.

Public Key Infrastructure
ESnet Hardware Security Module Evaluation
September 2002

Overview

The Certificate Authority signing private key is its most important asset. Protecting this key from unauthorized use or disclosure is essential. Relying parties need assurances that this signing private key is secure. Hardware Security Modules (HSM) offer protection from certain kinds of attacks. We evaluated several vendors' products and selected the nCipher product for our PKI project. The issues and our evaluation experiences with these products are discussed.

Table of Contents

1 Introduction

Computational grid projects are, typically, composed of diverse individuals and organizations. These elements have different security environments and policies, which must be made to work together if the grid in question is to work at all. Projects and software developers have made a substantial investment in X.509-based public key infrastructure (PKI) technology in order to meet this requirement.

The "infrastructure" component of the X.509 PKI is the certificate and certificate management techniques. X.509 was designed with the expectation that a small number of certificate signers ("certificate authorities" or CA's) would sign certificates for large numbers of subscribers. Relying parties – those with a resource on offer – would trust the CA's to sign certificates under reasonable criteria (proven identity or proven membership, for example), and would accept these subscribers' keys because a trusted CA had signed it. These key pairs might then be used as part of a login transaction, including authentication and authorization steps.

The grid security paradigm separates the authentication and authorization steps. Grid CA's for the most part provide a kind of identity certification; the certificates are used to authenticate users. User authorization is performed in a separate step. Grid security depends on the integrity of these identity certificates, however, as these are used to unlock authorizations everywhere. The trustworthiness of these certificates is directly related to the trustworthiness of the CA itself. The CA presumably follows a set of policies that allow it to sign certification requests; but the CA, as a computer application, may be subject to various kinds of security attacks. The principal asset the CA must protect is its signing private key. If this key is disclosed or compromised (made available for use to unauthorized parties), the CA may sign certificates outside its policy and may help enable damage or loss to relying parties in various ways.

We undertook an investigation of hardware security modules (HSM's), which promise to substantially reduce the threat from some security attacks on the CA's private key. These are physical devices that control the use of the CA's signing private key. The CA (or other applications) access the device through PKCS #11 API's, a widely used standard for interfacing to cryptographic modules (hardware or software). The hardware storage module may limit some PKCS #11 functionality to improve the security of the key. The HSM may provide additional functionality, including key generation, cryptographic algorithm acceleration, and key life cycle and management functions.

This document discusses the technical details of **Chrysalis, nCipher** and **Rainbow** Hardware Security Module products.

We conducted this investigation in order to identify a product or products we could use in our Certificate Authority project in support of DOE Office of Sciences computational grids. The investment DOE made in our project justified the time and expense of bringing this CA up to a higher standard of trustworthiness. We were able to identify some usable products and selected nCipher for our use, and this is described below. The experience we went through may be of benefit to others, and any discussion provoked by this report may be of benefit to us.

1.1 Scope of Problem

The heart of the PKI is the Certificate Authority (CA).

A certificate authority signs objects, such as subscribers' public keys; these objects are then trusted by relying parties. What that trust means is a matter left to the Certificate Policy (CP) document, and the relying parties themselves (and perhaps other agreements or statutory limitations).

The CA Signing Key is used to sign all certificates that are issued by that particular CA, as well as certificate revocation lists (CRLs) and other objects. The very foundation of the PKI is based on the integrity of that root key. If the signing key is compromised, all certificates and CRLs within that specific CA service are suspect and the Certificate Authority's overall credibility is brought into serious question.

In addition, ESnet needs to run some of the CA services "on line". Without going into the details of the iPlanet CMS software, the registration manager and the certificate manager are meant to be on-line services, available to customers, subscribers, registration authority agents, and administrators at all times. The servers that support these functions are web servers adapted by the vendor to support these functions. While some of these services can be operated off-line (off the net), this is not an optimal configuration for the product, nor is it practical for ESnet or the grid projects we are serving. An on-line web server has some serious security issues:

> The web server may have configuration problems or internal defects that enable a compromise;
>
> The underlying operating system may similarly be vulnerable;
>
> Other support software or networking software in routine use in the LAN may become compromised, and allow an attacker to gain privileges on the online CA

The CA server may be subject to physical attack. This kind of attack applies to any server, whether it is online server or one kept off the network.

> The server may be stolen or destroyed;
>
> The disks may be physically removed and copied;
>
> Input devices may be "trojaned";
>
> Personnel who have physical access to the server may be able to gain privileges on the CA;
>
> Personnel who have physical access to server backups may be able to mount attacks on the CA private key

Any successful hacking attack on a CA puts the CA signing key in jeopardy. If the key is not encrypted, it is difficult not to conclude that the CA is compromised. Even if the key is encrypted, the possibility exists that a plaintext exposure could occur because of trojanned software or other techniques, or an attack on the key encryption itself. The CA itself is finished at this point, and can no longer be trusted to sign additional certificates or keys. Community expectations do not allow for a graceful check pointing of this failure, and the general expectation is that the CA and ALL of its certificates should be revoked.

The iPlanet Certificate Management System (CMS) 4.2 SP2 provides internal tokens for generating key-pairs and certificates, and these data are stored on a host machine in the form of *db* files (keyX.db and certX.db). CMS 4.2 SP2 also has provision to use PKCS#11 compatible external hardware tokens. In case of external tokens the key-pair are generated and stored in an external HSM device; the *db* file maintains a reference to the HSM device. See below for information on PKCS #11.

1.2 What can we do?

We cannot solve every "edge case" problem by technical means. For example, it is possible that a vendor's software may be internally compromised. The operating system software is not certified to be free from security risks, our building site is not designed to be impervious to attack by a determined adversary, and cynically we are all vulnerable to various forms of coercion or "rubber hose cryptography". HSM's may help us with some of the following scenarios:

A hacked "on line" CA server is not compromised. Depending on the privileges extended, an attacker may gain control of some CA services, but cannot gain access to the private key. The CA may be able to continue operation once the CA system is restored; it may be possible to assess the compromise issues with earlier certificates without re-initializing the entire PKI.

Privileges and access to the machine can be separated. We exercise finer grained control over some signing operations, or some configuration operations.

Physical access to the server does not elevate privileges. A system administrator, or a craft worker, may have complete physical access to the machine, but the HSM should eliminate this person's ability to gain control of the private key.

The product should defend itself from attacks and show evidence of attacks (whether physical or electronic); contrast to a disk file or bits in memory, which might be copied on some systems without any audit trail.

NIST wrote a standard, FIPS-140-1 (now –2), to cover some of these security issues, and vendors of interest build and test to this standard (see below). We chose to evaluate the ability of products to meet, by technical means, some requirements and support some scenarios typical of our working environment. HSM's do not completely satisfy these desires, nor they are a complete security solution allowing us to forego policy, access controls, and other security methods.

1.3 PKCS#11

PKCS#11 is a standard API (and library) used by iPlanet and other vendors of cryptographic products. "PKCS # 11" is a set of standards documents describing an API, called *Cryptoki*. PKCS#11 isolates an application from the details of the cryptographic device, thus the application must employ an interface for PKCS#11-compliant cryptographic devices (usually provided by the device vendor, analogous to a device driver). The HSM device must carefully limit the *cryptoki* capability exposed to prevent plaintext transmission of the key data to the computer host (as might be the mode of operation of an SSL accelerator board). The iPlanet CMS application must be capable of using the HSM vendor's *cryptoki* library. This means the vendor must provide a reasonable implementation, must provide tools to interface the vendor's library to CMS, and must have some understanding of the CMS product in order to support our implementation.

1.4 FIPS 140-1 levels

To protect the PKI, the keys must be stored and managed in a highly secure and tamper resistant hardware environment. Current “best practice” is to use FIPS 140-1 level 3 hardware security modules to help protect private keys from compromise. The FIPS 140 standard was written and is maintained by NIST, and they certify testing authorities who test and rate HSM vendor products.

2 CA Patterns

We identified a number of patterns, or operational scenarios, that were troublesome to us in various ways. HSM evaluation included examination of the effect of the HSM on CA's operated in these ways, or included evaluations of the positive and negative consequences an HSM had on these scenarios.

2.1 Roles

We identified three parties in our operation that influenced, and could control, how a CA would operate.

Systems administrators own and maintain all the computer systems, hardware and software, in ESnet. All day-to-day operations and activities are dependent on their work. They have physical access to all machines. It might be better put that the technical services group decides how its personnel manages the ESnet computer systems. This group also manages internal network resources.

PKI Project personnel developed the architecture, installed the CA software, configured and managed the CA and HSM, and have advised network and systems personnel on details related to the PKI project computer systems. Project members are also responsible for day-to-day activities of the CA services. In other words, this party knows too much! They know the ins and outs of the products.

ESnet management is responsible for all our projects and personnel. As such they reap the benefit or the blame for project outcomes. However, management does not perform day-to-day service activities, does not delve into the depths of the software suites, or do hands-on systems configurations (or why would they need the rest of us).

(There is a fourth category in ESnet, which includes sophisticated staff members or guests who might conceivably gain access to our CA's through technical expertise or circumstance; but we intend that our security plan and various access control points will deal with this consideration.)

We felt that we could make the most straightforward case for trustworthiness of our CA's by making access and control of the CA's as balanced as possible between the three interested parties, systems administrators, PKI project, and ESnet management. In other words, we want CA actions to reflect ESnet policy, minimizing the possibility that some kind of social engineering or insider attack might alter the CA's behavior. Accordingly we look for patterns that reduce the access of the systems administrators and mitigate the capability of the knowledge that project members have. These patterns include:

- Key splitting or sharing between the parties
- Assigning roles to parties, such as
    - Giving ownership of tokens (or "smart cards") to one party
    - Giving decryption of tokens to a second party
    - Given power of creation of tokens to a third party
- Assigning identities and specific roles to individual members
- (and others, outside the scope of this document)

2.2 Types of Certificate Authorities

We anticipate creating several different kinds of certificate authorities, which may each require different patterns and security policies. Our list is not complete – there are many other configurations, and others may be of interest to ESnet at a later date. Some configurations under consideration today include:

An off-line, off-net root (self-signed) certificate authority, which may not be used very often, and which may be taken apart or even stored off-site for long periods. While never on the network, the physical integrity of the components of this system is a serious issue. A root CA of this type typically signs subordinate CA's. If this root CA were stolen and duplicated, rogue subordinate CA's could be created. These CA's could hijack transitive trust expectations. These rogue CA's could even be used to impersonate legitimate subordinates. This root CA system would probably be secured in a safe in a physically secure location during its inactive periods.

An off network subordinate CA. This CA signs end-entity certificates, but is disconnected from the network for security reasons. In this case, operators would have ready access to this system. The physical security issues are greater for this configuration, since the system would be more accessible to an attacker interested in hijacking keys, compromising the system, or "trojaning" the hardware or software. The system may need to be in an office environment where physical access control is difficult (more difficult than a computer server room).

An on-line subordinate CA, which supports on-demand signing of certificate signing requests. One form of this configuration is the typical iPlanet CMS configuration, where customers use a web front-end to post signing requests directly to the CA and signing agents use an authenticating front-end to order the CA to sign these requests. Another form might be an on-line credential issuing appliance such as Globus' proposed CAS service: a subscriber logs in to the service with an accepted credential, and then the server issues a Globus "proxy certificate" including some built-in authorization extensions. These servers have physical integrity issues comparable to any system in a computing facility. The server room may have physical access controls, but nevertheless many people may have incidental or occasional access to such a facility, including systems administrators, selected staff, vendors, craft workers, and fire or public safety personnel. In addition these servers will have long-term presence on the network, and may employ a web server or other well-known services as part of the customer interface. We can reasonably expect that these services will be subject to attack and that vulnerabilities may exist that would allow an attacker to gain access or control of the host.

An on-line registration manager (RM). This is a special service of iPlanet CMS that allows the signing authorization to be separated from the actual signing of the certificate. In the iPlanet configuration, the RM is a special trusted entity, but does not actually sign the certificate itself. The issues are nearly identical to those of the online CA. The RM keys do not appear in any relying party's trust chain, so replacing them is not as serious an operational problem as replacing a compromised CA's key.

In each of these cases we are also looking for opportunities to change the balance of control and increase the integrity of both the signing key and the key's uses. These include:

Shared control of key generation / copying / destruction

Shared access (*m* of *n*) to signing key usage operations

Identity-based access to signing key usage

Our on-line configurations are meant to run unattended for long periods. We do not fully understand the trust and security implications of running a public CA online for a long period. We are exploring different scenarios for controlling the problems that this seems to imply. Can we support a scenario where an online appliance is capable of booting up automatically and enabling access to the CA signing key? Can we support a semi-automatic configuration, where operators can enable/disable the CA signing key by providing/removing a special token? (Most HSM products have a smart card component that could serve.) Can we provide staged or time-based access: orders (signing requests, approvals) can be taken at any time, but fulfillment (signing) only occurs in the presence of operators (regular business hours).

An on-line CA might be subject to network-based attacks, and the host, or even the CA software, might be compromised. Is a key managed by an HSM a hardware object, or a software component? Replacing a CA's key pairs and all its certificates in the event of any suspected or proved compromise is a severe blow to a PKI, and serious questions might be raised about the cost and efficiency (or even the efficacy) of this practice in a grid environment. If the key pairs may be classified as hardware objects, there may be many scenarios where compromise of a CA does not automatically result in replacement of the PKI. The grid environments with which are involved are loosely coupled enterprises where distributed management is the rule; and the middleware has been tuned to appeal to relying parties instead of (or at the expense of) central management and coordination. One of the side effects of grid configuration is only minimal ability to revoke certificates reliably, and this applies as much to CA certificates as to end entity certificates (PGP has an analogous problem that might shed some light on our situation).

2.3 Destroying a CA

Typically, a software – based cryptographic module is used to store CA signing key pairs. (Example: iPlanet CMS uses *.db* files as mentioned elsewhere, *openssl* uses a configuration file and specially encoded key files.) Backup and disaster recovery policies usually result in duplication of the private key (either plaintext or encrypted), and duplication of the private key's security key, if used. When a certificate authority ceases operation, its signing key must be removed from service. To insure this key is never used again, at least during whatever validity remains to its signing certificate, the corresponding private key could be destroyed.   Assurance that the private key's backup copies and security keys have been totally disabled and destroyed is quite difficult to achieve.   This situation is particularly difficult with self-signed CA certificates with very long lifetimes.   The uncertainty period, from cessation of operation to expiration of the signing certificate, might be very long.

One paradoxical result might be to chose to keep the old CA keys but in an inactive state, in order to defend against a possible duplication.   The CA operator must assume the risk that this inactive CA might also be stolen and used.

When a CA is finished, can we do a better job assuring everyone involved that it is really gone for good?   We think the HSM, as a physical device, may reduce the opportunity for unauditable or neglected duplications of the private key, and eliminating the HSM or its tokens might be a way of guaranteeing the CA is incapable of signing any more certificates. (See the previous discussion for another viewpoint.)

3 ESnet Evaluation and Choice

3.1 Choice

ESnet chose the nCipher product for its PKI project.

3.2 Requirements

3.2.1 Security

System security compromise does not endanger CA's private key (defense against electronic or physical theft)

FIPS 140 level 2 or better

Device-generated keys (not host-generated)

No access to plaintext private key, even by system administrators

2048 bit RSA key size

3.2.2 Operational

Online CA

iPlanet CMS 4.2 SP2 support

PKCS #11

Solaris 8 (sparc) platform

Acceptable to ESnet system administrators

Adequate vendor support

3.3 Evaluation results

There are only a limited number of HSM vendors on the market. Our requirements eliminated some products.

The 2048 bit RSA key size was a requirement brought in by our collaboration with European Data Grid (EDG) CA's. This grid requires a 2048 bit key size for certificate authority signing key pairs. This is a reasonable key size for the next few years.

iPlanet CMS and the Solaris Sparc platform are requirements based on our infrastructure and other project commitments. The CMS product uses PKCS #11 API's to manage the interface to cryptographic modules, such as hardware storage devices. We also needed to make sure our choice was acceptable to ESnet system administrators, who retain control over our systems' day-to-day operations and are responsible for service and maintenance issues. Because of our hardware support requirements the IBM product could not be considered. Baltimore, or possibly a related company "AEP Systems", offers an HSM, but this product appears to be tied to Baltimore's UniCert CA (we were not sure that this product was available at the time of our evaluation).

The Rainbow product was unable to meet our 2048 bit key size requirement. However, the simplicity and cost of the unit are attractive, and we felt it might be useful for components that did not require this key size, such as a registration manager; so we continued the evaluation in order to determine whether it would work with our CA product.

Two products, nCipher and Chrysalis' Luna, met all of our requirements. Both would meet our needs (see the extensive discussion in the evaluation sections in the appendix). Despite the great differences between the products, it is difficult to choose between them. The Luna product is simpler to explain: the private key is managed in a separate device – *period*. NCipher has a more complicated story: its security world, where keys reside in encrypted form on hosts, and careful design downloads these encrypted blobs to the cryptographic module in a secure fashion.

The Luna product, because of its independence, completely separates all the authorization functions from the application. nCipher is partly integrated into CMS and relies on CMS' UI to interface the smart card to the nShield board.

3.3.1 Management and operational considerations

Management and operational considerations distinguish the two products. The nCipher product has fewer components, and one of them (the smart cards) is available from a well-known third party. nCipher also has considerably more configuration choices. The nCipher product security is completely dependent on these smart cards; if the nShield board is stolen, or the computer system is stolen, the thief will not be able to use any of the keys without access to the appropriate card set. On the other hand, CA operations can resume with any available nShield board, and a backup of key data and CMS data. Disaster recovery – recovery from a fire or other system loss – is simple.

Luna backs up keys by cloning tokens; additional tokens are quite expensive. Providing enough redundancy to survive a fire, token failure and similar contingencies would be more costly. The Luna hardware modules are all custom or proprietary, from the PED keys to the hardware tokens to the PCI board. This leads us to consider the effect on our operations should the hardware vendor change business plans (or go out of business). It would be much more difficult for a small operation like ours to continue if the vendor's proprietary hardware was no longer available.

3.3.2 On Line CA

We require the ability to run an on-line CA. This requirement is probably also of interest to others in our community, who are considering authentication gateways and authorization services of various kinds. We are not sure if we can avoid 24/7 uptime requirements in the future, as more dependence on CA services (validity/revocation checking) appears. Maintenance and unscheduled reboots need to be accommodated somehow. There are cases where we would prefer the online CA to automatically reboot and restart the CA services – in these scenarios we will trust that our physical access controls prevent unauthorized access to this service, and allow the token to remain logged in. nCipher has a slight advantage here, in that a card can remain in the smart card reader, and the system , security world, and CMS can be configured to log in automatically on reboot. Based on our current understanding of the product, we do not think that Luna offers this flexibility. Luna, on the other hand, enables the cryptographic module independently, so the system can reboot without affecting the state of the module; applications are not involved in enabling the device (see appendix for our experience with this). Luna will not recover from a scenario where power is lost to the whole rack or area where the devices are stored. It's not a likely scenario, but it gives a slight operational edge to nCipher.

3.3.3 Security considerations

It is axiomatically accepted that simpler is better in security, and for the most part Luna leads here. The PED keys' functions are distinguished by color (although one of these function is difficult to understand). The private key never appears on the host system; instead separate devices manage all operations. Keys are kept on physical tokens. PED PIN entry is through a hand held separate device, not through the keyboard. On the other hand, an internal hardware random number generator generates nCipher keys. The signing key is stored on disk wrapped by a 3DES key generated by the nCipher device – operators do not have direct access to the plaintext key, or to the 3DES key that unlocks it, it is locked to a smart card set. Smart card passwords can be long, complex pass phrases (or whatever suits the owner), but pass phrase entry is through the system keyboard. However, sniffer capture of this pass phrase doesn't buy any particular access; the card must still be obtained as well. In the nCipher system, control of the smart cards is the crucial problem, which seems to us like a reasonable one at this point. The additional security offered by Luna does not offer enough value and also adds additional costs, in our environment.

3.3.4 Flexibility

If for some reason you don't like the key-pair OR the certificate, you can delete those objects with Rainbow and nCipher. But initialization is the only option Chrysalis provides to deal with the content of a token. This is an awkward and dangerous way of dealing with deletion.

Chrysalis and Rainbow have some limitations on the number of objects their token could support. There is no practical limit for the nCipher device. It's not clear, in our application, whether the limits are important.

3.3.5 Performance

nCipher nShield is also a cryptographic accelerator (see features list). Luna performance statistics weren't available from the vendor, and the vendor sells crypto accelerators as another add-on product. It was clear from our use that the Luna didn't improve the cryptographic performance of CMS at all – perhaps reduced it. On the other hand improvement was noticeable with nShield. We could recover some of the cost of nShield in less expensive hardware platforms for the certificate authority and related products. This is a minor consideration in our particular application, and debatable due to the large impact of CMS' JVM on host performance. This consideration might be more important in some future development or use in a more demanding on-line certificate authority service.

3.3.6 ESnet review

NCipher received a better review by our engineering staff. The relative complexity of operations with Chrysalis and the custom input devices were not as well received as the simple nCipher card readers.

In order to evaluate the Chrysalis product, we had to buy a unit for evaluation. It wasn't possible to acquire a loaner or demonstration module. We were not too happy with this arrangement. Other vendors were willing to offer evaluation units (not always their top level products).

3.3.7 Summary

Both nCipher and Chrysalis Luna are reasonable choices. They both met our requirements. We have some concerns about both companies, as they are small and their market is obviously a very select one. We remain interested in both. nCipher was chosen because its operational characteristics and flexibility were felt to be a better match to our working environment at this time.

3.4 Pattern evaluations

HSM's cannot fulfill every feature of every pattern and scenario described, but all the candidate products support many of the necessary features. All implement roles in various ways (FIPS-140 requirement), but these roles are not an exact match for our organizational roles. HSM's implement various controls and checkpoints that support the various kinds of CA's described. Both vendors' products result in a private key that is better described as a hardware device than a software object. Many technical and operational issues remain – no product offers a perfect solution for the online CA problem, but we have improved the trustworthiness of our current deployment. We have more confidence that we can remove an old CA from service with assurance that it cannot be recreated. More work on these issues and protocol with the vendors is needed. Some of the details pertaining to these patterns appears in the following sections, where more in-depth discussion of the vendor products and evaluations are to be found.

Vendor Evaluations

We evaluated three products: the Chrysalis Luna CA^3, Rainbow CryptoSWIFT, and nCipher Security World (the latter is the architecture name, but the product must be analyzed as part of a whole system). Each section describes the hardware and software components, provides a picture of the hardware components, provides a diagram of the architecture, and summarizes evaluation experiences with a pros and cons list. The full evaluation notes for the three products are in the appendix. The product manuals must be consulted for additional details, much is left out here.

Table 1 Features Summary

Product / Features	Rainbow CryptoSwift	Chrysalis Luna CA[3]	nCipher Security World
FIPS 140-1 Level 3 compliance	Yes	Yes	Yes
Host Interface	PCI	PCI	SCSI & PCI
Reliable with iPlanet CMS 4.2 service pack 2	No	Yes	Yes
Key-Pair Random Number Generation (RNG)	No Data	Based on Annex C of ANSI X9.17 (Pseudo Random Number Generator)	HSM nCipher (hardware RNG)
Hardware used to unlock the private key	iKey (USB port)	PED keys	Smart Card
Utility	kmu – GUI based	enabler – command line utility	Keysafe – GUI based
***m* of *n* Authentication.**	2 of 2	*m* of *n* (*m,n* <=16)	*k* of *n* (*k,n*<=64)

PKI Algorithm & Key Size	RSA (384-1024)	RSA (512-4096)
		DSA(512-1024)
			RSA(512-4096)
			DSA(512-1024)

Performance **(Vendor specifications)**	Random number Generation – 18000 bytes/sec RSA private key with CRT performance- 4.95ms/operation at 1024-bit	No data	(1024 Key Size) Key Pair Generation- 7/sec Sign New Certificates- 394/sec Validate Signed Request – 1543/sec Complete SSL handshake – 393/sec
Device Storage Capacity	No data	256 KB Volatile Memory 1 MB non-volatile memory	No data
Customer Reference	N.A	Good/Satisfactory	Good

4 Chrysalis Luna CA^3

The Chrysalis-ITS Luna CA^3 safeguards digital key management by protecting keys in a hardware device and offloads sensitive operations from the CA server: key generation, verification, storage, signing and deletion remain safe from compromise. See Luna CA^3 Components on page 52.

4.1 Luna CA3 Components

The Luna CA^3 product consists of rugged hardware components designed with security considerations in mind. The private key is generated on the custom token, and never appears on the host. The signing key is essentially identical to this token, and back ups of the private key are achieved by using special hardware and security components to "clone" one token to another.

4.1.1 Hardware Tokens

A proprietary hardware device that implements cryptographic functions and stores keys, for use by suitably capable application programs.

4.1.2 Token reader / Dock reader

A special purpose card-reader device intended only for use with Luna Tokens. Controlled by a proprietary PCI-bus card, Luna DOCK enhances the speed of token interactions and permits chaining of multiple units.

4.1.3 Pin Entry Device (PED)

A Personal Identification Number (PIN) Entry Device for use with Luna token products. Requires the Luna DOCK Card Reader.

4.1.4 Pin Entry Device Keys

An electronically reprogrammable device in the shape of a physical key (the electrical contacts are in the key notches), used as a portable access-control device in compatible systems. Different keys for different functions.

4.1.5 Dock Reader Controller

This is a PCI interface card, which controls the DOCK reader.

4.2 Utilities/Tools

4.2.1 Enabler application

Enabler, as its name suggests, enables other application programs and Luna token products to work together. iPlanet CMS 4.2 sp2 uses the Luna token as a secure generator and repository for keys, certificates and other data objects that are created and manipulated during the CMS configuration. The enabler application has to initialize and activate the token for use by iPlanet CMS 4.2 sp2.

Refer to the Luna PKI HSM installation Guide for detailed information.

See Luna CA Architecture on page 53.

4.3 Hardware Management & Developer Experience

4.3.1 **Pros**

Installation and CMS configuration are relatively simple.

Private key is never copied from the Luna CA^3 token to the host in any form.

"*enable*r" key management application gives complete control over the token.

Token authorization is controlled by Luna CA^3 devices (PIN, PED keys, and PED). External application like iPlanet CMS don't need to manage this information. This allows an *m* of *n* authorization before enabling iPlanet CMS access to the token.

Default Timeout: The HSM token will lock if expected operations don't complete in a timely fashion. This prevents the unit from being inadvertently left in some maintenance state.

4.3.2 **Cons**

Complex PED Key management (SO key, User Key, Domain Key, Init Key & *m* of *n* key).

Key management is not flexible. Initialization is the only way to remove the content from the Luna Token.

Limitation on token's storage capacity. (256 KB Volatile Memory)

Every CMS instance backup requires its own Luna token, which is expensive.

4.4 References

Engineer from USPS: Good experience with product on Solaris, project completed and no longer in use

VeriSign: VeriSign has a long history with this company, and is a Chrysalis shop.

5 Rainbow CryptoSWIFT

The CryptoSwift HSM, hardware security module, is a FIPS 140-1, Level 3 compliant secure storage device. The CryptoSwift HSM uses a secure hardware area that is protected against external tampering, and any sensitive information that is transferred over the card's PCI bus is encrypted so that it never is exposed as plain text. This device has a simpler workflow than the other products; security of the product is highly dependent on securing the iKey components. See Rainbow Components on page 54.

5.1 CryptoSwift Hardware Components

5.1.1 CryptoSwift card

CyptoSwift HSM is a proprietary PCI card. It has a serial port, a Universal Serial Bus (USB) port, and LED for indicating status. The USB port is used for authentication and the initialization of the cryptographic authentication token (a Rainbow iKey).

5.1.2 iKey

The Rainbow Technologies iKey is a smaller, USB-based cryptographic token that is used as part of the authentication process. There are separate iKeys for two types of operator roles, one for the Security Officer role (SO) and another for the User role.

5.2 Utilities/Tools

5.2.1 KMU application (key management utility)

The KMU will allow users to interface with CryptoSwift and CryptoSwift HSM cards, and other non-CryptoSwift cards that comply with the PKCS#11, v. 2.10, specification.

Refer to the CryptoSwift HSM user guide for detailed information.

See Rainbow Architecture on page 55.

5.3 Hardware Management & Developer Experience

5.3.1 **Pros**

Simple install and configuration procedure.

Simple Key management (SO key, User Key and 2 factor authentication Key) USB compatible iKey for secure authentication.

5.3.2 **Cons**

Compatibility issue with iPlanet CMS 4.2 sp2.

Limited support for *m* of *n* authorization in Key Management

Limited PKI algorithm and Key Size support.

Lifetime of the USB compatible iKey is questionable.

5.4 References

None provided.

6 nCipher Security World

nCipher has developed a construct, called a "security world", that provides secure life-cycle management for keys. These procedures and protocols include the generation, distribution, use, storage, destruction, and optional archiving and disaster recovery of cryptographic keys. The nShield board does most of the cryptographic work, but security of the service is highly dependent on the security of the smart cards.

6.1 Security World Components

See nCipher Components 56 for several views of the hardware modules described in this section.

6.1.1 One or more nShield modules

The nShield module is a proprietary PCI or SCSI card where all cryptographic functions take place.

6.1.2 Smart cards and smart card reader

Set of smart cards used to authorize cryptographic functions. Smart cards are grouped into two sets, the administrative card set (ACS), which manages the security world, and the operator card set (OCS), which enables applications to use the cryptographic services (see). The smart card reader is a proprietary reader attached (cabled) to the nShield module. It uses the host computer's keyboard for input. The smart cards are Gem Plus cards.

6.1.3 Encrypted data stored on host computer

The nCipher security world uses the host's file system for long-term storage of keys. Keys are stored in Triple DES (3DES) format and never appear in plain text on the host computer; keys are converted from encrypted form to a usable form (plaintext) after they are downloaded to the nShield module and unlocked by the operater card set (OCS) smart card(s).

6.2 Utilities/Tools

The nCipher Security World has a large number of utilities and tools that support operations. Some of the typical tools used were:

nfast – the hardware server, a daemon started on bootup to represent the hardware device.

createocs – creates the operator card set (OCS)

new-world – creates the security world, and the administrative card set (ACS)

Refer to the nShield User guide for more information.

See nCipher Architecture on page 57.

6.3 Hardware Management & Developer Experience

6.3.1 **Pros**

Key Management tools.

Key Backup and Key Restore process is simple to follow.

Device Capacity, which is cost effective

Singe device can be used for multiple CMS instances on a same host.

Optimal hardware to maintain. (smart cards, card reader & PCI device)

Smart cards are industry standard devices (Gem Plus)

Security World backup is just like any other software backup.

6.3.2 **Cons**

Installation and Configuration is relatively complex.

Though it provides *k* of *n* multifactor token authentication, iPlanet CMS 4.2 sp2 could not handle *k* > 1. This could be solved using 'with-nfast' utility.

Lifetime of the smart card (gold contact) is questionable. Solution could be making a sufficient number of ACS and OCS cards.

Smart card reader is proprietary. This is probably unavoidable, due to the security requirements placed on this device and software.

6.4 References

StrongAuth (correspondent also formerly from Sun Microsystems). Good experience. Engineer used a mixed strategy at old site: Chrysalis for the root, nCipher for subordinate CA's; but chose nCipher for all levels in the new site. Pointed out less expensive (FIPS-140 level 2) modules could be used for some subordinate CA's and remote RA hosts.

7 Bibliography

7.1 Vendor

These references may require contacting the vendor.

"Luna® PKI HSM Installation Guide", Chrysalis-ITS, http://www.chrysalis-its.com, 2001

"Luna® PKI HSM Planning & Integration Guide", Chrysalis-ITS, http://www.chrysalis-its.com, 2002.

"CryptoSwift HSM User's Guide", iVEA (Rainbow Technologies), http://www.rainbow.com/, 2001

"CryptoSwift Cryptoki 3.xx Reference Guide", iVEA (Rainbow Technologies), http://www.rainbow.com/, 2001

"Getting Started Guide", version 4.1.37, nCipher, http://www.ncipher.com, 20 Mar 2002

"nShield User Guide Solaris", version 4.6.15 nCipher, http://www.ncipher.com, 29 Apr 2002

"Globus 2.2 Admin Guide", The Globus Project, http://www.globus.org/gt2.2/admin/index.html, 2002

"iPlanet Certificate Management System Installation and Setup Guide", CMS 4.2 SP2, iPlanet/Sun Microsystems, http://docs.sun.com/source/816-5541-10/index.html, 2001

7.2 Standards

"SECURITY REQUIREMENTS FOR CRYPTOGRAPHIC MODULES", FEDERAL INFORMATION PROCESSING STANDARDS PUBLICATION, FIPS PUB 140-2, National Institute of Standards and Technology (NIST), http://csrc.nist.gov/publications/fips/fips140-2/fips1402.pdf, May 25, 2001 [Most of our products certified under 140-1]

"FIPS 140-1 and FIPS 140-2 Cryptographic Modules Validation List", National Institute of Standards and Technology (NIST), http://csrc.nist.gov/cryptval/140-1/1401val.htm, 2002 [Changes regularly]

"PKCS #11 - Cryptographic Token Interface Standard", v2.11, RSA Laboratories, http://www.rsasecurity.com/rsalabs/pkcs/pkcs-11/index.html, November 2001 [website has additional information besides this document]

7.3 Miscellaneous

Schiller, Jeff, "Protecting a Private Key in a CA Context", Internet 2 Middleware, http://middleware.internet2.edu/hepki-tag/private-key-prot.html, 2000 [A useful discussion of the issues and patterns]

Appendix A Mike’s HSM Evaluation Experience

A.1 Rainbow CryptoSwift

A.1.1 Hardware Installation

A.1.2 Software Installation

I was partly involved in the initial hardware (token) and software installation. This was a simple matter of installing the PCI card in a slot, and installing software packages from the CD.

A.1.3 Device Configuration

The device was quite easy to test and configure; the manual covers this completely. The vendor provides a very nice X11 UI with a java backend that manages logins, admin and user roles, key lookup, and other functions. It was necessary to install the appropriate JVM (eg JRE 1.4) on the server for this function.

The original units could create up to 1024 bit keys.

After some time (6 weeks or so), we were able to get a driver update, which helped the CMS install process along (see CMS Configuration), and also allowed the device to create and manage 2048 bit keys.

A.1.4 CMS Configuration

CMS installation is, in its initial phases, identical to installation on a server without HSM. After the initial set of servers are in place, the crypto engine and the PKCS#11 library for it must be enabled. The instructions for this are in the Cryptoswift manual. PKCS #11 instructions were somewhat different than those for other HSM’s.

I was then able to start the CMS configuration. I had the CMS create its self-signed cert and SSL server cert, and was able to start the CMS server. Initially, those keys didn’t appear in the module’s operator console, but after restarting / refreshing, the keys were listed.

I was able to create the first administrator cert, but I was unable to use it. When I would connect to the service’s agent port, I would get this error:

```
"The server could not verify your identity"
```

The CMS's log file says

```
12164.https:conn-24 - [30/May/2002:15:57:03 GMT-08:00] [4]
[0] Accepted
client connection
12164.https:conn-24 - [30/May/2002:15:57:03 GMT-08:00] [4]
[0] Initialized
connection from fionn.es.net/198.128.1.30 user-agent: null
12164.https:conn-24 - [30/May/2002:15:57:13 GMT-08:00] [4]
[1] File Access denied: Failed to Authenticate -
java.net.SocketException: Error in SSL handshake (Peers
certificate has an invalid signature.)
```

Customer support was unable to help me – shortly after this, we lost contact with the company. A few weeks later we learned our sales rep had left the company, when a new sales rep contacted me.

These events had a couple of unusual side effects. Because the error message suggests something is wrong, and the most likely cause of it in non-HSM environments would be a bad password, we wound up "killing" the operator I-Key by trying to log in with too many bad passwords. This led me to explore Rainbow's token reset software. This software resets the USB-based tokens that Rainbow provides (they function like smart cards). It turned out that the associated reset utilities only work on certain old Windows operating systems, like Windows 98 or NT 4.0. None of these are currently supported at ESnet (or exist for that matter). Windows 2000 and Windows XP weren't able to run these applications. I was only able to reset these tokens because I have machines running these old operating systems at home.

The above events occurred around 01 June. I gave this unit another try in mid September. By this time we had had some progress on clearing up this problem, and on getting the driver or firmware updated to handle creation and management of 2048 bit keys.

When problems occurred in CMS operation (see below), I tried rebuilding the service using the same admin secmod build that the other HSM's used, without any change in behavior.

I had to re-init the card, and for this I had to use the administrator I-Key. This left the card in a state where it wanted to be reinitialized as soon as it was examined again, which I did. Curiously, the I-Keys seemed to remember and require their old passwords – I thought that re-initialization would clear them, but apparently they are tied to the I-Key device. There is a nice password changing utility.

This round or edition of the Cryptoswift software was even worse than the original edition I had tried and failed with in May.

I got to where the CMS generates its self-signed cert, but then get error back via java that keypair generation on token failed (see below).

```
2002/09/12 16:33:53: get LDAPConnection successfully -->
netscape.ldap.LDAPConnection@31b4c3 localhost 38901
cn=Directory Manager
2002/09/12 16:33:53: caTokenName Arcenciel
2002/09/12 16:33:53: Start removing key and cert
2002/09/12 16:33:53: token name -> Arcenciel algorithm ->
RSA key len 512 Token Error:
com.netscape.jss.crypto.TokenException: Keypair Generation
failed on token:
 at
com.netscape.certsrv.security.KeyCertUtil.generateKeyPair(K
eyCertUtil.java:534)
        at
com.netscape.certsetup.daemon.CreateCertificateProcessor.pr
ocessRequest(CreateCertificateProcessor.java:147)
        at
com.netscape.certsetup.daemon.InstallWizardDaemon.processRe
quest(InstallWizardDaemon.java:278)
        at
com.netscape.certsetup.daemon.InstallWizardDaemon.handleCon
n(InstallWizardDaemon.java:258)
        at
com.netscape.certsetup.daemon.InstallWizardDaemon.run(Insta
llWizardDaemon.java:191)
        at
com.netscape.certsetup.daemon.InstallWizardDaemon.main(Inst
allWizardDaemon.java:465)
ErrorResponse
Sent terminating null.
```

Key pairs are slowly accumulating on token, from the kmu (fancy UI).

I also tried to generate a key pair/cert with the "internal" token, but that doesn't work either. Hangs forever.

A.1.5 Backup/Recovery

I was unable to get the backup and recovery scheme described in the configuration manual to work.

A.1.6 Customer Support

I made many attempts to reach customer support engineers, with very little success. The company seemed to go through some reorganization while we were doing the evaluation, and we kept losing contact and having to re-establish the relationship. I was unable to resolve the problems preventing us from using this device.

A.1.7 Comments

Simplicity is very attractive. Unit doesn't work in our environment. External comment (not one of the other vendors) suggests that the PKCS #11 library is incomplete and thus not usable by Iplanet CMS. This is not proved.

The user manual talks about FIPS 140-1 level 3, but I am not sure it is certified at level 3, but rather at level 2. Nevertheless it probably meets level 3 requirements, or nearly.

A.2 Chrysalis Luna

A.2.1 Hardware Installation

A.2.2 Software Installation

I was not involved directly in the initial hardware (token) and software installation. This appeared to be a simple matter of installing the PCI card in a slot, and installing software packages from the CD. In addition an external reader and (at times) a PED key and keypad device are attached. I verified that that the PCI device was present in the Solaris host; a conventional PCI install. Also verified the presence of the software install packages provide by Chrysalis on distribution CD.

Next, ran */usr/luna/bin/lunadiag*, which provides a large number of tests of the module configuration.

A.2.3 Device Configuration

Ran *enabler* to configure the module for use. *enabler* is clearly a little dangerous, since it can be used to reset the module. Chose the initialization phase to reset the module and build a new configuration. The UI for enabler is a simple text-based menu of choices. It is not clear that this application could be scripted.

Now must deal with FIVE (5) separate physical PED keys, the little colored keys that look like baby toys. The gray key is the token master, and is used to re-initialize the token. The blue key is the SO [Security Officer] key and is used for ordinary administrative functions (mainly user configuration). The black key is the ordinary user key. The red key is a "domain" key. Why this had to be created is something of a mystery, but it has to do with key cloning. The green keys are the "*m* of *n*" key sets. These apply to both SO and user keys, if configured.

I chose to do simple configurations, and did not explore *m* of *n* or key cloning.

It is necessary to ask the *enabler* to refresh its configuration before doing any major work on the token. If you've manipulated the reader in any way (such as installing a token) it won't automatically update its state.

Configuration of the PED keys is described in more detail elsewhere. You are led through configuration of the domain key-by-key; each key is inserted in the PED reader in turn, and a password assigned to it. You also have the opportunity to create *m* of *n* keys, or duplicate keys. I wasn't able to create a duplicate PED key, perhaps because of a combination of errors. If you hesitate too long in answering a query, the software times out, and the operation would have to be restarted from the beginning.

A.2.4 CMS Configuration and Operation

CMS installation is, in its initial phases, identical to installation on a server without HSM. After the initial set of servers are in place, the crypto engine and the PKCS#11 library for it must be enabled. The instructions for this are elsewhere.

The token must be activated. The enabler application is also responsible for this and must be run; if it lacks *activate* and *deactivate token* options in its menu, then the */etc/Chrystoki.conf* file must be modified. These instructions are elsewhere.

If the token is enabled, and the PKCS #11 library and security module are configured, then CMS can now be configured. Do not chose to make a 4096 bit RSA key, CMS 4.2 SP2 cannot handle a keysize > 2048 bits.

A critical step is the configuration of the first CMS administrator, which is the first principal that can issue certificates via the agent port. If the password or name is forgotten, it is necessary to remove the CMS and start over. This can present a problem if you intend to make a CA with the same naming convention; a key with this nickname already exists on the HSM. There is no way to remove this key from the HSM except by re-initializing the token.

Passwords: only the single sign-on password is real. There are usually a couple of other fields that (sometimes) seem to have password fields that you can change, but these are meaningless. The real password is for the crypto module is through the enabler and the PED user key PIN.

The entire configuration operation, Luna and CMS, can be done in about 10 minutes.

What happens when the machine reboots? Everything must be restarted. The token must be started first, so the CA can start.

What happens if the token is ejected from its reader (see picture)? This simulates the effect of token failure.

The CA stays running, but an attempt to sign a certificate will result in an error on the CMS agent page:

```
Request has been completed but no certificate has been
generated.

Additional information:

       java.security.SignatureException

The Certificate Management System logs may provide further
information.
```

The request is "lost" (however, CMS provides a convenient method to clone the lost request, and deal with it when the token problem has been resolved).

CMS restart triggers “black [user] key request” sequence on the PED reader--but not reliably; and too long a delay results in lock out, failure. It seems best to enable the token with *enabler*, and start CMS separately. There is no easy provision for some kind of automated start up. The user keys do not need to be in the PED reader for normal operation.

Long hesitations in answering the request for the black key, whether by the CA or by *enabler*, are a bad thing. The driver seems to time these out, and then locks the unit; it has to be deactivated, reactivated, and the process restarted. Sometimes this doesn’t work.

I also tested multiple CMS instances against a token. This works; the management (user / black key – enabler) of the token is completely separate from CMS, and once the token is enabled, the new instance of CMS can create keys and certificates. However, this instance’s keys must have distinctly different “friendly names” from any other instance’s keys.

We discovered a curious phenomenon related to the PED reader. Once you have “enabled” the token (ie had the black key or user *m* of *n* keys log in), the keys can be put away. And so can the PED reader. We found that the PED reader *should* be removed after installation, because over time its presence seemed to perturb the module somehow; it appeared it would lock up and the CA would be unable to sign or do other operations.

Perhaps the driver detected something, and interpreted some mechanical, thermodynamic, or electrical fluctuation as an attempt to enter codes on the PED keypad or as an attack, and would time out or lock. We don't know, but we had more reliable performance once the PED keypad was removed.

A.2.5 Backup/Recovery

For true disaster recovery, a LUNA installation will require duplicate PED readers, PCI boards, and token readers. We only have one unit for evaluation, so we cannot test a scenario involving different hardware.

The domain backup will require multiple token cards and PED keys. Copies of the drivers and CMS must also be available. Backup and recovery of a LUNA installation involves: backing up the CMS installation using its internal backup procedures (see Iplanet doc, and nCipher below); copying PED keys; copying tokens.

Since the private keys never leave a LUNA token by conventional means, tokens must be *cloned* beforehand; shortly after the CA is configured is a good time, since the token now contains the keys it uses in daily operations. If these are changed, then the cloned token must be updated too. The LUNA token reader has two slots and is designed to handle cloning. The procedures are described in chapter 6 of the "Luna Planning Guide". The clone or destination token must be initialized just like the original token, except that it must be a part of the domain of the original, and so must use the original's red key to acquire the domain membership. Since the token is initialized separately, it can have its own gray, blue, and black keys (or m of n policies). This procedure does suggest that the *red key* is the critical resource, and copies of this must be made to ensure that the domain can survive a catastrophe.

Objects are cloned from one token to another on a case-by-case basis. It is really only necessary to clone the user objects, because these are what are used in day-to-day operations.

I was able to successfully initialize, clone, and restart using the cloned token in about 5 minutes.

The "Group Keys" option in enabler allows you to build a multi-token infrastructure with just one set of keys rather than a different key set (user keys) for each token. However the operation of grouping and managing the pins for these keys and making sure the keys worked from token to token is quite complex. I couldn't begin to explain it, and didn't try it.

You can duplicate the user key if it is in an m of n configuration (an option to duplicate is available through *enabler*). Otherwise, it appears that the only opportunity you have to create duplicates, especially the all-important red domain key, is at the very beginning, at token initialization time. I did not appreciate this detail in full at the time. I attempted to duplicate some of these keys, but without success (perhaps due to time out, or mishandling the blue SO key). As a result I had no duplicates and was unable to use a duplicate domain key in disaster recovery.

A.2.6 Customer Support

I didn't need much customer support; there were a few areas I should have clarified. My contacts with their sales / engineering staff were favorable.

A.2.7 Comments

The system is somewhat old-fashioned, appears to be solid and conservative, but is rather complex and has many roles that don't seem necessary for our operation at this time. I had the feeling that some of the functionality (group keys, m of n, domains) was sort of layered in later on, adding more complexity.

This system by its nature is FIPS 140-1 level 3.

If implementation choices were kept simple it would be a great root CA or offline CA solution. The value in this device is the absolute integrity of the signing private key. This would also be a good online CA device for an institution requiring a very high degree of assurance about its CA keys, and content to support the complexity of operation required.

I am puzzled by the critical nature of the red domain key, and the lack of ability to copy this key for disaster recovery. Probably there is something that I missed.

The cloning operation is simple and straightforward … but since the keys only inhabit the tokens, it would be quite easy for things to get out of phase, over time. The tokens will likely be dispersed geographically and this introduces another set of problems into the management.

A.3 NCipher nShield

A.3.1 Hardware Installation

A.3.2 Software Installation

I was not involved directly in the initial hardware (token) and software installation. “Hardware installation” is a conventional matter of installing the PCI card and driver, and software installation a simple matter of installing the packages from the distribution CD.

The device relies on a daemon (*nfast*), which it calls the hardware server. This needs a *tcp* port (currently uses port no. 9000), not clear why.

/opt/nfast/bin/enquiry shows card state; the smart card reader can be tested with */opt/nfast/bin/slotinfo* <module number> <slot number>

A.3.3 Device Configuration

Device configuration requires manipulation of the *nshield* card, and creation of a new security world. Security world creation will include the creation of the administration role cards. The basic procedures are described elsewhere. Two critical decisions are made at this point: FIPS-140-1 level 3 compliance, and the number and nature of administrator cards. It is possible to have an m of n admin card configuration. We learned later that it is not possible to change the admin card configuration. FIPS level 3 adds an additional element of complexity later (usually seen in a need to specify the card reader module, but also shows up in admin functions).

After the *nshield* device is prepared and the admin card set created, the *nshield* device must be switched back to operational, and the *hardserver* daemon must be restarted. If you don’t do this, various errors occur in subsequent steps, usually “module failed”.

Once the *nshield* card is reset to “O”, create the operator card set with *createocs* (described elsewhere). Consider using the *–p* persistent flag. The *–F* flag for “FIPS 140-1 level 3” is required if you have created a security world FIPS-compliant. The details are described elsewhere.

Install the PKCS #11 library and reference in the Iplanet security database. This is identical to the Chrysalis installation (except for the naming of libraries).

In the course of doing this configuration, I discovered a FIPS or security related problem that hadn’t been noticed previously. I found some of the smart cards just didn’t work; *createocs* would reject them, and so would *new-world*. We subsequently learned that smart cards must be handled carefully. OCS or user smart cards can only be erased and readied for re-use by the admin card set controlling the active security world. Admin cards from a different security world can be erased any time, but the service will not erase admin cards from its own set (without some special handling). Once admin cards are all erased, remaining OCS cards are useless—they cannot be used, and they cannot be erased and prepared for re-use. I managed to burn up quite a few cards before I realized what was happening, and we had to suspend testing for a few days while we acquired new cards. We also learned that nCipher uses only a particular type of Gemplus card; our old Gemplus 159’s from smart card experiments last year were rejected.

Note: the installation process sets up the *nfast* / *hardserver* daemon to start automatically on reboot.

A.3.4 CMS Configuration

CMS installation is, in its initial phases, identical to installation on a server without HSM. After the initial set of servers are in place, the crypto engine and the PKCS#11 library for it must be enabled. The instructions for this are elsewhere.

If the nfast or hardserver daemon is running, and the PKCS #11 library and security module are configured, then CMS can now be configured. Do not chose to make a 4096 bit RSA key, CMS 4.2 SP2 cannot handle a keysize > 2048 bits.

Passwords: Both the single sign-on and the crypto module password are real – the crypto module password must match the password of the operator card set.

This exposed a defect in the CMS product. Passwords are cached on the disk of a CMS server, encrypted using the single sign-on password (which ordinarily is not on disk). But one can operate without this password cache; or one can try running the service with multiple operator/crypto module passwords -- for example, putting a unique password on each operator card, or putting a bogus password in the password cache. CMS will then try to do a password query from the console. This wanted to present an X11 UI for password, but some defect in it prevents it from using the X11 MIT authentication cookie, and so the password query fails.

For the time being, until this problem is investigated in CMS, an operator card set is restricted to a single password, it must match what’s in the CMS password cache, and the cache must be used. There are some other possible configurations that weren’t tried, but this one works and avoids pitfalls.

I was able to bring up many instances of CMS, with separate host (SSL) keys, using RSA 1024, 2048, and DSA-1024 keys, and simultaneously do certificate operations with them. 2048-bit SSL server key pairs worked quite well. I did not test the limits of the SSL accelerator that this product includes.

The *nfast* / *hardserver* daemon is critical. If it exits, the token seems to be logged out. Restarting the daemon doesn’t seem to be enough to restore the CMS to operational; it seems to be necessary to restart CMS.

I chose to make the card sets persistent. This wouldn’t be the right choice under every circumstance, but it may be useful for an online CA that expects only occasional reboots. The card is used for the initial CMS start up, and then it may be removed from the card reader and secured in a safe. (In fact the reader can be removed as well.) In the case above where multiple instances ran, some CMS instances used one card set, and some used another; all were configured as persistent cards, and removed from the device after CMS start up. Here is how the cardsets looked after that experiment:

```
rocs> list keys
  No. Name                        App        Protected by
    1 caSigningCert cert-MCr   pkcs11     Hand2
    2 caSigningCert cert-MCr   pkcs11     module (Hand2)
    3 ocspSigningCert cert-MCr pkcs11     Hand2
    4 ocspSigningCert cert-MCr pkcs11     module (Hand2)
    5 Server-Cert cert-MCr     pkcs11     Hand2
    6 Server-Cert cert-MCr     pkcs11     module (Hand2)
    7 Remote Admin Server-Cert pkcs11     Hand2
    8 Remote Admin Server-Cert pkcs11     module (Hand2)
    9 caSigningCert cert-yoyo  pkcs11     Hand2
   10 caSigningCert cert-yoyo  pkcs11     module (Hand2)
   11 ocspSigningCert cert-yoy pkcs11     Hand2
   12 ocspSigningCert cert-yoy pkcs11     module (Hand2)
   13 Server-Cert cert-yoyo    pkcs11     Hand2
   14 Server-Cert cert-yoyo    pkcs11     module (Hand2)
   15 Remote Admin Server-Cert pkcs11     Hand2
   16 Remote Admin Server-Cert pkcs11     module (Hand2)
   17 caSigningCert cert-Monke pkcs11     Monkey
   18 caSigningCert cert-Monke pkcs11     module (Monkey)
   19 ocspSigningCert cert-Mon pkcs11     Monkey
   20 ocspSigningCert cert-Mon pkcs11     module (Monkey)
   21 Server-Cert cert-Monkey  pkcs11     Monkey
   22 Server-Cert cert-Monkey  pkcs11     module (Monkey)
   23 Remote Admin Server-Cert pkcs11     Monkey
   24 Remote Admin Server-Cert pkcs11     module (Monkey)
```

```
list cardsets
No. Name                         Keys (recov) Sharing
  1 Hand1                        0 (0)        1 of 2;
persistent
  2 Hand2                        8 (8)        1 of 2;
persistent
```

```
  3 Monkey                   0 (0)        1 of 2
  4 Monkey                   4 (4)        1 of 2;
persistent
```

If the right card isn’t present in the reader, you get a “single sign on” error, which seems like the wrong error message – it’s not the single sign on that’s the problem.

It is possible to configure the operator cards to be persistent, but to require a re-login after a set time. I didn’t test this; it is not clear how CMS would react to this configuration.

A.3.5 Backup/Recovery

For true disaster recovery, an nCipher installation will require duplicate card readers and PCI boards. We only have one unit for evaluation, so we cannot test a scenario involving different hardware.

Quite a bit of time was spent studying the card backup problem (see above for details). I exercised the OCS duplication capability as well as I could; I lacked sufficient cards to do a complex experiment, such as changing the nature of the OCS card set, adding more cards, changing persistence value. I lacked cards to do an administrative card recovery (this is essentially replacement).

Here is the cardset after the creation of a new cardset; all the keys originally started out in Hand1, and after using rocs, they were migrated to Hand2.

```
rocs> list keys
  No. Name                     App        Protected by
    1 caSigningCert cert-MCr   pkcs11     Hand2
    2 caSigningCert cert-MCr   pkcs11     module (Hand2)
    3 ocspSigningCert cert-MCr pkcs11     Hand2
    4 ocspSigningCert cert-MCr pkcs11     module (Hand2)
    5 Server-Cert cert-MCr     pkcs11     Hand2
    6 Server-Cert cert-MCr     pkcs11     module (Hand2)
    7 Remote Admin Server-Cert pkcs11     Hand2
    8 Remote Admin Server-Cert pkcs11     module (Hand2)
    9 caSigningCert cert-yoyo  pkcs11     Hand2
   10 caSigningCert cert-yoyo  pkcs11     module (Hand2)
   11 ocspSigningCert cert-yoy pkcs11     Hand2
   12 ocspSigningCert cert-yoy pkcs11     module (Hand2)
   13 Server-Cert cert-yoyo    pkcs11     Hand2
   14 Server-Cert cert-yoyo    pkcs11     module (Hand2)
   15 Remote Admin Server-Cert pkcs11     Hand2
   16 Remote Admin Server-Cert pkcs11     module (Hand2)
rocs> list cardsets
No. Name                         Keys (recov) Sharing
  1 Hand1                        0 (0)        1 of 2;
persistent
  2 Hand2                        8 (8)        1 of 2;
persistent
```

I did a quite elaborate and complete service backup and recovery, which included simulating recovering the CA service from "a fresh install". To some extent it got a little tangled up with the OCS card recovery work.

- tarred up kmdata & netscape/server4
- did cmsbackup, moved to /var/tmp
- did config ldap backup, moved to /var/tmp
- did database backups of two ldap servers, moved to /var/tmp
- copied admin-config to /var/tmp
- stopped all the servers

Forgot to copy slapd basic configs

Then I moved kmdata & netscape/server4, and rebooted the server host.

Re-installed Iplanet CMS with original download; used parameters from backed-up config files for ports and other config items.

- Restored the test-pki1 config directory
- restarted ldap server
- restarted admin server

admin server has same config:

```
diff -br config /var/tmp/admin-config/config
```

```
Only in /var/tmp/admin-config/config: secmod.db

diff: config/secmodule.db: No such file or directory



 setenv LD_LIBRARY_PATH /usr/netscape/server4/lib
```

[Built a fresh security module:]

```
../../shared/bin/modutil -dbdir . -nocertdb -create



WARNING: Performing this operation while the browser is
running could cause corruption of your security databases.
If the browser is currently running, you should exit
browser before continuing this operation. Type 'q <enter>'
to abort, or <enter> to continue:



Creating "./secmod.db"...done.

ssh:test-pki1# ../../shared/bin/modutil -dbdir . -nocertdb
-add "Hand2" -libfile
/opt/nfast/toolkits/pkcs11/libcknfast.so
```

[I had to know what operator card set I was going to use….]

```
WARNING: Performing this operation while the browser is
running could cause corruption of your security databases.
If the browser is currently running, you should exit
browser before continuing this operation. Type  'q <enter>'
to abort, or <enter> to continue:

Using database directory ....

ERROR: Failed to add module "Hand2".
```

Now added driver &c as in PKCS #11 install—but this is too early, CMS wouldn't start.

Had to remove everything and reboot. What I did was restore out of sequence. Start by rebuilding the ncipher security world; first mkdir kmdata, then new-world, using an existing administrator card. This requires switching the "new" card to init state.

```
new-world -l -m 1 -s 0

new-world: no card in module 1 slot 0

Insert an administrator card into module 1 slot 0 and press
return...

Passphrase for administrator card 1:

security world loaded; hknso =
51a454daaca39158e8955ea27ff1128609f4d45c

No messages were displayed; must be a -v option somewhere

ssh:test-pki1# /etc/init.d/nfast stop ; /etc/init.d/nfast
start

reset to operational

/etc/init.d/nfast stop ; /etc/init.d/nfast start
```

nFast server now running

Now I put in one of my operator cards.

```
cardpp -s 0 -m 1 check

Passphrase for card `AceOfSpades' (1) of cardset `Hand2':

cardpp: passphrase for card `AceOfSpades' (1) of cardset
`Hand2' OK
```

Now back to CMS install

Oops – created the new instances of cms and its ldap database with the wrong instance names; this may not be bad (won't be visible to customers) but would be confusing; junk these cms / ldap instances and reinstall. Then do the CMS restore:

```
./cmsrestore /var/tmp/CMS_MCr_BACKUP-20020917160706.zip

Would you like to (c)ompletely restore ALL data specified
in the package, or would you like to be (p)rompted
```

```
to restore SELECTED data from the package
    [(c)omplete/(p)rompted/(q)uit]?  c
Please enter single sign on password:
********
Waiting for server to start. . .
Error: CMS failed to start: Token Hand1 not found.
Please enter single sign on password:
```

Is passwd cache there? Yes.

Doesn't like the other cardset; something is screwed up.

Whoops; cfg still has Hand1 in it from card recovery experiments. The password cache must be edited using the cms tool to include the name of the right OCS card set.

Ok; now can log in, and the ldap database is restored!

The one missing thing is the config database contents;

need to investigate this so as not to lose things. The lesson here is to not use the CMS Iplanet server domain for anything else, don’t put other ldap infrastructure, user logins &c here.

Total time taken: 2 hrs; about 35 min for this later successful recovery.

A.3.6 Customer Support

The company seemed to go through some reorganization while we were doing the evaluation, and we kept losing contact and having to re-establish the relationship with a new sales representative. Curiously, however, the sales engineer was the one constant in this period. nCipher’s SE was very helpful in resolving some confusing points in smart card handling and role management that came up in the evaluation and while doing backup and recovery.

A.3.7 Comments

nCipher has a lot of “presence” in this industry but it’s not clear how much market penetration they have made.

The local SE is great, but losing him could be bad if we chose this product. On the other hand, the other vendors didn’t provide this kind of support at all.

The product is very simple and “neat”. The card reader and cards can be removed after start up, leaving a clean set up; they are also reasonably compact and can probably be left in the rack without consuming a lot of space.

On the other hand, the architecture is complex. The best way to deal with this is to take the minimum set from the architecture.

Storing the private keys on disk, in the /kmdata part of the security world, is going to be hard to explain. nCipher appears to use a hardware rather than a pseudo random number generator, and these important keys are both generated and wrapped by 3DES keys based on the RNG; but the disk copy of these keys is bound to make some people very nervous.

Security world management is very flexible.

Smart card management is unexpectedly more complex. The cards are also somewhat prone to failure in my experience. One has to invest in a lot of cards and duplicates. At least one has an alternative source for these cards, in Gemplus.

m of *n* is not possible for user (operator) keys in this service.

I do not understand why the *nfast* service needs a *tcp* port.

Appendix B Dhiva's HSM Evaluation Experience

B.1 Rainbow CryptoSwift

B.1.1 Hardware/Software Installation

I was not involved in the initial hardware (token) and software installation. Driver installation from the CD seems to be simple (like any other Solaris PKG install).

B.1.2 Device Configuration

Device configuration was simple. The Key Management Utility (KMU) requires Java Runtime Environment 1.3.0. Please note that KMU graphical Interface shows up, even with the lower version of JRE. But it doesn't recognize any HSM device attached to the host.

First time Invocation of KMU starts the Initialization process of the Device. Subsequently it leads to role management operations (Security Officer and User setup).

The original unit has the capability to create key-pairs up to 1024 bit keys.

B.1.3 CMS Configuration

CMS installation is, in its initial phases, identical to installation on a server without HSM. Rainbow PKCS#11 library has to be enabled as an External Token for CMS, so that CMS can use the Rainbow Device. The instructions were given in CMS installation manual. And this is common for most of the HSM devices. Finding the PKCS#11 driver library is important for this step.

I was able to run thru the rest of the CMS configuration. I was able to create the Self Signed Root CA certificate and Self signed SSL certificate for my CMS instance. Till this point I haven't encountered any problem OR any unusual instruction to follow.

The problem was with First Agent Cert. As I doing the configuration, I need to get a first Agent certificate, so that I can approve other request. I was able to get this Certificate installed with my Web browser successfully. But the service was not able to authenticate this certificate as an Agent. The web browser shows the following error message:

```
"The server could not verify your identity"
```

The CMS log file has the following error message:

```
"File Access denied: Failed to Authenticate -
java.net.SocketException: Error in SSL handshake (Peers
certificate has an invalid signature.) "
```

This error message led me in the wrong direction. I ran through the RootCA and SSL cert creation cycle again to make sure that I hadn't selected any unwanted extensions.

Also tried equating the HSM user password, CMS token password and the Single Sign on password. I also checked the file system permissions. But none of these tries worked. I couldn't find any information in the manual and the FAQ section. On this occasion I lost the password, and as a result the iKeys got locked – they are configured to lock after several unsuccessful password attempts. I had to gave them to Mike to reset the iKeys. Mike had installed the required software on his Win 98 machine (which is the only platform on which these administrator utilities seem to work).

Rainbow Customer Support responded and they come up with a new driver. I have received the FTP download information from the Rainbow Sales Engineer. According to Mike's discussion with Rainbow I was informed--I was under the impression--that the new driver does support 2048 bit key- t. So I tried to complete the CMS install with 2048 key-size. The CMS key-pair generation had failed with the following error message (from the CMS log file):

```
…. RSA key length 2048 Token Error …. Keypair Generation
failed on token.
```

Later on I came to learn that the 2048 bit key-size was never supported. Rainbow does have the driver for 2048 size, but it wasn't approved by FIPS. It also requires a cryptoSWIFT firmware upgrade. Rainbow Tech support team requested the CMS distribution for themselves to simulate the problem. I sent them the distribution, but never got a response back from Rainbow Tech Support about this.

B.1.4 Backup/Recovery

I was able to do the backup as described in the manual. However I was unable follow the recovery scheme as described. Wrap key needs to be generated by SO, in order to create the Backup. The backup archive was a xxxx.oar file with 3DES encryption using the Wrap key.

The Recovery operation requires the Rainbow HSM device, new iKeys (SO & User) and from the backup the xxxx.oar blob and CODE-xx iKeys. The CODE-xx keys are basically labeled as CODE-SO and CODE-USER. The manual has the instructions about Split & Combine operation of the Wrap key. But it doesn't have enough information about the CODE-xx iKeys and the Recovery Procedure.

Rainbow Sales Engineer responded to my query about the Recovery procedure. The CODE-SO and the CODE-USER iKeys are attached to the SO and User Roles and not the actual keys. This means the same Wrap key and the Backup data can be duplicated to any number cryptoSWIFT for any SO and USER. The SO and the USER needs to authorize the 'Combine Key' & 'Recovery' operation. I was able to restore the CMS and Rainbow environment successfully.

B.1.5 Customer Support

Considering the problems, solutions and tech support response as described above, from my point of view the customer support was OK.

B.1.6 Comments

Very simple to install and configure. Unit works fine for only 1024 key-pair token. We could consider this unit for Remote Registration Manager. Remote Registration Manager is a special kind of entity, and it doesn't need to have 2048 size key-pair. The two black split keys attached to the SO & User roles instead of SO & User is a surprise to know. I am worried about the lifetime and the reliability of the USB base iKeys. Rainbow does not have the strategy to duplicate this iKey and keep it in a safe vault. If you lose the SO password, then you cannot de-initialize / initialize the unit.

B.2 Chrysalis Luna CA3

B.2.1 Hardware/ Software Installation

I was not involved in the initial hardware (token) and software installation. The driver installation of this Luna CA3 is just like any other Solaris Package install. The manual has the instruction for connecting the Dock Controller card, DOCK card reader and PED device together.

To verify the software & hardware install, I ran the utility /usr/luna/bin/lunadiag, which confirms the successful installation.

B.2.2 Device Configuration

The heart of the Luna CA^3 is the 'enabler' application, which has the complete control over the HSM unit. "Luna PKI HSM Planning & Integration Guide" has the complete details about the enabler application (/usr/luna/bin/enabler)

Preparation of Luna CA^3 token:- Preparation of a Luna token starts with the Initialization phase. It is required to know about the PED keys and Pin Entry Device from this point onwards. You need to have PED device, PED keys, token and PIN for every operation. Installation Guide has the information about these peripherals.

I have gone through the following operations for Luna CA^3 preparation:

- Ran the enabler application
- Chose the initialize a token
- Inserted a Token into slot 1
- Chose 'freeze' as a name of the token
- Subsequently provided the required PED keys (GRAY, BLUE, BLACK and RED). Also I have decided the PIN for BLUE & BLACK keys. BLUE key is a Security Officer key and the BLACK key is User key.

I have also explored '*m* of *n'* keys using the GREEN keys and exercised 'key cloning' using the RED keys. If you choose the *m* of *n* security option, then most of the SO and User operations requires minimum of *m* keys along with the SO & User keys.Though the manual explains about these operations, I got confused with the following message on PED device: **"Do you wish to use a group PED key Y/N?"** and **"Are you duplicating the PED Key Y/N?"** So most of the time I had to back track the sequence to move forward. This is because I had to deal with many numbers of keys in five different colors. Sometimes I did overwrite the key with the new pin. If you overwrite the key, then you will not be able to access the associated objects stored in a token. Probably those objects live in a token as garbage.

The 'Planning and Integration Guide' doesn't have enough information about configuration for iPlanet CMS, Apache and other products. The enabler application does have options like 'activate token' and 'deactivate token' for external application. Initially when I ran the enabler application it didn't show up those options. I was able to fix this problem, with the help of Chrysalis Tech Support. They replied in time for my queries. I needed to include the following changes in 'Chrystoki.conf' configuration file:

```
Misc = {
NetscapeCustomize=1023;
AppIdMajor=2;
AppIdMinor=4;
}
```

Please note that you need to make changes to the file under */etc* folder. */usr/luna/etc* is another folder where you can find a 'Chrystoki.conf' file, but the application was using the file */etc/Chrystoki.conf*. After these changes I was able to activate the token for CMS use. This basically opens a session for a CMS application to connect via PKCS#11 library to do cryptographic operations. This action requires PED device operation and subsequently it also requires USER key and USER PIN authorization.

The long hesitations in answering the request for the black key, whether by the CA or by *enabler*, are a bad thing. The driver seems to time these out, and then locks the unit; it has to be deactivated, reactivated, and the process restarted. Sometimes this doesn't work.

B.2.3 CMS Configuration and Operation

CMS installation is, in its initial phases, identical to installation on a server without HSM. Then I configured the Chrysalis PKCS#11 driver as an External Token with iPlanet CMS.

The Luna token must be activated, before the CMS key-generation event. Once the Token is activated, we can close the *enabler* application. CMS' internal key generation requires a password to be assigned to the signing key pair (a mandatory field in the UI). Since the *enabler* application opened up a session for CMS, we don't have to provide the real Luna User password during the key-pair generation. But we need to give some bogus password, because the CMS won't allow empty password. I was able to continue the CMS configuration and generated Key-pair and certificate of 2048 bit key size with Luna Token. After that I was able to generate the Agent certificate and sign a request.

The long hesitations in answering the request for the black (user) key, whether by the CA or by *enabler*, are a bad thing. The driver seems to time these out, and then locks the unit; it has to be deactivated, reactivated, and the process restarted. Sometimes this doesn't work.

I wanted to use the same token for another CMS instance, but it didn't work. The CMS throws an error saying 'Token was already found on the token'. So I had to reinitialize the token

B.2.4 Backup and Recovery

The backup procedure was explained in the iPlanet CMS manual. I ran the *cmsbackup* utility for CMS backup. For the Luna Token backup, we need to clone the Token. This is because the *enabler* doesn't provide any function to export the token content in any form. The Luna CA^3 Token can be cloned for disaster recovery. All the PED keys can also be duplicated for disaster recovery. We must have the RED PED key, to clone the token. I did clone the Luna Token using the *enabler* application. I named the backup token as 'spring'. The original token name was 'freeze'. Then I uninstalled the CMS software and Luna driver.

I started the recovery process with the backup token, cmsbackup zip file and the required PED keys. I was able to install the CMS and Luna software successfully. I was also able to activate the token 'spring' for CMS application. Though the backup token has a different name I was able to use the same PIN and same PED keys to authorize token. I wanted to use the same PIN and PED keys, especially because it was backup. I was able to complete the *cmsrestore* operation. But the CMS instance didn't start, because the internal database has a reference to a token name 'freeze' for all key pairs and certificates it uses. Tthere are some workaround available if you want to use a different name for the backup token. You have to find and replace the token name in CMS.cfg configuration file

I started the backup and recovery exercise all over again and named the backup token as same as original token. Then I was able to restore the CMS and the Luna Token environment successfully.

B.2.5 Customer Support

As I have mentioned earlier, I was in touch with Tech support for Chrystoki.conf file changes. The technical support team was good.

B.2.6 Comments

This system by its nature is FIPS 140-1 level 3.

Facts about the *Enabler* Application:-

- Anybody can destroy the token, if they have access to the enabler application and the GRAY key; it doesn't require any password or authorization.
- 'activate token' option was good as well as bad. Good, because it was self contained, so that we don't have to provide the password OR pin, when we come through an external application such as CMS. At the same time, this means anybody who has access to that machine can use the TOKEN for similar cryptographic function. For. example. I could set up another CMS instance and still use the same token for the key-pairs and certificate,because I don't need to know any password OR pin to access the token.
- Luna environment was too complicated to maintain, particularly the different set of keys and their duplicates along with different passwords and PIN.
- Two slots are available with the Token Reader. We could opt for high availability token environment: If the token in Slot one got corrupted, then the application could use another token available on Slot two for cryptographic functions.

B.3 NCipher nShield

B.3.1 Hardware/Software Installation

I was not involved in the initial hardware (token) and software installation. The driver installation of this nCipher is just like any other Solaris Package install.

The nCipher environment is called "Security World", and the *hardserver* is the heart of this world. There are few command line utilities available under */opt/nfast/bin* folder, to check the status of the security world and token status. 'enquiry', 'slotinfo' and 'nfkminfo' are a few of them.

B.3.2 Device Configuration

Device configuration involves creation of security world and creation of Admin Card Set (ACS) and Operator Card Set (OCS). ACS and OCS are sets of smart cards. ACS is used for functions such as backup and restore. OCS is used for other cryptographic functions such as key generation, certificate creation and signing. Security world creation will include the creation of the administration role cards (ACS). The basic procedures are described in the nShield manual. The security world can be created with FIPS-140-1 level 3 compliance. If the Security world is FIPS-140-1 level 3 compliant, then OCS creation needs ACS authorization. Otherwise OCS can be generated without any authorization. But you need to have a hard server running with a security world for OCS generation.

It is possible to have an *m* of *n* admin card configuration.

Once the security world is in place with ACS, then change the device state to operational. I have learned that it was better to restart the hard server at this point and then proceed to OCS creation. OCS creation is explained in the User Manual. OCS can also be *m* of *n* configuration. We can also have a "persistent" OCS.

Now the nCipher device is ready for CMS as an External Cryptographic Token. We can install the PKCS #11 library and reference in the Iplanet security database. Unlike the other devices, we cannot install the nCipher PKCS#11 library at any time. We must have a Security World, ACS and OCS in place. In particular one of the OCS card has to be in card reader.

There are situations where one of these smart cards becomes bad; we can't use it anymore. FIPS compliant ACS and OCS cards cannot be recycled by a non-FIPS compliant Security World. So we must erase all ACS and OCS cards, before we trash the security world. We could reuse the ACS cards with the new security world by overwriting it. But OCS cards are tightly bound to the Security World, where they were born. If we decide to trash the security world, then we should first erase all the OCS cards. Otherwise we wil not be able to use them with the new Security World. Sometimes smart cards just go bad by themselves.

Note: the installation process sets up the nfast / hardserver daemon to start automatically on reboot.

B.3.3 CMS Configuration

CMS installation was similar to installation on a server without HSM. After the initial set of servers are in place, the crypto engine and the PKCS#11 library for it must be enabled. You must provide 1 OCS card in the card reader, during the nCipher PKCS#11 driver configuration.

Note:- There are three database files for maintaining the key, certificates and the security tokens. Key3.db, cert7.db and secmod(ule).db are those files and specific to iPlanet CMS. Initially these files are placed under $CMS_HOME/admin-serv/config folder. On completion of any CMS instance install, these files are copied into $CMS_HOME/<CMS-instance> folder.(*wd-key3.db, wd-cert7.db & wd-secmod.db*) From this point onwards these files are specific to this CMS instance's keys and certificates. This behavior is specific to iPlanet CMS and nothing to do with the External Tokens. Assume that you already have a CMS instance, which was configured to use internal software token. If you want to change this internal token to an external token, then you should configure the *wd-secmod.db* under $CMS_HOME/<cms-instance> folder.

Initially we were given a small number of smart cards. So I configured OCS with 3 smart cards and 1 card is required for operation. I went through the full CMS configuration cycle without any problem. I have used 2048 bit RSA key. Once the CMS is up and running, then we could also remove the OCS card from card reader, if it is configured "persistent".

I continued the evaluation with different m of n OCS combination. I chose 2 of 5 OCS. i.e. minimum 2 operator cards are required out of 5. This exposed a defect in the CMS product. I was not able to configure the CMS, because of an error message '**Invalid Password**'. The CMS configuration UI was not designed to accept more than one password for external token. One other problem could be the CMS was not able to handle the response from the Security World. Ideally in this scenario, security world should be asking for card no. 2 and relative password; but CMS has only one password field in its UI and was not able to ask the user for another card, it could still be giving the same password.

I also tried the card set as non-persistent. This would be the right choice for the Root CA. Persistent card set could work well for Subordinate online CA. Persistent card set is a good choice for running multiple CMS instance on a single machine.

B.3.4 Backup and Recovery

Ran the '*cmsbackup*' and got the zip file as a backup. The zip file contains the CMS internal data and CMS configuration including its Admin server. We also need to archive the */opt/nfast/kmdata* directory for nCipher generated key-pair objects. I moved the kmdata directory and the cmsbackup zip file into AFS shared file system. As I mentioned earlier the persistent card can be removed and kept in a safe locker along with the other card set. I had started the recovery operation, on the same machine with all the above components.

I uninstalled the CMS and the nCipher software on the host.

I started the recovery operation by installing nCipher software and the CMS software from the distribution. Then initialized the nCipher device and restarted the hard server. Security World can be loaded from the kmdata directory, for recovery purpose. I was able to load the security world using the ACS and the kmdata directory. The step by step procedure can be found in another document or you could refer the nCipher manual.

Then I configured the nCipher PKCS#11 library for CMS as anexternal token. Then I configured the CMS instance as a root CA, using software token. Now I ran the *cmsrestore* utility with the CMS backup zip file. This operation gives several options by which we can do complete recovery or module-by-module recovery. This procedure was available in the iPlanet Command Line utility document.

I spent about an hour on restore operation and it was successful. We could also use different operator card; I exercised the recovery operation with the following Assumptions:-

- Operator Card set was persistent
- Used the same Operator Card for recovery For example. Card 1 out of 5 cards
- All the port numbers were as same as the original CMS instance
- All the passwords were same as the original

B.3.5 Customer Support

The sales engineer was helpful providing information for our problems. We did ask the Tech Support for help with the FIPS compliance issue we had with the CMS. They were prompt and helpful.

B.3.6 Comments

The local SE is great, but losing him could be bad if we chose this product. On the other hand, the other vendors didn't provide this kind of support at all.

The product is very simple and "neat", although I felt dealing with the security world was complex, compared to other products. The generated key-pairs are stored in the hard disk in a 3DES encrypted format.

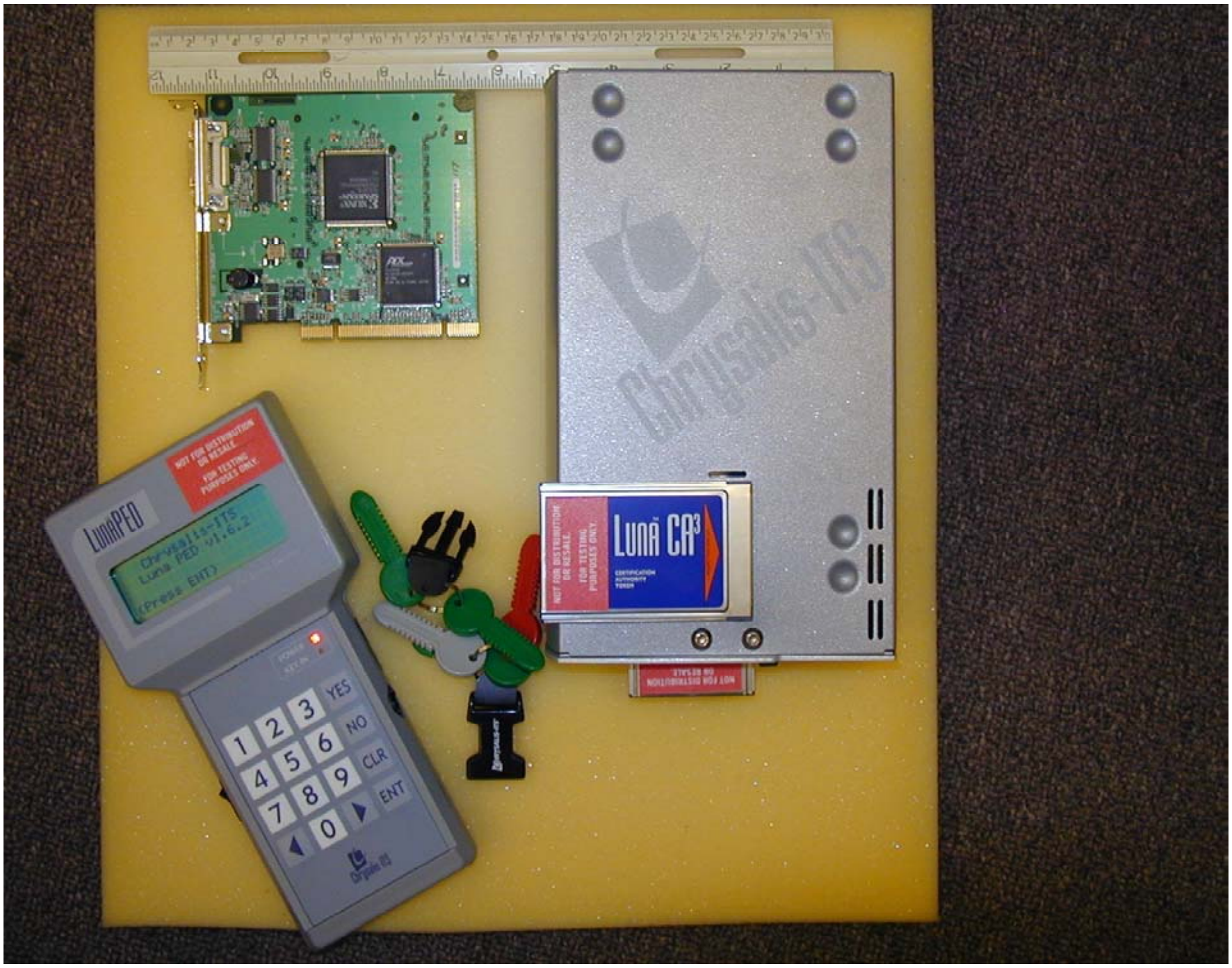
Security world management is very flexible.

Smart card management is simple. This could be because I haven't gone through replacing the OCS and ACS cards.

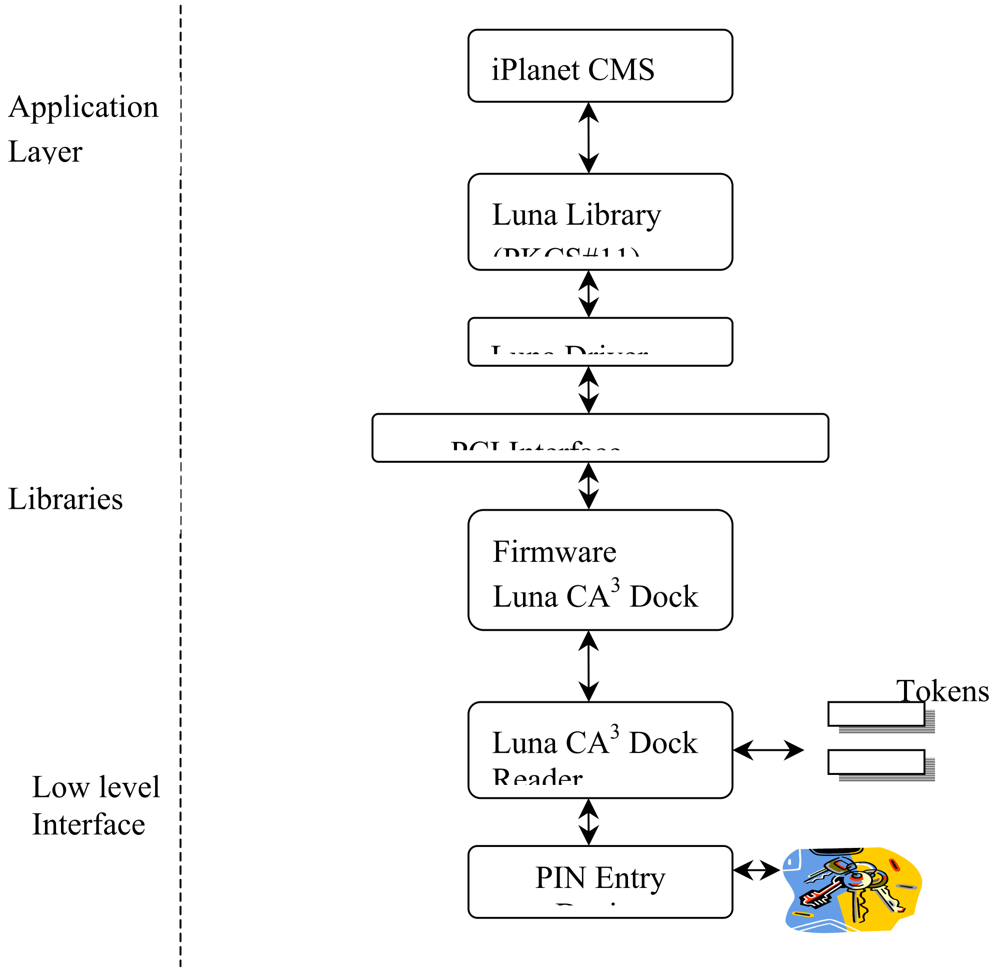
Though the Security World provides *m* of *n* keys for Operator Card Set, it is not compatible with iPlanet CMS. This could be resolved by using the 'with-nfast' command line utility. This tool was basically like the 'enabler' application with Luna. We are working with nCipher Tech Support on this.

Appendix C HSM Pictures

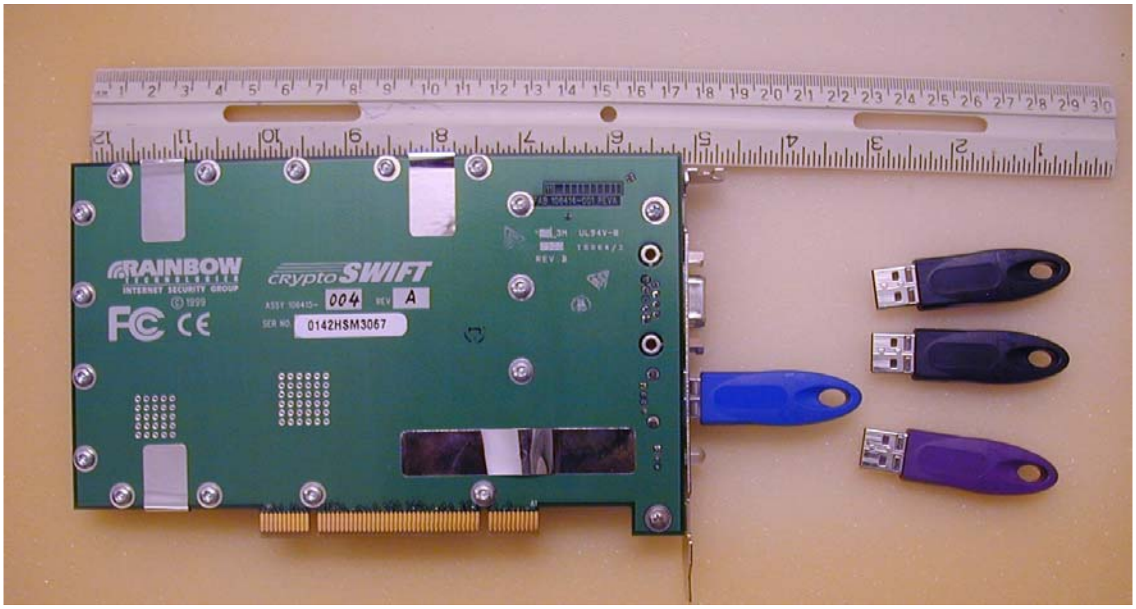
C.1 Luna CA^3 Components



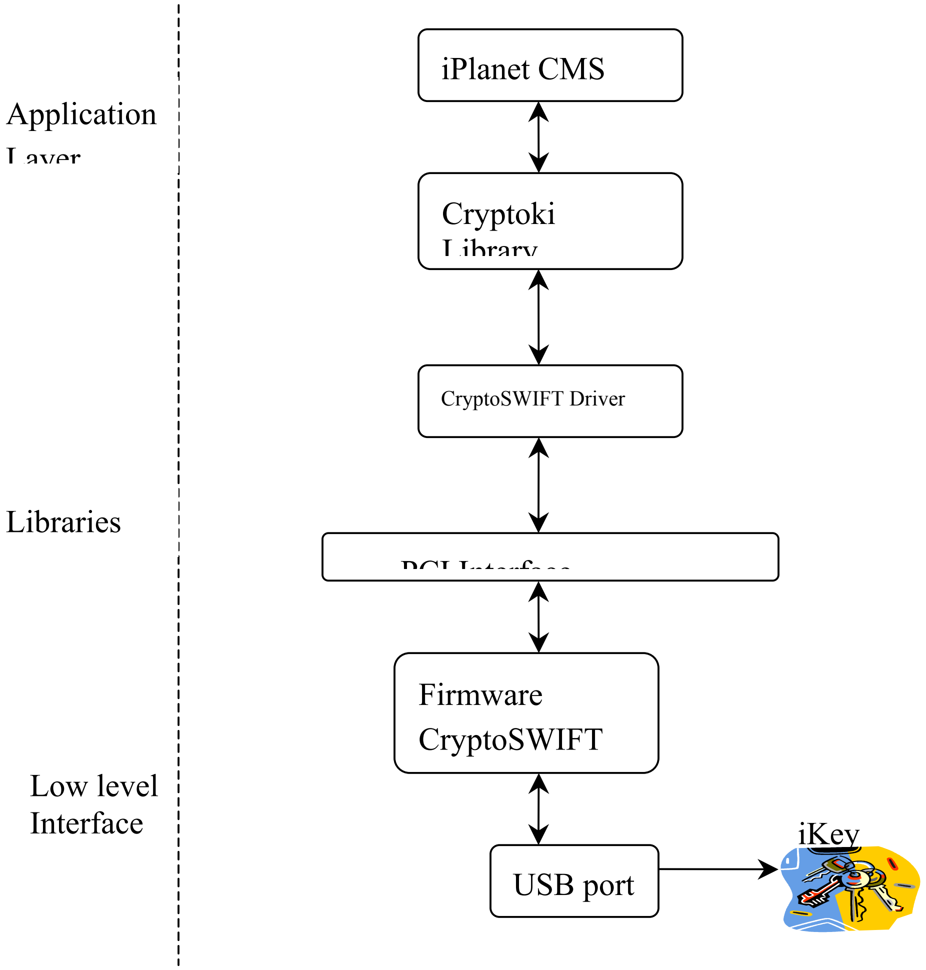
C.2 Luna CA Architecture



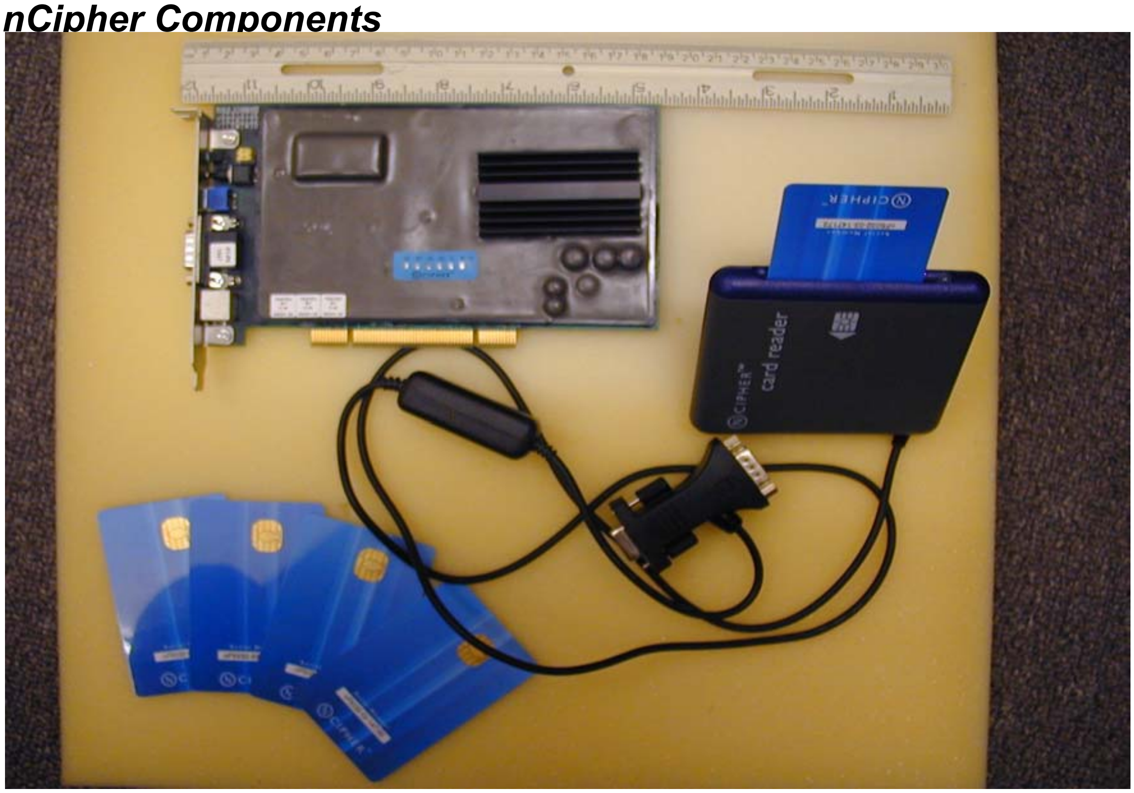
C.3 Rainbow Components

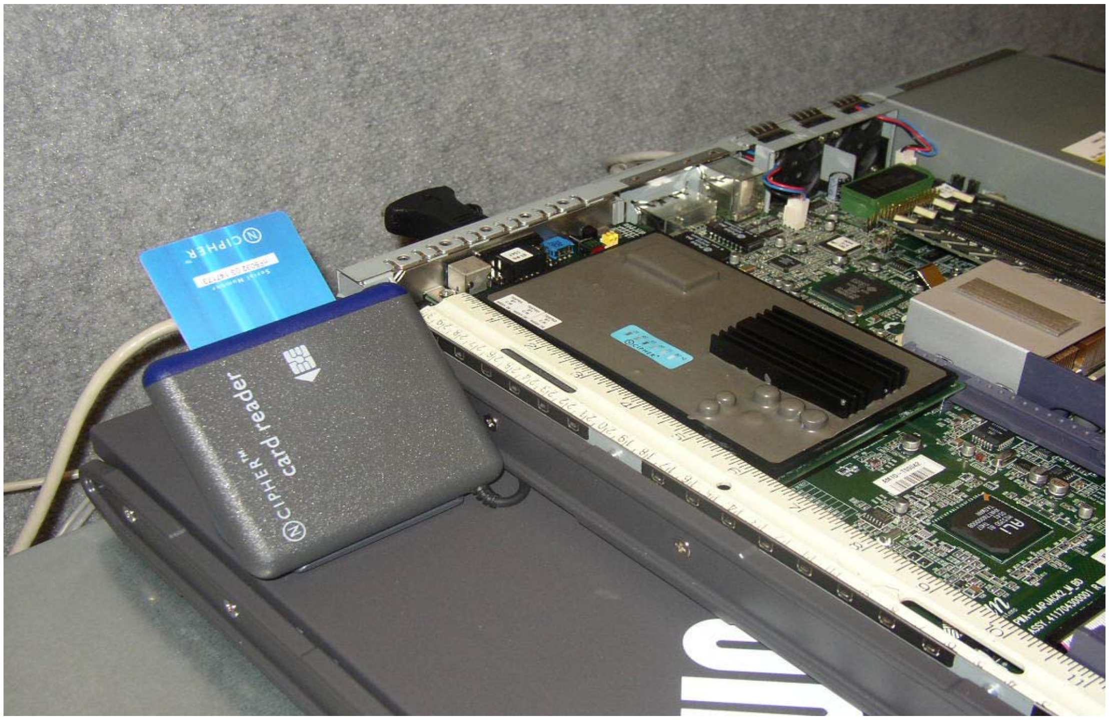


C.4 Rainbow Architecture

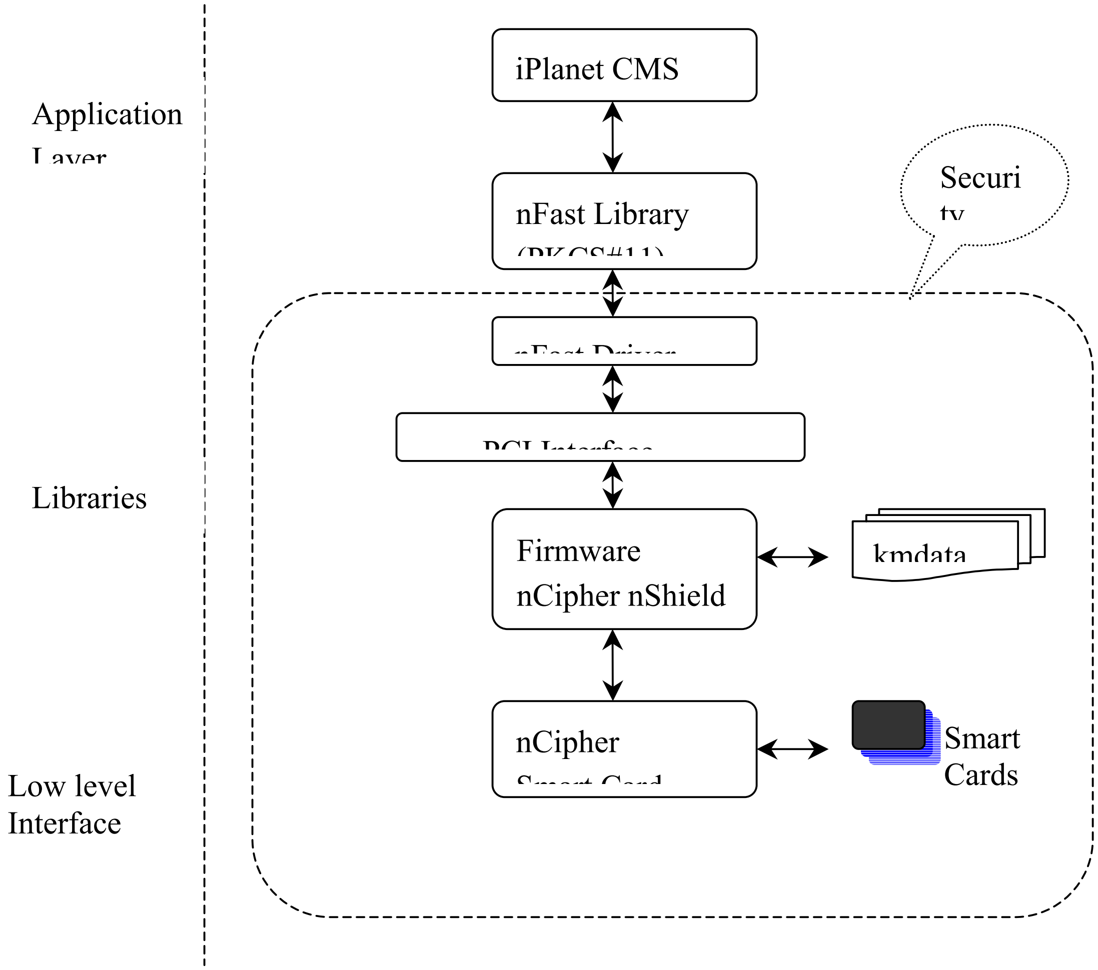


C.5 nCipher Components





C.6 nCipher Architecture


Application
Layer
Libraries
Low level
Interface
iPlanet CMS
nFast Library
(PKCS#11)
Security
nFast Driver
PCI Interface
Firmware
nCipher nShield
kmdata
nCipher
Smart Card
Smart
Cards