SKS (Secure Key Store)
API and Architecture

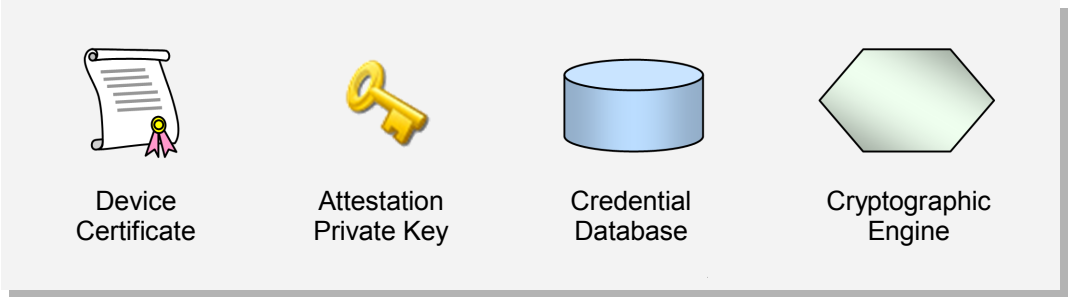
Table of Contents

Introduction

This document describes the API (Application Programming Interface) and architecture of an electronic component called SKS (Secure Key Store). SKS is essentially an enhanced smart card that is optimized for on-line provisioning of cryptographic keys and associated attributes.

Architecture

Below is a picture showing the components in the SKS architecture:



All operations inside of a SKS is supposed to be protected from tampering by malicious external entities but the degree of protection may vary depending on the environment that the SKS is running in. That is, an SKS housed in a smart card which may be inserted in an arbitrary computer must keep all data within its protected memory, while an SKS that is an integral part of a mobile phone processor may store credential data in the same external Flash where programs are stored, but sealed by an SKS-resident “master key”.

The *Device Certificate* and the associated *Attestation Private Key* form the foundation for the mechanism that secure provisioning of keys, which remains secure even if the surrounding middleware and network are unsecured.

The *Cryptographic Engine* performs in addition to standard cryptographic operations on private and secret keys, the core of the provisioning operations which from an API point-of-view are considerably more complex than the former.

Provisioning API

The core of SKS is the cooperation between three associated systems: The KeyGen2 protocol, the SKS architecture, and the provisioning API described in this document. i.e. *these items must be matched* in order to create a secure and interoperable system. A question that arises is of course how compatible this scheme is with respect to current protocols, APIs, and smart cards. The simple answer is: NOT. The reason why SKS still makes sense is that none of the common protocols, APIs and smart cards actually support secure on-line provisioning to end-users because *the current generation of smart cards are almost exclusively personalized by fairly proprietary software used by specific card administrators or by automated production facilities*. It is clear that mobile phones need a scheme that is more consistent with the on-line paradigm since SIM-cards due to operator-bindings do not scale particularly well. *“On the internet anybody can be an operator of something”*.

“Key User” API

In this document Key User API refers to operations that are used by security applications like SSL-client-certificate authentication, S/MIME, and Kerberos (PKINIT). The Key User API is not a core SKS facility but its implementation is anyway RECOMMENDED, particularly for SKSes that are featured in connectable containers such as smart cards since card middleware have proved to be a major stumbling block for wide-spread adoption of PKI cards for consumers. The described Key User API is fully mappable to the subset of CryptoAPI, PKCS#11, and JCE that most existing PKI-using applications utilize.

An extension facility in the Key User API supports Information Cards which can be provisioned through KeyGen2 together with matching X.509 credentials for primary (IdP) authentication.

The Key User API does not use authenticated sessions like featured in TPM 1.2 because this is a local security *option*, while the provisioning API has its own self-contained (mandatory) authentication scheme.

If another Key User API is used the only requirement is that the things that are provisioned by the Provisioning API, are compatible.

Return Values

Method List

http://webpki.org

Name	Type	Comment
status	byte	See Return Values

SKS_Activate [24]

Input

Name	Type	Comment
status	byte	The actual CyberLink command
		2
		Sista raden

Output

Name	Type	Name
	byte	
aa		
		aa

This is a sample API method m

SKS_Deactivate [25]

Input

Name	Type	Comment
status	byte	See Return Values

Output

Name	Type	Comment
	byte	
aa		

This is a sample API method.

You may call it from SKS_Activate hhh