Martin, thank you so much for the great paper which is very helpful for understanding of general ideas. Having said that, let me put some questions and comments. And please forgive me if questions are too stupid.

RED marks text which I suggest to remove or alter.
GREEN marks text which I suggest to add
YELLOW marks comments, questions and explanations
CYAN marks text to which nearby comment relates

ENTERPRISE
CLASS
Slogan should be simply and clearly understood. "Enterprise class" is defined and can be understood at least as 1) something which serves a corporation, 2) something which can be a corporation itself, 3) something which commemorate the Enterprise NX-01 spaceship from Star Track saga. Eg Russian law about electronic signature differentiate PKI systems into 2 classes: public and corporative. This can form bad interference with this slogan.
I would suggest something more neutral and general, as one of the following:
1) BUSINESS LEVEL
2) PROFESSIONAL LEVEL
3) ULTIMATE LEVEL
4) HIGH CLASS

OPEN SOURCE
TRUSTCENTER
SOFTWARE

White Paper - Architecture Overview

OpenXPKI

2005-11-18 Draft Version 0.1 -Martin Bartosch

Introduction

OpenXPKI aims at implementing a complete and flexible Trust-center and PKI software that handles the entire workflow related to requesting, creating and delivering X.509 Digital Certificates.

Targeted environments range from small installations using software keys on a single machine to large scale and high performance multi-node deployments using enterprise class (see above) software and hardware, such as Oracle or DB/2 databases or Hardware Security Modules.

With strong emphasis on open standards and interoperability the system is engineered to be easily integrable with other standard compliant IT components.

Building on several years of experience with implementation, operation and maintenance in another Open Source PKI Project I would place here at least a footnote with a name OpenCA, because this name is more or less known and respected, the designers and developers of OpenXPKI took the chance of addressing the limitations of existing solutions. In order to obtain a clean and efficient solution, OpenXPKI was designed and engineered almost from scratch. The primary goal of the redesign was to create a system that is highly and easily configurable, modular and scalable,

- is highly and easily configurable,
- is modular,
- is scalable,
- has productivity which follows external load as far as to a certain high saturation level, and after that system stays at this productivity level capturing all input data without loss,
- is still offering a majority of regular features present in other high class (?) PKI products.

while still offering a majority of regular features present in other products.

As OpenXPKI is mainly written in the Perl programming language, it naturally takes advantage of the existing code base available freely at CPAN and employs many components that have been published there.

This documents describes the overall design principles and architecture of the OpenXPKI system as planned for the initial version 1.0.

System Architecture Overview

An overview of the system's architecture is shown on a picture. given in Figure 1. Once you refer to Figure 1, you need a figure capture with the same number under the figure. Also, it is not fully clear, what is the meaning of an arrows shown on the figure and why they are not bi-directional.

Users and applications using PKI services access the system by means of an interface implementation. For end users this will commonly be a web interface, administrators might choose to use a command line interface, whereas computer systems or network components will probably communicate with OpenXPKI e. g. via the SCEP protocol which is implemented by the SCEP interface.

It is possible to extend the system with additional and independent interfaces in order to support other user interfaces, future PKI standards or proprietary systems.

All interfaces use the common OpenXPKI Server API that defines a programming interface for communicating with the PKI system. The Server API itself is implemented by the OpenXPKI Server which is de-signed to run as a daemon process on a Unix system.

The OpenXPKI Foundation

Launched October 2005

The major goal of the OpenXPKI foundation is to support the OpenXPKI community to develop free PKI software. The foundation forms the organizational base of the project and supports the community with management of the project's name, license and infrastructure. This includes organization of support and promotion events like workshops and conferences. All structural decisions are made by a board that is elected in regular intervals by users and developers.

This central OpenXPKI Server process communicates with a an XML Configuration Layer that provides access to all configuration values for the system, e. g. CA configuration, process definitions or database configuration parameters.

All events that are processed or generated by the system are subject to extensive and configurable logging and auditing via a separate module. The Logging and Auditing module can log to flat files, Unix syslog or database tables.

We distinguish between two types of data in the system:

- Static data. This includes all configuration data, data describing logic of a workflow diagram of the system, and the CA's key material.
- Variable data. This includes all working data being created during system run. . In particular data that describe the state of the systems in a sense of the workflow diagram Log and audit information is not included is this type of data.

All state information variable data of the system is stored in a common database that is the canonical source for all variable information within the system. Because As there is no need for storing stateful information to store variable data within the server's file system, building a redundant PKI system or a high-performance cluster of multiple nodes processing the same set of data can be achieved quite easily.

Backup and restore requirements are reduced to a standard database backup (variable data) and a backup of the system's static data. configuration plus the CA's key material (static data).

Modular Design

One major design objective was to generate a system that is highly modular, allowing to easily extend the software with new cryptographic toolkits and algorithms, database drivers or interface implementations.

All system components that are inherently stateless are factored out of the OpenXPKI Server into standalone modules what modules do you mean here? Cryptolayer, DB layer, ...? that could also be used in other applications what applications? If some yet unknown applications, maybe remove the whole phrase about "other applications"?. Only those parts that actually glue the server components together and handle stateful information system variable data to be kept in the database are subsumed under the OpenXPKI Server module.

Every component of OpenXPKI, including the server, is distributed in a hierarchy of Perl modules it would be great to give a graphical or textual picture of this hierarchy, at least in details known today. When deploying the system to an actual OpenXPKI instance this means that OpenXPKI will be installed just like other Perl modules; only the startup scripts and configuration will have to be installed separately, allowing for a very simple and easily understandable deployment process.

A large set of atomic base functions and cryptographic or PKI related building blocks are encapsulated in modules, hence extension of the system with new functionality, protocols, algorithms or data formats is easily possible by reusing the existing modules and can be done without running the risk of affecting the rest of the system.

More complex operations basing on these atomic functions are cleanly separated in Server Commands, which have access to state information or objects stored in the database by means of a caller context. Could you please explain what it means at all.

Workflow Integration

A key feature of OpenXPKI is the complete separation of process logic from implementation details. To achieve this, OpenXPKI adopts the semantics of commonly known Business Process Modeling techniques:

All internal Why? Workflow can (and should) process external data exchange like sending approved req from RA to

remote CA and wait for the answer. PKI related operations are completely handled and monitored by a Workflow Engine.

On startup the OpenXPKI system reads all defined process definitions and builds a process repository from this data that is accessed by the Workflow Engine during runtime. Thus Workflow Engine keeps the logic of a workflow diagram in-memory.

Each submitted (or automatically generated) request triggers the creation of a corresponding Workflow Instance. This is done by Workflow::Factory module. Workflow Instance is nothing but a set of records in a database (including e.g. request data), which 1) trace current state of this Instance in terms of the workflow diagram, and 2) keeps all pre-history of the Instance, which is a track of this Instance across the workflow diagram.

We differentiate several types of system workflow diagrams, which are also types of Workflow Instances. Those include: workflow reacting on self-singed CR, workflow reacting on CRR, workflow reacting on CR with key-generation on the server-side, etc.

Workflow Engine on a regular basis disturbs each Workflow Instance to find out which state it is in, and what could be done to move this particular Instance further ahead along the workflow diagram of particular type. Technically, the Engine calls functions create_workflow() fetch_workflow() to create in-memory Workflow Object, which refers to the "active" Workflow Instance (that is the Instance being processed by the Engine this very moment). Besides all state-related data extracted from the Instance, the Object is equipped by the workflow logic valid for the type of this Instance.

By operating on this Workflow Instance the Workflow Engine tracks the state of the linked objects (e. g. request data). The Workflow Instance itself is persistently stored in the database and not only references all internal data objects required for processing the request (e. g. certificate request data) but also includes its complete modification history.

In other words the Workflow Instance itself knows what data objects it has to access to perform certain actions associated with this instance. Is it true? I thought that it is only Workflow Engine, who knows workflow diagram. Thus only Engine (or at least Object) can know, what particular Instance has to do next. .

Throughout the whole life time of the Workflow Instance it is possible to obtain all workflow related information by accessing the Workflow Instance object. This includes all operations that can be performed on the Instance object in its current state. See previous comment. This information directly influences the interface representation of the object and hence only displays operations that are sensible and actually executable for a user working with the object.

For example, a Certificate Request that has not yet been approved by a Registration Officer may still be modified. After referencing the Process Repository about the possible transitions from this state the Workflow Instance will thus indicate that "edit" is a valid operation on this particular object in addition to the "approve" action. The web interface will obtain this information from the Workflow Instance and accordingly display "edit" and "approve" buttons for this particular object.

Process Modeling

Just like the rest of the system configuration the Process Definition is stored in a XML format proprietary to OpenXPKI. This "nonstandard" representation was chosen because it is tailored for the use within OpenXPKI and only contains the necessary elements to model its processes. The resulting main advantage is that OpenXPKI Processes can be defined with simple tools (like a text or XML editor) that are readily available for anyone without requiring expensive closed source modeling tools.

For a quick start OpenXPKI will be shiped with a set of Process Definitions that will be sufficient for small to medium scale implementations and that can easily be customized.

Relations to External Business Process Modeling Software

The obvious disadvantage of using a proprietary process definition dialect is that interfacing with professional BPM software becomes more difficult.

As of this writing a number of process modeling dialects compete on the market, and as currently there is no agreed standard format of representing business logic, a simple and straightforward representation was chosen in favor of heavy weight standards like XPDL or BPML (which are hard to read and nearly impossible to compose manually).

However, in a future version support of external business process modeling tools that e. g. export XPDL, BPML or the ARIS XML format could be added via an XSLT processor that understands certain industry standard dialects and transforms them into the OpenXPKI representation.

XML Configuration

All static configuration of the OpenXPKI system is contained in a set of XML files that not only describes normal system configuration such as database setup, authentication methods, directory service support and of course CA definitions, but also includes a complete description of the PKI related workflows (process definitions).

As the configuration engine support XInclude, the user can decide whether to distribute the configuration across multiple files or to place all configuration in one single XML file.

For the entire XML configuration an XML Schema is supplied that allows for reliable verification of an actual configuration set.

In its first release the XML configuration will be read-only for the OpenXPKI system. All changes to the configuration must be done manually by the administrators e.
g. in an editor. In future versions a configuration editor frontend (e. g. a web frontend) may be added to the system, allowing to change configuration entries and write back the modified configuration (if allowed by policy settings).

During startup the server process reads the configuration files and caches the information throughout the lifetime of the server process. Queries to configuration will then be answered from an in-memory cache.

Configuration Inheritance

For most OpenXPKI installations beyond very simple demonstration setups the configuration will almost certainly require duplication of local configuration settings, such as CA token configuration or CA profile settings.

Configuration redundancy being one weak point of some of OpenXPKI's competitors, this shortcoming was addressed by introducing a powerful configuration inheritance scheme, allowing to inherit entire XML subtrees from another location in the XML configuration tree.

This key feature will ease initial configuration substantially, as the system will be distributed with a set of default configuration settings defining a very basic PKI installation.

Starting with the base (or default) configuration the user can then successively add local modifications that overwrite or extend inherited values from the default configuration.

As the local modifications are kept closely together in the inherited subtree, it is immediately visible which settings were changed related to the default values.

The inheritance feature will also prove extremely convenient not only for initial configuration but actually for maintenance and extension of productive systems.

It will no longer be necessary to cut & paste entire configuration sections to e. g. create a new issuing CA, but instead it will suffice to create a new subtree that simply inherits from the predecessor CA, only changing the necessary values such as CA certificate and key. By adding just a few lines of configuration whole system components can be extended with a fully working but slightly derivative version of the parent configuration.

As an actual OpenXPKI installation ages over time, maintainers will find this feature extremely useful as it reduces the differences between configuration revisions substantially and reduces the amount of configuration that must be reviewed by the administrator.

Inheritance also greatly simplifies global configuration of complex systems. For an installation using a larger number of CA instances that all inherit from a common default configuration, it is extremely easy to modify the definitions throughout all configured CAs simply by changing the corresponding values in the default configuration.

PKI Features and Implementation

Multi-CA Support via PKI Realms

An OpenXPKI installation may provide one or more externally visible (i. e. via an Interface) CA instances or PKI Realms. A unique PKI Realm Identifier is assigned to each PKI REalm that is used to distinguish between the individual PKIs.

Each PKI Realm provides its own independent name space in terms of profile, common name, serial number, access control etc.
This makes it possible to run different and completely independent CAs in one single installation without having to install multiple program installations.

Selection between instances for end users and administrator staff is delegated to either the web server running the CA installation (e. g. by using different URLs or port numbers for instance distinction) or alternatively may be performed by a selection mechanism on the PKI login screen.

A PKI Realm encompasses a complete PKI configuration set, including CA certificate, key, LDAP directory, Authentication, Authorization etc.

All PKI Realms can share the same database and are distinguished by the software on database row level.

Each PKI Realm may consist of an arbitrary number (zero or more) of issuing CAs. All these CAs should be capable of issuing certificates for the namespace defined by the PKI Realm. All CAs in a PKI Realm will usually use the same Certificate Profile and very similar configuration, but this is neither mandatory nor technically enforced by the OpenXPKI system. In particular, each CA uses its own CA certificate and private key.

Automatic CA Rollover

When reaching the end of a CA certificate lifetime there is a certain point in time after which no usable end entity certificates can be issued whose desired validity fully fits into the CA certificates validity.

To address this problem an automated CA Rollover is implemented. The basic idea is to have multiple issuing CAs (with overlapping certificate validity) that are logically responsible for the same set of end entity certificates.

OpenXPKI will automatically detect which CA to use for a given operation and use this CA to process the request.

On startup the OpenXPKI server examines all PKI Realms that are configured. For each PKI Realm OpenXPKI determines which issuing CAs belong to this CA instance and analyzes the corresponding CA certificate. The validity information of each CA certificate is stored internally for later use by the CA request dispatcher.

In order to make CA rollover work, administrators must make sure that an issuing CA exists that is capable of taking over the certificate issuance duties of the expiring issuing CA. If no Rollover CA exists, the CA system will not be able to find a suitable candidate CA for the requested operation and will stop working after the last active issuing CA exceeds a certain point in time after which it is not possible to issue end entity certificates with the required validity.

By using the automatic CA rollover feature it is possible to run an OpenXPKI installation without having to redeploy the system due to expiring CA certificates.

Modular Cryptographic Backend Support

Among Open Source projects OpenSSL is probably the most popular and advanced cryptographic toolkit, and hence OpenXPKI uses OpenSSL as its primary Cryptographic backend. In order to decouple cryptographic operations from implementation, the OpenXPKI system uses an abstraction layer that allows to replace the underlying cryptographic toolkit with alternative cryptographic backend implementations.

Hardware Security Module Support

In order to make the OpenXPKI system usable in high-security environments (such as Trustcenter installations), the software supports Hardware Security Modules in the cryptographic backend modules for protection of key material.

Hardware support will be included for nCipher nShield and Safenet LunaCA Hardware Security Modules and will also include SmartCard based environments via the OpenSC project.

Infrastructure and Interoperability

Database Support

OpenXPKI requires a relational database for data storage. In order to abstract from implementation differences, OpenXPKI provides a database abstraction layer that will include drivers for MySQL, PostgreSQL, Oracle and DB/2.

Adding support for additional databases is possible by writing a database driver for the required database engine.

Strictly limited for demonstration and for test purposes only the self-contained what does it mean? low profile SQLite database engine is supported as well by the system.

Multi-Node Capability

OpenXPKI will support distributed architectures where PKI tasks are split regionally (e. g. via local Registration Authorities) or functionally (e. g. separation of enrollment interface, registration interface and CA operations).

Functional separation in offline components (CA operations) increases security by making it impossible to attack the CA via the network. However, when using an offline CA it is necessary to exchange data between public frontend, registration authority and CA.

By encapsulating PKI data objects in Workflow Instances that transport their own state the Workflow Engine contributes greatly to making this task possible and reliable. I absolutely do not understand who make what in this sentence. Synchronization of distributed nodes not sharing the same database will be achieved by exchanging the Workflow Instance information and the referenced data objects.

Performance and Clustering

Even with high performance server hardware and Hardware Security Module support there will be an upper limit for the achievable certificate issuance rate. Although it is possible to parallelize many operations in the process, this will still not be sufficient for environments where a high number of certificates must be created within a certain time frame.

On the other hand, critical environments may require redundancy for maintaining a high level of availability.

Both requirements will be addressed by a feature that automatically distributes workload between distinct machines of a multi-node cluster setup. The main prerequisite for this to work is a shared database that can be accessed by all cluster nodes. (For high-availability setups this means that this database must be redundant and highly available as well, which is easily possible with standard features offered by enterprise scale databases such as Oracle or DB/2.)

In other words, here we can build up overall productivity by adding a parallel conveyor coupled to the same database. The price for thus increased productivity is longer processing time for each individual request, because a distributed database serving several conveyors will need more time to synchronize remote nodes of the database.

If a number of machines are configured to work together as a cluster, these systems automatically negotiate and designate a cluster manager node that takes the role of distributing workload packages to all cluster worker nodes. The cluster manager node itself will also assume a worker node role in addition to managing the cluster (unless explicitly

configured to only act as manager).

If a worker node fails the manager node detects this condition and redistributes the remaining workload of the failed machine among the remaining nodes.

Should the cluster manager node fail for some reason, the worker nodes will detect this after a timeout period and re-negotiate a new manager node among the remaining systems.

Adding a new node is as easy as setting up the hardware, installing the software and the common configuration and adding the system to the network. The new node will automatically be integrated into the cluster by the manager node.

As there is no upper limit of cluster nodes, the OpenXPKI system will scale extremely well. To compensate for higher throughput demands it will suffice to simply add more nodes to the cluster.

Internationalization and UTF-8 Support

OpenXPKI is designed to be multi-lingual from ground up. This does not only mean that the primary user interface (the web interface) will be available in several translations, but includes the fact that all internal data handled by OpenXPKI is fully UTF-8 compliant. As a result, request data or generated certificate may include special characters from text in special characters is a non clear term, do you meant regular characters from foreign language? literally any living human language in the world.

Authentication and Authorization Framework

Authentication and authorization is separated from the process logic and is implemented in a modular and extensible approach.

Both authentication and authorization can be handled internally (for smaller or stand-alone installations), but for integration in existing infrastructures interfaces for external authentication and authorization methods are provided.

External authentication methods include LDAP and Unix PAM.

Authentication methods can be stacked, i. e. an authentication request is sequentially checked against several authentication systems, making it possible to integrate multiple authentication systems used in the target environment.

Authorization will primarily be supported via LDAP, but additional methods can be added if need be.

Logging and Auditing

Logging and auditing is an important feature of security relevant software. OpenXPKI abstracts all logging operations via a standard Perl module that allows configurable logging targets (including Unix syslog, database or flat files).

A complete audit trail for object manipulation is available via the system log and the individual Workflow Instance history that keeps track of each operation performed on a certain Workflow Instance.

For each CA private key operation the audit system maintains a usage counter in the database that is incremented for each use. If supported by the underlying cryptographic tokens (certain Hardware Security Modules provide a private key counter feature implemented in t he syst em fir mware), t he OpenXPKI auditing system will compare the private key counter stored in the database with the HSM's private key counter, and raise alarm if needed.

If the expected private key does not match with the counter value obtained from the HSM, the audit system will raise an alarm via the monitoring system that a possible private key abuse has been detected.

Monitoring Integration

Monitoring systems can be integrated via a plugin mechanism maybe some explanation is needed, what it is? that is based on hooks that are called for certain events by the OpenXPKI system. Hence a monitoring integration can be as

easy as a set of shell scripts that act upon certain events or may comprise integration into professional Monitoring systems such as Tivoli or HP OpenView.

A future release of the OpenXPKI system will natively support at least one Open Source Monitoring solution, most probably Nagios, and it will also include a module that will be able to generate SNMP traps.

The Way Ahead

Although the system architecture outlined in this document is a very ambitious project, the developers are confident that a first beta version of the system will be ready by end of Q1 2006. This first version will include most of the features mentioned in this document.

A first stable release is planned for end of Q2 2006.