vital cryptographic operations in the OpenHSM protocol.

The creation of an auditors group must be made just after the initialisation of the HSM, and the creation of the administrator groups, so it will be possible to audit everything that happens in the HSM life cycle. The **createAudGroup** algorithm has been proposed to handle the creation of auditors group.

```
Algorithm createAudGroup($id$, $m$, $n$) [1]
Creates an auditors group $id$ following threshold $n$ and
$m$, authenticating the administrators group before. A
key pair is created and assigned to it
1: $ks_{ad} \leftarrow$ authenticateGroup( $ADS$ )
2: $c_h$, $ekr_h \leftarrow$ load( $ADS$ )
3: $kr_h \leftarrow$ decrypt( $ekr_h$, $ks_{ad}$ )
4: $ks_{au} \leftarrow$ createGroup( $AudDS$, $m$, $n$, $c_h$, $kr_h$ )
5: $kr_{au}$, $ku_{au} \leftarrow$ genKeyPair()
6: $c_{au} \leftarrow$ genCert( $id$, $ku_{au}$, $c_h$, $kr_h$ )
7: $ekr_{au} \leftarrow$ encrypt( $kr_{au}$, $ks_{au}$ )
8: store( $AudDS$, $id$, $c_{au}$, $ekr_{au}$ )
9: return $c_{au}$
```

The **createAudGroup** algorithm starts when the administrators group enters the identifier, id, and the thresholds for the new group, m and n. In step 1, the administrators group is authenticated using **authenticateGroup()** algorithm, recovering the group's session key (ks_{ad}). From ADS, the certificate c_h and encrypted private key ekr_h of the HSM is loaded. The private key is decrypted in step 3 using ks_{ad}. In step 4, the auditors group is created and the group's key session ks_{au} is the result of it. In step 5, the key pair of the group is created. HSM, in step 6, issues the group's certificate. In step 7, the group's private key is encrypted by using its key session. After that, all required data describing the auditors group are stored into $AudDS$, such as the group's identifier (id), certificate (c_{au}) and encrypted private key ekr_{au}. Finally, the group's certificate is exported as a result of the algorithm.

In order to achieve the objectives of auditors groups the next algorithm is presented. It implements the protocol to export the HSM's logs. It enables the auditors to trace all operations that have occurred in the HSM. The algorithm, basically, authenticates the auditors group which is requesting the log and, using its private key, signs the log package. Afterwards, the signed log package is exported.

```
Algorithm exportLog( $id$ [, $rangeDate$] ) [1]
Allows an auditors groups, identified by $id$, to export the
signed HSM's logs. It's possible to specify a date range
$rangeDate$
1: $ks \leftarrow$ authGroup( $AudDS$, $id$ )
2: $ekr \leftarrow$ load( $AudDS$, $id$ )
3: $kr \leftarrow$ decrypt( $ekr$, $ks$ )
4: $L \leftarrow$ load( $LDS$, $rangeDate$ )
5: $sL \leftarrow$ sign( $L$, $kr$ )
6: return( $sL$ )
```

The **exportLog** algorithm starts when an auditors group informs its identifier id and, optionally, a date range, specifying a period for the logs. In step 1, the auditors group is authenticated and the group's session key ks is returned as result of it. The group's encrypted private key is loaded in step 2, and then, decrypted. In step 4, the specific block of log is selected and, therefore, signed in step 5 using kr. Finally, the signed log sL is exported.

5. BACKUP SCHEME

The following sections present diagrams and descriptions of the algorithms relating to the backup procedures, showing how to create a new operational HSM by installing the most recent copy of the data used by the old HSM into new hardware.

5.1 Preparing an HSM to be a backup unit

The first step, in having a secure backup system, is obtaining a spare hardware which is specifically configured to replace the existing HSMs when required. This hardware should be on standby, ready to become functional by receiving data previously copied from an operational HSM.

The preparation requires no initial information from any operating HSM, only the hardware in its factory state. The procedure is even possible when there are no operational HSMs. As soon as an HSM is prepared to be a recipient, it becomes possible to make a backup of the operational HSM. A backup of an operational HSM can only be made by using the certificate of the backup HSM. The administrator group must install this certificate into the operational HSM before it can create any backup file.

Once the backup HSM is a backup unit, it cannot be used for other activities. The backup itself is an encrypted copy of the data from an operational HSM. This encrypted copy is made by the administrators of the HSM from which should make new copies from time to time, as established in the

PKI practice statement. The physical security of the HSM is responsible for guaranteeing the backup private key, kr_{bkp}, enclosed.

The algorithm proposed to handle the preparation of an HSM to be a backup unit is as follows:

```
Algorithm prepareBkpHsm( $id_{bkp}$ ) [1]
Prepares an HSM to be a backup unit, generating a key
pair ($kr_{bkp}$ and $ku_{bkp}$) and, then, issuing a self-signed
certificate $c_{bkp}$ using it. The certificate is exported as
result of it
1: $kr_{bkp}, ku_{bkp} \leftarrow$ genKeyPair()
2: $c_{bkp} \leftarrow$ genSelfSignedCert( $kr_{bkp}, ku_{bkp}, id_{bkp}$ )
3: store( $BDS, kr_{bkp}, c_{bkp}$ )
4: return $c_{bkp}$
```

To run the **prepareBkpHsm** algorithm, the administrators group sends the information id_{bkp} to the new HSM, which is used for issuing the self-signed backup certificate c_{bkp}. Starting in step 1, the HSM generates a key pair kr_{bkp} and ku_{bkp}. This pair and the id_{bkp} information are the required data to issue the self signed certificate c_{bkp} in step 2. Therefore, kr_{bkp} and c_{bkp} are stored in the backup data storage (BDS), keeping kr_{bkp} inside the cryptographic perimeter of the HSM until it is necessary to recover a secure backup.

To enable the administrators to follow the next algorithm, the result of this algorithm is a certificate c_{bkp}, which contains the information sent by the administrators during the initial phases of this protocol run.

5.2 Importing a Backup Public Key Certificate

Figure 1 illustrates how a backup certificate is exported from a backup HSM and imported into an operational one. It is possible to use the same or different host machines to do so. By running the remote management system in the **Host**, the new hardware is commanded to work as a backup HSM. Upon receiving the command, the new hardware generates its own key pair and self signed certificate. Next, **Host** downloads the certificate and saves it as a file. By using the remote management system again, the certificate is uploaded into the operational HSM. After this, the operational HSM can make backups of its internal data. With this design, there is no limit to the amount of backup certificates that can be imported into an operational HSM. The backup, made in this way, can be installed in any one of the prepared backup HSM. Thus, the PKI management team could decide to have additional backup HSMs prepared, which can be used if one of them fails.

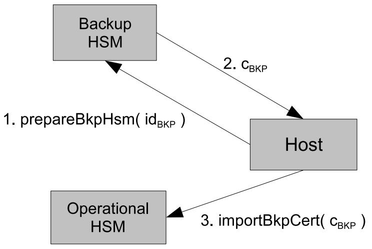

Figure 1: Copying the backup certificate.

Purely cryptographic means cannot assure the origin of a backup certificate because it is a normal certificate and could have been generated anywhere. So, here, it is essential to have a record and an auditing group to inspect it. Certificates that have been imported must be verified by the auditor group: importing a flawed backup certificate could lead to leakage of sensitive material.

To import a backup certificate into a operational HSM, it must be initialised and must have an administrators group beforehand. The algorithm to import a backup certificate into an operational HSM is as follows:

```
Algorithm importBkpCert( $c_{bkp}$ ) [1]
Imports a backup Certificate ($c_{bkp}$) which has been ex-
ported from a backup HSM. The authentication of ad-
ministrators group is required.
1: $ks \leftarrow$ authenticateGroup( $ADS$ )
2: $ekr_h \leftarrow$ load( $ADS$ )
3: $kr_h \leftarrow$ decrypt( $ekr_h, ks$ )
4: $sc_{bkp} \leftarrow$ sign( $c_{bkp}, kr_h$ )
5: store( $BDS, sc_{bkp}$ )
```

To start the **importBkpCert** algorithm, the administrators group must give the backup certificate c_{bkp} to the HSM. Following, the HSM authenticates the administrators group, in step 1, and release the HSM's private key kr_h over the steps 2 and 3 (this process is covered in section 4). Finally, c_{bkp} is resigned using kr_h and stored into backup data storage (BDS). The backup certificate is resigned to avoid any unauthorised inclusion in the BDS of unrelated backup HSM certificates.

5.3 Creating Backups

One of the most important operations in an HSM is to create secure and targeted backups. First, it must be stated that the backup creation process is a very delicate procedure, because it goes directly against normal goals, i.e., to keep sensitive cryptographic material inside the HSM protected perimeter. It is also something that must be done to ensure the continuity of the key life cycle, even in the case of a hardware fault, or because of tampering.

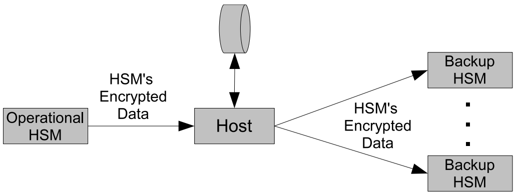

Figure 2: Creating backups.

Despite this, periodic copies of the HSM content must be made in order to enable the recovery of the whole environment just in case of a failure. Figure 2 illustrates this idea.

Data can only leave the internal perimeter of the HSM in an encrypted form; the design of the proposed backup process can therefore be described as ”secure” as stated in FIPS 140-2[8]. And it can also be labeled as ”targeted” because the Backup Package File BPF can only be opened in previously designated $HSMs$.

The administrators group must exist as at least one backup certificate must be imported to perform the backup of an

HSM. Additionally, for security reasons, it is necessary to have one or more auditors groups. The last requirement comes to not blindly trust to a single group (administrators group) the backup process.

The following algorithm is proposed to handle the generation of a security copy:

Algorithm bkpHsm() [1]

```
Creates a secure copy of an operational HSM just after the administrators group authentication. This copy can be recovered at any backup HSM whose backup certificate has already been imported into the operational HSM. This procedure copies the whole internal environment, excluding non-exportable ($NXD$'s) data.
1: $ks \leftarrow$ authenticateGroup( $ADS$ )
2: $ekr_h$, $c_h \leftarrow$ load( $ADS$ )
3: $kr_h \leftarrow$ decrypt( $ekr_h$, $ks$ )
4: store( $BPF$, load( $CTL$ ) )
5: store( $BPF$, load( $ADS$ ) )
6: store( $BPF$, load( $AudDS$ ) )
7: store( $BPF$, load( $ODS$ ) )
8: store( $BPF$, load( $KDS$ ) )
9: store( $BPF$, load( $BDS$ ) )
10: $C_{bkp} \leftarrow$ load( $BDS$ ) )
11: $ks_{bkp} \leftarrow$ genKeySession()
12: $eBPF \leftarrow$ encrypt( $BPF$, $ks_{bkp}$ )
13: $seBPK \leftarrow$ sign( $eBPK$, $kr_h$ )
14: for $c_{bkp_i}$ in $C_{bkp}$ do
15:     $eks_{bkp} \leftarrow$ encrypt( $ks_{bkp}$, $c_{bkp_i}$ )
16: end for
17: return $seBKP$, $EKs_{bkp}$
```

Once the backup process is triggered, the administrators group is authenticated using **authenticateGroup** algorithm, stated by step 1. In step 2, the HSM's certificate (c_h) and encrypted private key (ekr_h) are loaded from ADS, and then, the HSM's private key is released in step 3. The steps between 4 and 9 copy the current content of all data stored into backup package file (BPF), excluding non-exportable data storage (NXD) as explained by Martina[13]. Briefly, the NXD stores a part of the secret required for the administrators to perform any action over operators groups, then, once the backup is recover in a new HSM, each operators group must explicitly authorise the adminitrators to act again on its behalf. This gives assurance to the operators group in question that no backup of their keys can be recovered and become usable without their consent.

In step 10, the HSM loads the set of backup certificates (C_{bkp}) previously imported to define the target HSMs. The backup session key is generated in step 11 and used in step 12 for encrypting the BPF. A signature is made in step 13 from the encrypted backup package file for external checks. Therefore, in steps 14 to 16, the loop performs an encryption of the backup session key ks_{bkp} using each public key of backup HSM's certificates, creating as many encrypted session keys as backup certificates there exist. Finally, the list of encrypted backup session keys and the signed encrypted backup package file ($seBKP$) are exported as a result of the execution.

5.4 Recovering Backups

Recovery of HSM backups is only allowed in HSMs already prepared as backup HSM, i.e., where the algorithm from subsection 5.1 were run, and the exported backup certificate has been imported previously before the backup was made. The backup receiver unit has its private key kr_{bkp}, which it keeps protected against disclosure inside the HSM perimeter and will be used for decrypting the backup package file BPK, recovering the content of the HSM.

Recovering backup procedure authenticates administrators and an auditors group. Both authentications are only required whereas these procedures are considered administrative and necessary to keep the auditors aware of a new operational HSM.

The following algorithm is proposed to handle the security copy recovering:

Algorithm recoverBkp($seBPF$, $EK_{S_{bkp}}$, id_{audit}) [1]

```
Recovers a backup package file $BPF$, extracted from a signed encrypted package $seBPF$, in a previously prepared HSM, authenticating administrators group and an auditors group identified by $id_{audit}$. Both authentication are not really required to happen by cryptographic means
1: $kr_{bkp} \leftarrow$ load( $BDS$ )
2: $ks_{bkp} \leftarrow$ decrypt( $eks_{bkp}$, $kr_{bkp}$ )
3: $BPF \leftarrow$ decrypt( $seBPF$, $ks_{bkp}$ )
4: store( $CTL$, load( $BPF$ ) )
5: store( $ADS$, load( $BPF$ ) )
6: $ks_{adm} \leftarrow$ authenticateGroup( $ADS$ )
7: store( $AudDS$, load( $BPF$ ) )
8: $ks_{audit} \leftarrow$ authenticateGroup( $AudDS$, $id_{audit}$ )
9: store( $ODS$, load( $BPF$ ) )
10: store( $KDS$, load( $BPF$ ) )
11: store( $BDS$, load( $BPF$ ) )
```

The private key of the backup HSM (kr_{bkp}) is loaded from backup data storage BDS in step 1. In step 2, the session key ks_{bkp} used to encrypt the backup package file (BPF) is decrypted using kr_{bkp}. The BPF is decrypted in step 3 using ks_{bkp}, released previously. As explained before, this algorithm must authenticate the administrators group and a auditors group, so, in steps 4 and 5, the HSM reads CTL and ADS from BPF and, in step 6, authenticates the administrators group. Once again, in step 7, $AudDS$ is read from the BPF, enabling the HSM to authenticate the auditors group in step 8. Finally, in steps 9 to 11, ODS, KDS and BDS are read from BPF and a new instance of the HSM is made fully operational.

6. ADDITIONAL PROCEDURES

Security hardware by itself, in real situations, is not sufficient to guarantee the security of critical systems. Additional procedures are necessary, such as the testimony of witnesses. These procedures and testimonies should be previewed in a PKI practice statement as well as the description of all ceremonies. Important ceremonies include the creation of backups and the moment at which a backup HSM is made operational.

Despite very pertinent to PKI operation, this topic is oftenly forgotten. Recently, it was revisited by Ellison [7], who points its real importance to cover all outbound operations, normally done by human nodes, in any secure operation which involves mechanised nodes. Normally we do have to consider for security reasons, all operations in any security protocol or algorithm, even those needed to create the requisites to their operation.

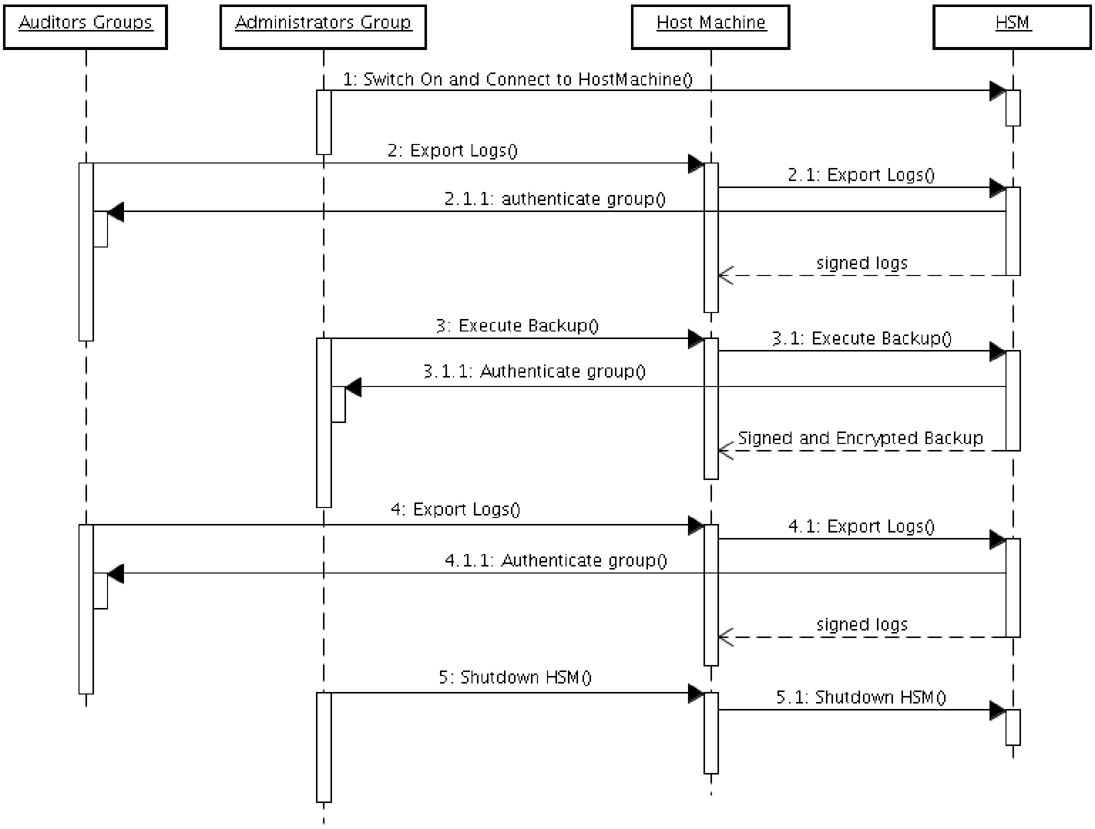

Figure 3: Creating backup file ceremony

In our case,as a prerequisite, every operation must be logged because these logs allow the auditors groups to create activity trails. If any problem occurs with the main HSM, its backup can be recovered. Its internal state will be just as at the backup procedure, without logs and procedures done subsequently. So, just a backup does not solve their problems. The auditors groups have an important role in this context: tracing operations, controlling the administrators, validating the operators groups' activities and others.

As shown in the last section, the creation of a backup involves many steps and, consequently, some critical points are met. A useful way of avoiding possible mistakes and maintaining the systems is to perform ceremonies. Figure 3 shows the ceremony to create backup files in the OpenHSM.

The full ceremony description for the OpenHSM usage is not the focus of this present paper, but this small fragment already shows us two important concerns:

- A log extraction in the beginning of the ceremony, to enable us to trust the HSM before starting the procedure, thus avoiding operation if any tamper tentative was tried since last operation;
- A log extration in the end of the ceremony, enabling us to assure that the operation succeeded and that nothing else happened during this operation.

Ceremonies should become essential resources for all main operational activities. Ceremony steps should be planned and should extract logs from the beginning and the end of the executed procedures, with registers of all the ceremony steps, involving as many people as possible and, ideally, recording the ceremony.

After a ceremony, a final report must be generated using all available data, which should also include ceremony steps' registers and logs. Finally, attendees present sign the report guaranteeing the veracity of the operation. Tests on the ceremony steps should be performed beforehand, using a parallel environment to try to avoid problems. These ceremonies will also allow the auditors groups to follow the system trail using logs.

Additionally, to avoid any possible compromise of the solution, some additional measures should be taken. If, for instance, an HSM becomes inoperative, the HSM should not be repaired. In this case, the HSM should be burnt, for instance, in a incinerator. This eliminates any possibility of keys being recovered.

Finally, the operational HSM and its backup HSM should be kept in different sites, avoiding the loss of both at the same incident.

7. FINAL CONSIDERATIONS

Open Hardware Security Module (OpenHSM) is a project carried out by the National Education and Research Network (RNP). RNP provides a Brazilian infrastructure of advanced networks for collaboration and communication in the fields of teaching and research. OpenHSM is a specialised HSM directed to PKI applications such as Certification and

Registration Authorities.

This paper described the deployment of the OpenHSM in a Public Key Infrastructure. Its main contribution was to improvements in the auditing system and backup operations for the OpenHSM project. It covered descriptions of algorithms used by the OpenHSM to enable it to perform secure backups and provide an audit trail. It also introduced a ceremony procedure to support the operation of such HSMs in a PKI deployment.

The next step in this project will be synchronising the internal clock to a trusted external time source. Then, the OpenHSM can also be used as a security time stamping server. Other future work can include the formal analysis of all protocols relating to the OpenHSM and the creation of all related ceremonies.

8. ACKNOWLEDGEMENTS

We would like to acknowledge the Brazilian Research Network (RNP) for the financial support in the GT ICPEDU workgroup, Brazilian Chamber of e-Commerce (Câmara e.net) for scholarship support in LabSEC, and also to Chris Sowton, Kristian Glass and Mark Reynolds for proof reading the paper.

9. REFERENCES

[1] Y. Amir, Y. Kim, C. Nita-Rotaru, and G. Tsudik. On the performance of group key agreement protocols. *ACM Trans. Inf. Syst. Secur.*, 7(3):457–488, 2004.

[2] G. R. Blakley. Safeguarding cryptographic keys. In *AFIPS 1979 National Computer Conference*, pages 313–317. AFIPS, 1979.

[3] J. Brown, J. M. G. Nieto, and C. Boyd. Efficient and secure self-escrowed public-key infrastructures. In *ASIACCS '07: Proceedings of the 2nd ACM symposium on Information, computer and communications security*, pages 284–294, New York, NY, USA, 2007. ACM.

[4] S. Chokhani, W. Ford, R. Sabett, C. Merrill, and S. Wu. Internet X.509 public key infrastructure certificate policy and certification practices framework. RFC 3647, 2003.

[5] Chrysalis-ITS. Luna pki hsm planning and integration guide. http://www.chrysalis-its.com, 2002.

[6] D. Dolev and A. C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.

[7] C. Ellison. Ceremony design and analysis. Cryptology ePrint Archive, Report 2007/399, 2007. http://eprint.iacr.org/.

[8] FIPS. Security requirements for cryptographic modules, FIPS PUB 140-2, 2002.

[9] IBM Coorporation. Ibm 4758 model 2 and 23 pci cryptographic cooprocessor manual. 2000.

[10] iVEA (Rainbow Technologies). Cryptoswift hsm user's guide. http://www.rainbow.com/, 2001.

[11] iVEA (Rainbow Technologies). Cryptoswift hsm user's guide. http://www.rainbow.com/, 2001.

[12] J. E. Martina. Project of a hardware security module focused on public key infrastructures and its applications. Master's thesis, Federal University of Santa Catarina, March 2005.

[13] J. E. Martina, T. C. S. Souza, and R. F. Custódio. OpenHSM: An open key life cycle protocol for public key infrastructure's hardware security modules. In *Fourth European PKI Workshop: Theory and Practice, EuroPKI 2007*, volume 4582 of *LNCS*, pages 220–235. Springer-Verlag Berlin Heidelberg, 2007.

[14] nCipher Corporation Ltd. Nshield user guide linux version 4.17.38. http://www.ncipher.com, 2004.

[15] S. Rafaeli and D. Hutchison. A survey of key management for secure group communication. *ACM Comput. Surv.*, 35(3):309–329, 2003.

[16] P. Samarati, M. K. Reiter, and S. Jajodia. An authorization model for a public key management service. *ACM Trans. Inf. Syst. Secur.*, 4(4):453–482, 2001.

[17] A. Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.

[18] T. C. S. Souza. Aplicações embarcadas para gerenciamento de chaves criptográficas. Technical report, Federal University of Santa Catarina, 2005.

APPENDIX

A. CONVENTIONS

The algorithms in this paper are subject to conventions from Table 2. This table presents the objects which store data from the HSM. This information is used to, with the protocols, manage the HSM. For instance, in the auditors data storage *AudDS* has details of auditors groups such as groups' name, groups' size and others.

Table 2: Principals of the Protocols

Principal	Description
ADS	Administrators Data Storage
AudDS	Auditors Data Storage
BDS	Backup Data Storage
BPF	Backup package file
CTL	Certificate Trust List
KDS	Keys Data Storage
LDS	Log Data Storage
NXD	Non-exportable data
ODS	Operators Data Storage

Additionally, these are the description of all primitive functions used, along with the algorithms, which have not been introduced yet. Basic functions are considered primitive when only a couple of operations are done and/or its name is self-explanatory:

ctDecrypt(*ct*, *edata*, *eu*) Uses the private key stored in the cryptographic token *ct* to decrypt data *edata* and nonce *eu*. Before returning the result, *data* is encrypted using nonce *u*.

decrypt(*edata*, *k*) Decrypts encrypted data *edata* using key *k* (symmetric or asymmetric).

encrypt(*data*, *k*) Encrypts *data* using key *k* (symmetric and asymmetric). If *k* is a certificate, its public key is used.

genCert(*id*, *ku*, c_{ca}, kr_{ca}) Certification authority *ca* generates a certificate with identification *id* and public key

ku. c_{ca} is the certification authority certificate and kr_{ca} is its private key.

joinSecret(Ks **)** Joins the shares Ks in order to reconstruct session key ks. Remember that the set Ks must have the minimum number of shares, as when split.

load (DS[**,** a)] Reads information from data storage DS (DS might represent the host machine). Optionally, it specifies one or more pieces of information to restrict the result, such as identifiers.

genKeySession() Generates a random session key.

genKeyPair() Generates an asymmetric key pair

genSelfSignedCert(kr**,** ku**,** id **)** Generates a self signed certificate with identification id and public key ku. kr is the private key.

sign($data$**,** kr **)** Signs $data$ using the private key kr.

splitSecret(ks**,** m**,** n **)** Splits, using secret sharing scheme, session key ks in n shares where at least m of them are required to reconstruct it.

store (DS**, ...)** Stores into data storage DS all other remaining parameters.