

Generic Support for PKIX Certificate Management in CDSA

Shabnam Erfani
WatchGuard Technologies
serfani@watchguard.com

Sekar Chandrasekaran
Microsoft Corporation
sekarcha@microsoft.com

Abstract

The Common Data Security Architecture (CDSA) from the Open Group is a flexible standard that defines APIs for security services needed for implementing Public Key Infrastructure (PKI). The emerging IETF Public Key Infrastructure (PKIX) standards provide certificate management protocols geared toward the Internet. The PKIX specifications define the expected behavior of the PKI, but do not provide abstractions that can be used by exploiting applications. In this paper we show the feasibility and design methodology of extending CDSA abstractions to support PKIX certificate management. To achieve this, we model a general, end-to-end system architecture based on CDSA that PKIX certificate management model, and discuss the merits of this system from the application and system architecture perspectives. We conclude the paper with a discussion of the resulted generic CDSA version 2.0 API that support PKIX certificate management model.

1. Introduction

As the applications exploiting public key infrastructure (PKI) grow more complex, the market demands integrated security services that are platform independent, scalable and standards compliant. Integration of PKI into distributed applications such as health care and E-commerce requires availability of basic security services such as cryptography, certificate management and secure storage services in various components of the system in a ubiquitous and transparent manner. Furthermore, as the application security models become more mature and distributed, provision of the basic security services needs to become configurable according to a given policy. For example, different countries (jurisdictions) exercise import/export policies for providing strong cryptography to applications. Or, an enterprise may need to impose a policy for key storage and backup that needs to be enforced transparently in the infrastructure. Last but not least, software maintainability mandates that applications not embed policies or the details of a particular PKI

protocol. Otherwise, if policies or protocols change the application has to be modified or rewritten. Clearly, appropriate services should be provided as an encapsulation and based on the configured policy enforced by a mediating layer. A good approach to address these issues is to provide a functionally rich set of security services in a controlled manner through a framework-based architecture such as the Common Data Security Architecture (CDSA). This is the goal and motivation behind our approach to integrate PKIX management services into CDSA.

CDSA defines a framework-based approach to providing basic security services as depicted in Figure 1. The framework layer defines Application Programming Interfaces (APIs) for each category of service that can be called by the application, and a matching Service Provider Interface (SPI). Pluggable service provider modules that implement the actual services support the SPI. The framework mediates and dispatches an API call to the appropriate SPI implemented by a selected service provider. This architecture not only provides a well-defined set of APIs and SPIs, but also transparent policy enforcement and integrity checking within the framework. This method separates the application from a particular service implementation since the application calls the same API, but different implementations (in the form of different service providers) can be selected at run-time. The services can be used to implement another layer that provides system security services as depicted in Figure 1. Framework architecture is also used in Microsoft Cryptographic API and Java Cryptography Architecture, and has proven merits.

CDSA provides 5 service categories to applications: Key recovery (KR), Cryptography (CSP), Certificate library (CL), Trust Policy (TP), and Data Library (DL). CDSA also provides APIs for module management and run-time discovery of service provider capabilities. Service provider modules benefit from the module management services in the framework. A service provider can invoke the services of another SP by calling that provider through the CDSA API. For example, a CL performing certificate verification can invoke a CSP to check the cryptographic signature on

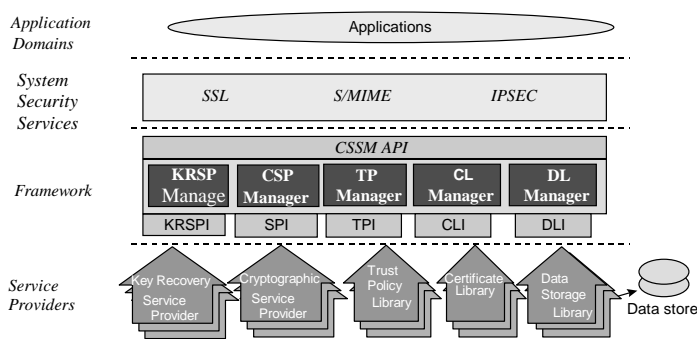


Figure 1. CDSA Architecture

the certificate. In this respect, the CL acts as an application to CSP.

In this paper, the CL and TP APIs are areas that we consider for further development. The CDSA version 1.2 defined a number of APIs in CL and TP that were appropriate for smaller, localized applications that needed certificate services such as issuance, trust policy validation and revocation. They did not permit distributed or asynchronous certificate lifecycle operations. The CL provided APIs for syntactic manipulation of the certificates and certificate revocation lists (CRLs), and the TP allowed certificate and CRL validation. This model however was not adequate for implementing the PKIX standards that are becoming dominant. Specifically, we felt a need for expanding the CL and TP abstractions to accommodate a PKIX-based certificate management model. This work has been a joint effort with Intel Architecture Labs and has been included in the CDSA version 2.0.

The Public Key Infrastructure Drafts (PKIX) from the IETF establish a model for certificate lifecycle management. They also define a common profile for X.509v3 certificates and certificate revocation lists (CRLs) geared toward the Internet, providing a common ground for interoperability. The discussion of PKIX requirements and architecture in this section is a distillation of [3]. The goal is to provide a background and set of requirements for the extending CDSA certificate and trust policy services described later in the paper. The existing APIs for data storage and cryptographic services are functionally rich enough to support implementation of a complex PKI.

The PKIX architecture is composed of three entities: the end entity (EE) or the client, the registration authority (RA) and the certificate authority (CA). Note that the RA presence as a separate entity is optional and can be combined with CA into one entity (certificate server) if needed. Without loss of generality, in this paper we assume that the RA is a separate entity. The RA acts as a broker between the CA and the EE, however, the EE can contact the CA directly as well in the PKIX model. Typically, the RA performs tasks such as EE authentication, policy verification, key gener-

ation, and archival, etc. The CA in fact relies on the RA to perform almost all the necessary checks on incoming requests. The CA acts only as the signer of certificates and CRLs. This division of labor has several benefits:

- The CA can work in a protected manner
- The CA can be generic and applicable to many installations of the RA
- The RA can be customized for various policies and business workflow and CRL distribution
- Multiple RAs can be present in the system, permitting scalability and distribution of the workload

The PKIX certificate management model, however, does not clearly divide the labor between RA and CA. It is up to the system administrators and designers to determine which functions are performed by which entity in the PKI. In the current state of PKIX, the EE configuration is performed out of band. The RA initialization and certification process (enrollment) by CA is also performed statically out of band. In future, PKIX may provide protocol primitives to perform RA enrollment programmatically. The PKIX certificate management is composed of a number of stages as summarized below:

1. CA establishment where the CA is installed and configured independently or as part of a hierarchy with CA Keys. The CA certificate is published in the directory and CA policies are established. PKIX does not specify how this operation is accomplished.
2. EE initialization involves the following steps:
 - Request and import a root CA public key certificate (out of band)
 - CA/RA issues a secret values (initial authentication key) and reference value that is transported to EE via an out of band channel
 - EE Key pair may be generated elsewhere but has to be transported to EE by some means
3. Certification results in creation of a certificate for an entity and is composed of the following steps:
 - Initial registration/certification
 - Key pair update
 - Certificate update
 - CA key pair update
 - Cross certificate request
 - Cross certificate update
 - Certificate confirmation

4. Certificate/CRL discovery is the step taken to publish or distribute the issued certificate or CRL. PKIX provides messages for publication, or the PKI can use the directory to publish the certificates and CRLs.
5. Recovery operations are part of PKI management and are needed when an end entity loses its Personal Security Environment (PSE). The PSE is the collection of EE keys, certificates and trusted roots that are essential for performing secure operations. The recovery operation typically is performed to recover the client key archived or backed up at RA or CA.
6. Revocation is done when an authorized party requests the CA to revoke a certificate. This request results in a change to the CRL and possible publication of the CRL.
7. PSE operations include actions such as changing the pin and key backup that are not in the scope of PKIX.

PKIX defines a certificate management protocol (CMP) that enables some the stages described above. The CMP defines protocol messages for initial registration and certification, certificate publication, and revocation. Other stages are either considered out of the scope of PKIX, or are under development. Furthermore, PKIX also mandates EE authentication and proof of possession (POP) of the private key by the RA/CA. The details of how these operations are performed can be found in [3].

The PKIX certificate management model is an enabling technology that is moving to become standard and widely accepted. Therefore, it is desirable to provide support for PKIX in security infrastructure architectures such as CDSA. There are more advantages to building support for PKIX in CDSA. PKIX is still evolving, therefore hiding the implementation inside a service provider as done in CDSA is highly desirable. Furthermore, CDSA allows separation of PKIX certificate management model and the actual implementation of the protocol. The system can be designed such that the application is aware of the certificate lifecycle, but the service provider encapsulates the underlying management protocol. Additionally, other management protocols also can be implemented within the service provider without affecting the application drastically.

In the rest of this paper, we describe the methodology we used to design the CDSA abstractions needed for PKIX certificate management support. We present a model for a generic system that uses PKIX entities to enable certificate management. Then, we further show how the model can be built using CDSA as the infrastructure and the properties that CDSA should satisfy to enable the construction of the model. Based on the developed model we infer the requirements and the design pattern that we need to build the relevant API in CDSA. We also present an analysis of

the system properties and then describe the generic CDSA APIs that neatly enables support for the PKIX certificate management model.

2. System Architecture Model

The three entities in the PKIX model exploit cryptography, secure storage for keys, certificate management and trust policy services to accomplish their tasks. Therefore, CDSA is an ideal candidate for providing infrastructure services for these entities since it provides a rich set of APIs for each category. PKIX requires these entities to support a wide set of cryptographic algorithms for encryption, MAC and digital signatures. The current set will grow as new algorithms are introduced. CDSA Cryptographic Service Providers can easily satisfy this requirement. Furthermore, by encapsulating trust policy and certificate generation capabilities into CDSA TP and CL, CA and RA can be easily combined into one entity. The most important task of RA is to perform policy checks and user authentication on various incoming requests. CDSA service providers can be designed to encapsulate such functions. If the RA for some reason becomes optional, the service providers can be moved to CA and exploited there. Furthermore, the CDSA services can be exploited for key management. For example, PKIX defines primitives for user key recovery and update. The key recovery APIs and service providers in CDSA can be exploited to implement these required services in PKIX.

Figure 2 below depicts how the system composed of EE, RA and CA can be modeled using CDSA API and service providers. Note that we only discuss the part of the architecture that is relevant to the PKIX flow and its interaction with CDSA APIs. Other internal details of EE/RA and CA implementation (policy configuration and contents, permanent storage for audit records, etc.) and system issues are not considered in this paper.

As illustrated in the figure, each entity can be implemented as a layer on top of CDSA taking advantage of the underlying service providers. The components of the architecture are:

1. **End Entity:** The End Entity (EE) is the client side of the system where most of the requests are generated. The EE application places a call to TP API to make a request (authenticated or unauthenticated) from the PKI. The service provider in turn uses the PKIX protocol to submit a request to the PKI. The box labeled as the PKIX protocol manager on the EE side is composed of a set of libraries that encapsulate mechanisms for creation and parsing of PKIX protocol messages. We have separated the EE components from the PKIX protocol handler to decrease architecture dependency

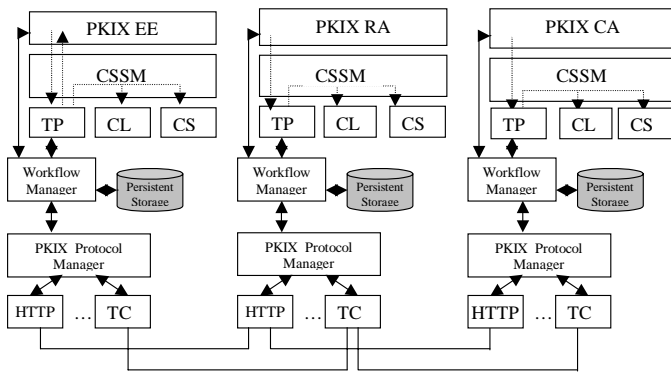


Figure 2. System model based on CDSA for implementing PKIX entities

on PKIX and increase reusability of the system components.

2. **Registration Authority:** The RA entity acts as the intermediate node between the EE and the CA. If the RA is present in the system, it must be certified by the CA and be configured with appropriate certificates, private keys and policies that it enforces for that CA. We assume that this information is established during RA initialization. Similar to EE, the RA relies on a separate component to interface with other PKIX entities (CA and EE), the PKIX Protocol Manager. The PKIX RA can be a separate process initiated by a master RA that dispatches various requests to the appropriate subordinate RA. The RA design is composed of an application layer that sits on top of the CDSA framework, and takes advantage of TP, CL, and CSP modules that provide lower level PKIX services. The TP module encapsulates all the trust primitives needed for validation and verification of incoming requests. The CL module provides the syntactic manipulation capabilities for certificates, and finally CSP can be used for cryptographic functions. A DL module can be used to access LDAP, or to store records needed for auditing, etc. at RA.

Since the presence of an RA is optional, one can move all of the RA functionality to the CA system by incorporating the RA application properties within the CA application and making the RA-preferred providers (TP, CL, CSP and possibly DL) available on the CA system. Using this mechanism, combining an RA and CA would involve attaching and using the RA-preferred service providers and the CA-preferred SPs on a single CA system. The CDSA-based architecture easily supports migration of system configuration. Another aspect of RA architecture is interoperability with different CAs. It is desirable to have an RA that not only works with a PKIX CA but can also interoperate with

non-PKIX CAs. We believe using CDSA in the architecture will significantly reduce the complexity of achieving this goal. The fact that the PKIX protocol handling (the message format and message forwarding operations) are encapsulated inside the TP module allows the RA application to interface with non-PKIX CAs by attaching a TP module that can handle a different protocol while preserving the semantics of the certificate management model.

3. **Certificate Authority:** The CA entity in terms of architecture is quite similar to the RA. The main functionality of CA is to provide a signing facility where the approved requests are processed and returned to the source of the request, EE or RA. Similar to RA, the CA relies on a PKIX protocol manager to manipulate PKIX messages and a master CA to dispatch the requests to other peer CAs. The PKIX CA application is a layer on top of CDSA that takes advantage of service provider services. Similar to RA, most of the trust and certificate operations are encapsulated inside CL and TP service provider. By specializing the SPs, and allowing the PKIX CA application to select different service providers, a PKIX CA can interoperate with non-PKIX CAs.
4. **PKIX Protocol Manager (PPM):** The PKIX protocol manager is a logical entity that provides PKIX protocol management and queuing services (through the depicted persistent storage) for the three PKIX entities EE, RA and CA. Note that physically at least between the RA and CA the PPM can be shared if needed. Moreover, the PPM can use any generic storage facility to queue messages and store other transport layer information where the application and Service Provider can access them. The PPM, however, does not provide PKIX session management and state services. All state information pertaining to PKIX request sessions is managed by the workflow manager. The PPM provides PKIX specific services such as message encoding and verification. To provide a transport, it relies on another underlying layer that provides transport protocol services. As a result, the PKIX messages can be encapsulated in various transport protocols such as HTTP, TCP, SMTP, etc.
5. **Workflow Manager (WM):** The PKIX model relies on exchange of requests and responses to determine the state of every session and object in the system. The state information can be managed by a separate entity that we call the Workflow Manager. This component is either a generic or customized component. The WM is responsible for keeping track of the state of work objects in the system. Object structures

are defined based on the events they must record, the creating entity, the current state and the entity where they are located. The work objects can be persistent or not. The workflow manager relies on a persistent storage that contains information that could be used for system recovery and auditing as well. Moreover, the unit can be used to implement a customized CA process. For example, if a CA only operate offline, the WM on the CA can be designed to batch the incoming requests, and send them to CA when it is online and send the responses back when available.

The workflow manager and the protocol manager cooperate to provide session management and message forwarding on behalf of the application. These two are reusable components on all three entities. The basic flow in this architecture starts when the application calls a TP API to place a PKI request such as certificate request, renewal or revocation. The TP performs policy checks on the request and forwards it to the workflow manager. At this point, the request becomes a workflow object that can be transported to another platform and contains a state. The object state and other relevant information are updated as the protocol messages are exchanged between entities. This architecture allows clean encapsulation of functions within the service providers. It also provides a model for development of APIs with the transactional properties required by PKIX. Also, the architecture insulates applications from the details of the PKIX protocol. The application knows about the states in the lifecycle of requests and objects, but not the underlying protocol. Therefore, the PKIX service providers could be updated or replaced with minimal impact on the application.

3. CDSA Trust Policy APIs for PKIX

The PKIX certificate management model assumes certain properties in the system that need to be addressed by the designed APIs. There are other properties that are nice to be designed into the system that enhance interoperability and scalability. Furthermore, since PKIX is an evolving standard, there has to be backward and forward compatibility in the APIs to minimize the impact on applications.

The PKIX transaction model is designed to cater to both centralized and distributed systems. It also allows for either synchronous or asynchronous operations. The Certificate Management Protocol (CMP) includes authenticated and unauthenticated flows, depending on how the system decides to do certain operations. For example, depending on how the system is designed to perform POP (directly or indirectly) the CMP flow may or may not be authenticated. Also, the CMP allows batch messages and responses.

These requirements introduce other issues that need to be taken into consideration when designing a suitable API. Should the API support polling, callbacks or event notification? How do we incorporate the appropriate transactional properties into the API to enhance performance and scalability while supporting PKIX requirements? Given the CDSA design patterns, we addressed these requirements by defining various primitives that helped create an elegant and generic set of APIs that not only accommodate PKIX, but also can be used to support non-PKIX protocols. These primitives are:

- **Caller authentication context:** This data structure encapsulates the input parameters, method and timing of the caller authentication process. This context is supplied to the API from the application to be used for the requested operation. The context allows applications to control the granularity of the credentials usage. The same context can be used across many API calls, or can be change between calls. The information in the context can be used locally to authenticate the caller or can be passed on to the protocol manager to perform signing of PKIX messages, or be transported to another entity.
- **Groups and Sets:** We introduced groups of objects that can be used for batch processing. Defining structures that support groups of objects such as CRLs, fields and requests supports the PKIX requirement and also allows scalability and interoperability in the container objects.
- **Operation Contexts:** This data structure is defined for certificate operations such as certificate verification similar to cryptographic contexts. The verification context encapsulates all the necessary parameters for certificate validation such as the trust roots, CRLs, parsed certificates, etc. Introduction of the context collects all the relevant information into a manageable container object that can be used across APIs. It can also act as storage for caching information.
- **Transaction Model:** : We designed a granular polling mechanism for supporting distributed operations. This model allows use of the same API for both local and distributed operations and is suitable for PKIX model. When a request API is called, a transaction reference identifier and an estimated time is returned to the application. The application can use these parameters to call the corresponding receive APIs to get the results back.
- **Protocol Sequence:** We observed that almost all PKIX CMP messages follow the same pattern:

- The application submits a request on a client entity such as EE or RA
- The client application retrieves the response from server
- The client application confirms the response if needed
- The server application awaits the confirmation from client before closing the session

Many data structures also turned out to be common for multiple service types. These patterns were used to design generic APIs that support multiple service types.

The generic APIs fall into three categories: Submit, Retrieve and Confirm. The Submit interface accepts service-specific inputs and a service type to perform. The Retrieve interface outputs service-specific results. Each of these interfaces can be authenticated (get a caller authentication context as input) or not. The Confirm interface allows confirmation of the results from the client to the server. This function also can be authenticated or non-authenticated. Each category interface also receives an input that specifies the requested service type. Based on this parameter, the service provider can identify the data structure types passed in. For the sake of interoperability and following the PKIX certificate management model, the following certificate service types and the corresponding data structures have been defined:

- **Issuance:** for requesting certificates. Certificate renewal can also be accomplished using this service type.
- **Revocation:** for handling certificate revocation operations requested by authorized parties
- **Verification:** enables remote certificate validation operations
- **Notarization:** specifies data notarization services
- **Key Reclaim:** enables recovery of a private key associated with an issued certificate
- **Hold and Release:** allow changing the state of a certificate. For example, using this service type an authorized party can suspend or resume a certificate

The API also provides a CRL issuance service type. Full details of the API and supporting data structures can be found in [1], however, we give an overview of the API functions:

- **TP_SubmitCredRequest**(
CSSM_TP_HANDLE TPHandle,
const CSSM_AUTHORITY_ID *PreferredAuthority,

```
CSSM_AUTHORITY_REQUEST_TYPE RequestType,
const CSSM_TP_REQUEST_SET *RequestInput,
const CSSM_TP_CALLERAUTH_CONTEXT *CallerAuthContext,
sint32 *EstimatedTime,
const CSSM_DATA_PTR ReferenceIdentifier)
```

- **TP_RetrieveCredResult**(
CSSM_TP_HANDLE TPHandle,
const CSSM_DATA *ReferenceIdentifier,
const CSSM_CALLERAUTH_CONTEXT *CallerAuthCredentials,
sint32 *EstimatedTime,
CSSM_BOOL_PTR ConfirmationRequired,
const CSSM_TP_RESULT_SET_PTR RetrieveOutput)
- **TP_ConfirmCredResult** (
CSSM_TP_HANDLE TPHandle,
const CSSM_DATA *ReferenceIdentifier,
const CSSM_CALLERAUTH_CONTEXT *CallerAuthCredentials,
CSSM_TP_CONFIRM_RESPONSE *Responses,
const CSSM_AUTHORITY_ID *PreferredAuthority)
- **TP_ReceiveConfirmation** (
CSSM_TP_HANDLE TPHandle,
const CSSM_DATA_PTR ReferenceIdentifier,
CSSM_TP_CONFIRM_RESPONSE_PTR *Responses,
sint32 *ElapsedTime)

4. Conclusion

In this paper, we presented an overview of CDSA and PKIX, two standards that provide infrastructure services to PKI enabled applications in different forms. We described the rationale and a model for implementing the PKIX certificate lifecycle management in systems that take advantage of CDSA for providing infrastructure security services. Observing the patterns in the PKIX model we described how the APIs that support PKIX have been designed as part of CDSA version 2.0. This architecture unification allows a more coherent design for the PKI infrastructure of secure applications. Future work entails implementation and evaluation of the CDSA based system and introduction of new APIs as more PKIX services become available

5. Acknowledgements

This work is the result of cooperation of two teams at IBM and Intel Architecture Labs and funding from IBM Internet Division. We would like to thank and acknowledge the following people for their contributions to the API

development, and their thoughtful review and feedback on this paper: Denise Ecklund and Marion Shimoda from Intel Corporation who did considerable amount of work on collecting requirements and revisions of the API, as well as Ian Morrison and Sohail Malik from IBM Corporation. We would like to also thank the members of the IBM Vault Registry team – Hatem Ghafir, Tom Gindin, Khalid Asad, Mark Fisk, and Dave Rusnak – for their excellent review and suggestions on development of the system architecture model.

References

- [1] Common Data Security Architecture Specifications, version 2.0, [HTTP://www.opengroup.org](http://www.opengroup.org)
- [2] PKIX RFC 2459. <http://www.ietf.org/html.charters/pkix-charter.html>
- [3] PKIX RFC 2510. <http://www.ietf.org/html.charters/pkix-charter.html>