Using Unmanaged Keys

Integration Guide

CryptoServer

utimaco®

Imprint

Copyright 2020	Utimaco IS GmbH
	Germanusstr. 4
	D-52080 Aachen
	Germany
Phone	AMERICAS +1-844-UTIMACO (+1 844-884-6226)
	EMEA +49 800-627-3081
	APAC +81 800-919-1301
Internet	https://support.hsm.utimaco.com/
e-mail	support@utimaco.com
Document Version	1.0.0
Date	06/10/2025
Status	PUBLISHED
Document No.	IG-2025-0004

Table of Contents

1 Version Information

Partner Name	
Product Name	Using Unmanaged Keys
Partner v[Version]	
Document Type	Integration Guide
Utimaco Product	CryptoServer
Utimaco v[Version]	
Document Image	
Document Number	IG-2025-0004
Document Version	1.0.0
Status	PUBLISHED

2 Introduction

CryptoServer is the Utimaco family of general-purpose hardware security modules (HSMs), i.e. physically protected, specialized computing units, designed to perform sensitive cryptographic tasks and to securely manage cryptographic keys and data objects. General-purpose HSMs from Utimaco aggressively protect the cryptographic identity - the digital keys - of your enterprise and your digital wallet.

The CryptoServer lines are certified by NIST, to FIPS 140-2 Level 3 (the Se Gen2), or Level 4 for physical security, Level 3 overall ("Level 3+", the CSe). Additionally, the CP5 variant is certified to EAL4+ for Common Criteria, according to EN 419221-5 *Protection Profiles for TSP Cryptographic Modules*.

The HSMs themselves are standard format PCIe (1U, full height, half-length) cards, and can be supplied in two form factors: Either as the card itself to OEM customers, or as clusterable 1U, 19" rack-mount appliances for use in HA/FT environments by end-users.

The PCIe form-factor does *not* come with server software; it can only be accessed via libraries on the workstation/server where it is installed.

SecurityServer is the software, firmware and modules that drive the CryptoServer hardware security modules. Normal integration is via standards-based or proprietary APIs (PKCS11, CNG, etc, and Utimaco's CXI), supplied with the SecurityServer package. These APIs allow you to make use of the HSMs abilities from within your business logic. Additionally, Utimaco can provide different SDKs that an enterprise may use to create private software modules that run directly on the HSM. Use of the SDK allows you to protect your cryptographic keys *and* the IP that uses them.

These private modules can be either for performance optimization or to implement custom cryptographic algorithms or mechanisms - ie, to run your business logic directly on the HSM, to prevent exposure of intermediate artifacts, etc. Performance optimization allows multiple, chained cryptographic operations to run with a single HSM call, and custom cryptography is useful for Post-Quantum Cryptography or when non-standard curves or symmetric encryption or hashing is required. Available SDKs are for C- or Lua-based modules, or for PKCS#11 Vendor-defined-mechanisms.

In a CryptoServer-based security system, security-relevant actions can be executed, and security-relevant information (i.e., cryptographic keys) can be stored. Given its general-purpose nature and extreme programmability, the Utimaco CryptoServer CSLAN form-factor is used as a

universal, independent security component in heterogeneous computer systems, supporting multiple use cases, concurrently, and from different hosts.

Utimaco HSMs are priced according to protection level and performance, *not* by number of users, applications or algorithms available.

3 Use Cases

This integration guide discusses strategies, rather that provide direct integration steps.

The SecurityServer libraries provide two methods for storing and managing keys, internally (on the HSM) and externally. For this integration guide, these are collectively referred to as the 'managed keystores'.

Internally managed keys are stored in and read from the CXIKEY.db database. All internal keys -- regardless of which standard API is used -- are stored and used via the CXI module (the host-side API is a bridge to CXI).

Externally managed keys are stored in a flat-file database, someplace accessible to the hist libraries (SAN, RAID, local file system, etc).

Where an enterprise will store its cryptographic keys for use with the CryptoServer, depends on the architecture and use-case. Each application can use its own strategy for storage and use. A root CA ('certificate authority') will probably maintain the Root Key using the internal key store, while signing keys, subordinate/issuing CAs and such may store their keys externally. It all depends on the enterprise's security posture, policies and procedures.

Managed keystores are directly supported by the SecurityServer libraries, whether internally or externally.

Each of the managed keystores has limitations on the number of keys that can be physically or reasonably supported.

The internal database is limited to 8Mb of key data. Each key object (or "row" in the internal database) contains structural, meta and sensitive key information, so a "key" may be longer or shorter than expected, and you may be storing information like a certificate, etc with a key. While backup and restore of the database is simplified by the csadm/CAT tools, it is still an extra step to perform regular backups.

The external database is limited to the file-size limit of the storage file system in use. While this means that the number of keys is effectively unlimited, the reality is that the more keys you have, search, retrieval and key synchronization between applications can make larger key databases unwieldy. On the other hand, backup strategies are straightforward, and can be transparent to the user.

The rule of thumb is if an application has more than 100 keys to worry about, consider the internal database. If it has 10,000 keys to worry about, the external keystore is suitable.

But, what happens when you are providing keys to an IoT implementation, and you are talking about 100,000, or 1 million, or 100 million asymmetric key pairs?

In this case, you should consider managing the keys yourself, within your own application -- by making use of enterprise systems that are designed up front for this use case: RDBMS, or KMIP (Key Management Interoperability Protocol) systems.

3.1 KMIP Server-based key management

A KMIP-server based KMS (key management system) has most of the same features of an RDBMS, but is aimed at adding additional layers of security around the storage, use and protection of *keys*.

Utimaco provides a key management server called the ESKM.

The ESKM key management system can store 10 million keys+, and does so in a distributed high-available/fault-tolerant cluster of servers.

3.2 RDBMS based key storage

RDBMS systems are not designed for storing *keys*, but are designed for storing *data*. An encrypted key blob is indistinguishable from 'data'. And, they have backup and failover systems, are fault-tolerant, support load-balancing, and are optimized for search and retrieval.

An RDBMS (i.e, in MySQL, SQL Server, Oracle, etc) is not provided by Utimaco as an option, as a back-end for its external keystores.

Depending on configuration, an RDBMS-based solution can store more than 1 billion individual keys.

3.3 External keys

Custom key storage relies on 3d-party enterprise software for how the encrypted, external key "blobs" (binary objects) are stored, and how their life-cycle is managed. How a custom key storage methodology should be backed up/restored depends on the media or methods provided by the enterprise software, or by a 3d party.

If using custom keystores, the keys must be created as "external" (CXI_KEY_FLAG_EXTERNAL) keys.

When creating keys for use in an unmanaged environment, the keys should be created with the CXI_KEY_FLAG_EXTERNAL flag set, for the 'flag' (the first) parameter.

```
// create a key
PropertyList keyTemplate;
keyTemplate.setAlgo(CXI_KEY_ALGO_AES);
keyTemplate.setSize(256);
keyTemplate.setName("AES test key");
Key aesKey = cxi->key_generate(CXI_FLAG_KEY_EXTERNAL, keyTemplate);
// alternately, cxi->key_open(CXI_FLAG_KEY_EXTERNAL, keyTemplate);
```

Creating, generating or opening a key with CXI_FLAG_KEY_EXTERNAL is a common requirement, regardless of language (Java, C++, etc), and regardless of how you decide to store the key.

In CXI, the Key is a ByteArray and is already serialized, no other steps are needed to store it as a byte [] into a database row, etc.

In Java_CXI, the Key is a Key object, and you will need to call `<key>.getEncoded()` on it.

```
// create a key
KeyAttributes attr = new CryptoServerCXI.KeyAttributes();
attr.setAlgo(KEY_ALGO_AES);
attr.setSize(256);
attr.setGroup("test");
attr.setName("AES test key");
Key aesKey = cxi.generateKey(FLAG_EXTERNAL, attr);
byte [] extKey = aesKey.getEncoded();
```

In both these cases, the byte array should begin with 0x4B ('K').

For formats of serialized key blobs - which may be backup blobs, simple blobs, key handles, etc - please see the SecurityServer documentation.

4 RDBMS

To support BC/DR, the root, underlying requirement is: Not knowing when disaster will strike, how do we securely store a CryptoServer over long periods of time, without losing any secrets, and without needlessly running down the battery? What do we need to capture, to get back to operational with a minimum of downtime? And with a minimum of data or capability "gap"?

4.1 Key storage

There are three different methods that Utimaco provides for key storage: *Internal*, *External*, and *Custom*.

4.1.1 Internal key storage

Internal key storage uses the CryptoServer itself as the key storage medium. The keys are stored in the file `CXIKEY.db`, which can be seen via `csadm ListFiles`. The `csadm BackupDatabase` command is used to make an encrypted copy of the data in this file, locally, on the host where the `csadm` command was issued.

The `csadm BackupDatabase` command can be used for **most** database (`.db`) files on the CryptoServer - it will *not* make backups of the MBK database, for example. It can and should be used also for private database files, if you have custom modules, written using the various SDKs, which may create them.

There are additional methods for backing up `.db` files that `csadm BackupDatabase` cannot target. The MBK database, for example, must be **populated** from what is effectively the backup artifacts (`cf csadm help=MBKImportKey`). The original smartcards or keyfiles - or backups of them - that were provided to the CryptoServer in order to create the MBK internally are what you are using for the backups.

4.1.2 External key storage

External key storage uses the local filesystem (specifically, a filesystem that the host libraries can see). Depending on the API used, the use of an encrypted, external keystore may be transparent (managed by the SecurityServer libraries) or be managed (managed by your API-written code).

In both cases, making a backup of this keystore is as simple as using the operating system's "copy" command (Windows 'copy', Linux 'cp', or via the OS GUI file management front-end for cut+paste or drag+drop where supported/available).

4.1.3 Custom key storage

Custom key storage relies on 3d-party enterprise software for how the encrypted, external key "blobs" (binary objects) are stored, and how their life-cycle is managed. How a custom key storage methodology should be backed up/restored depends on the media or methods provided by the enterprise software, or by a 3d party.

For example, an enterprise application with millions of keys and certificates may decide to store them as B*OBs ("binary <size> objects") in an RDBMS table row. In this case, the RDBMS management methods should provide the ability to replicate, backup and restore the database, even in real-time.

If using custom keystores, the keys must be created as "external" (CXI_KEY_FLAG_EXTERNAL) keys.

The scripts below are broken down into multiple lines in order to improve readability. Where a command is meant to be typed as a single line, a trailing \ character is used to indicate that the line following is meant to be typed on the current line. Not all shells understand the `\[newline]` syntax to indicate `'next line continues here'`.

```
csadm \
Dev=288@10.19.29.200 \
LogonSign=bobo,:cs2:cjo:USB0 \
GetState
```

is equivalent to, and should be typed as:

```
csadm Dev=288@10.19.29.200 LogonSign=bobo,:cs2:cjo:USB0 GetState
```

where `'next line continues here'` is not supported.

5 KMIP

KMIP is an OASIS standard.

6 Further Information

This document forms a part of the information and support provided by Utimaco IS GmbH. Additional documentation can be found in the product bundle in the documentation directory. All Utimaco CryptoServer documentation is also available at the Utimaco IS GmbH website: https://support.hsm.utimaco.com

If you have questions about this document, suggestions for improvement on unclear wording, or see any typographical errors, please email support-cs@utimaco.com, with the subject line *Doc Comments on IG Unmanaged Keystore*.

While we cannot guarantee a targeted response to your questions or suggestions, they will be considered when generating later versions of this document.