SIGMA DESIGNS

SMP8634 Security

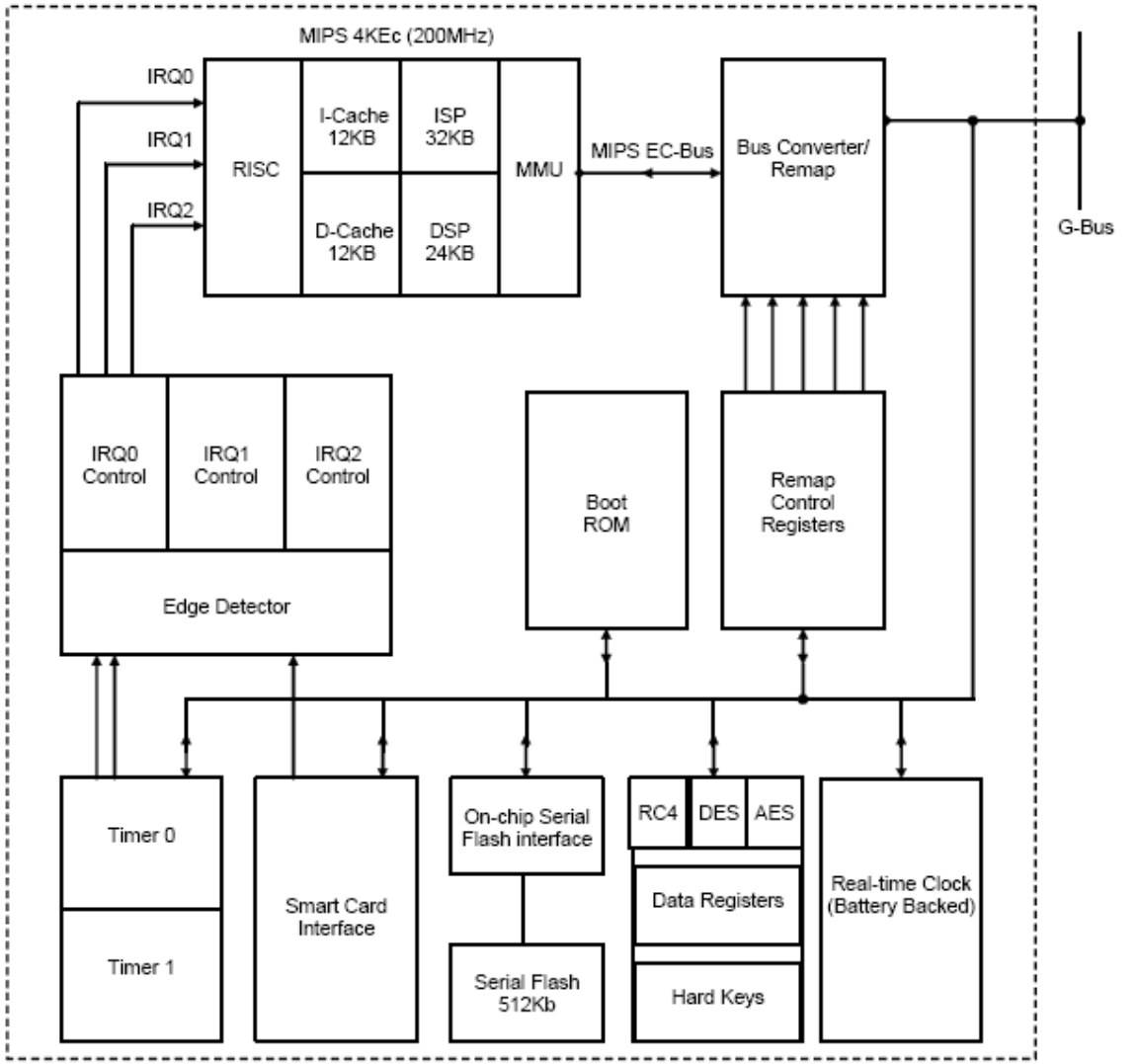
Security - Overview

- Dedicated security processor (XPU) for securely executing DRM/CA software including the following:
  - WM DRM (Janus and Cardea, provided by Sigma)
  - DTCP / IP for home networks (provided by Sigma)
  - CSS, CPPM, CPRM, AACS, BD+ (provided by Sigma)
  - NDS VGS, SecureMedia, Widevine, Irdeto for IPTV
  - ITRI for FVD players
  - ARIB, ATSC, DVB-CI, DVB-CSA, OpenCable conditional access
- Boot ROM: Upon hardware reset, the XPU begins executing from the BootROM.
- Embedded serial flash: Securely stores the XPU's operating system (xos) and the highly confidential secrets of each DRM/CA

XPU - Overview

- xboot : Upon hardware reset, the XPU begins executing from the Boot ROM.
- Embedded serial flash: Securely stores the XPU's operating system (xos) and the highly confidential secrets of each DRM/CA
- Data and Instruction Scratch Pads: Internal SRAM used exclusively by the XPU
- DRAM data can be encrypted
- Parallel flash data can be encrypted and digitally signed
- Secure data path: Creates a secure data path inside the chip to protect the decrypted data from being improperly accessed.

XPU Block Diagram



Security

- Secure CPU: Overview
- Secure bootloader
- Secure OS: xos

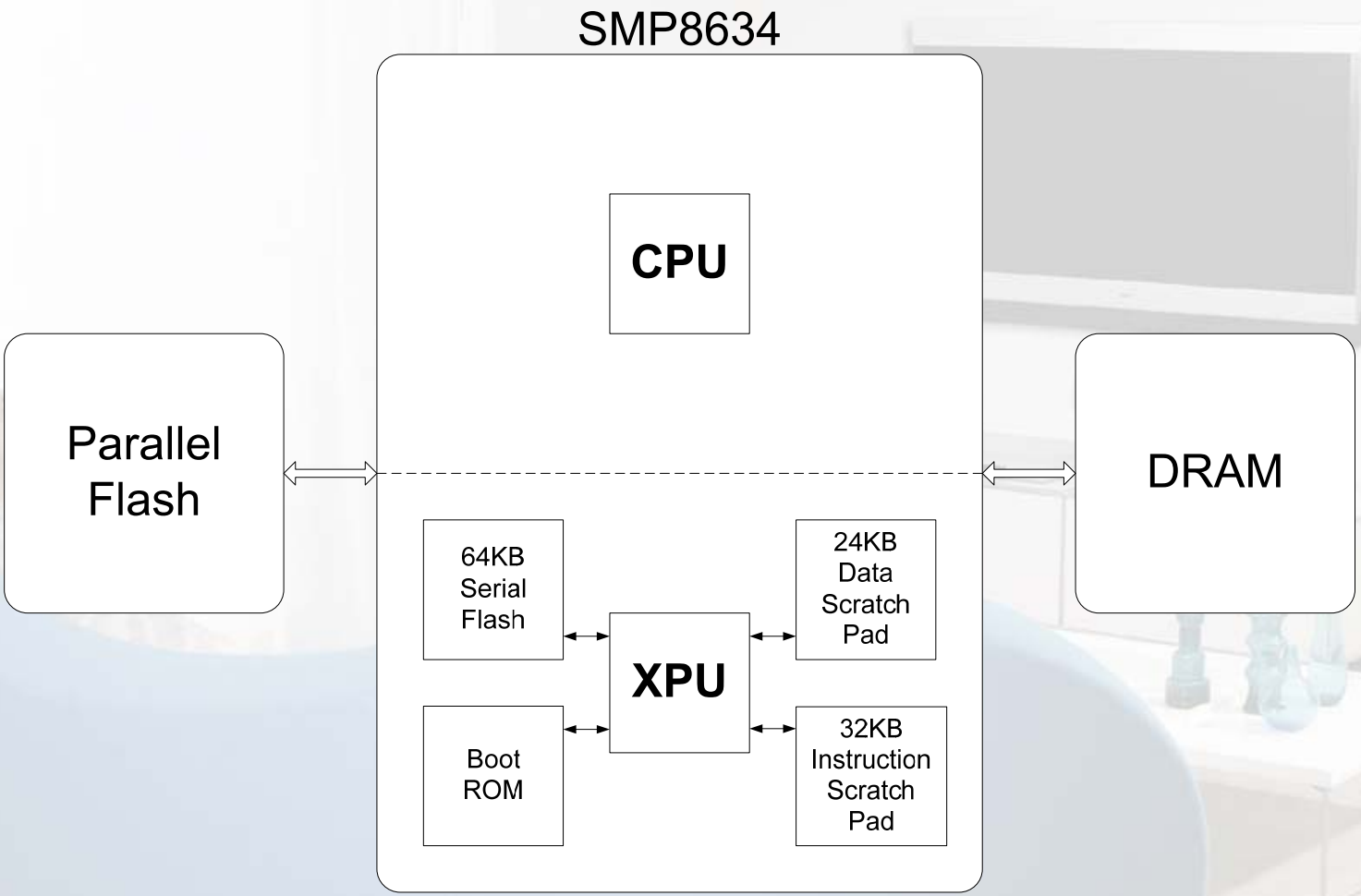
Secure CPU - Overview

- SMP8630 comes with two MIPS 4KE processor called cpu and xpu.
- The first processor will be used to run a standalone OS (CE, Linux with no special limitation)
- The xpu is used as a security processor, it is located in a privileged hardware block that has the ability to boot first and (un)lock all other SMP8630 hardware blocks by running properly authenticated code
- The xpu includes an instruction scratchpad of 32kB and a data scratchpad of 24kB

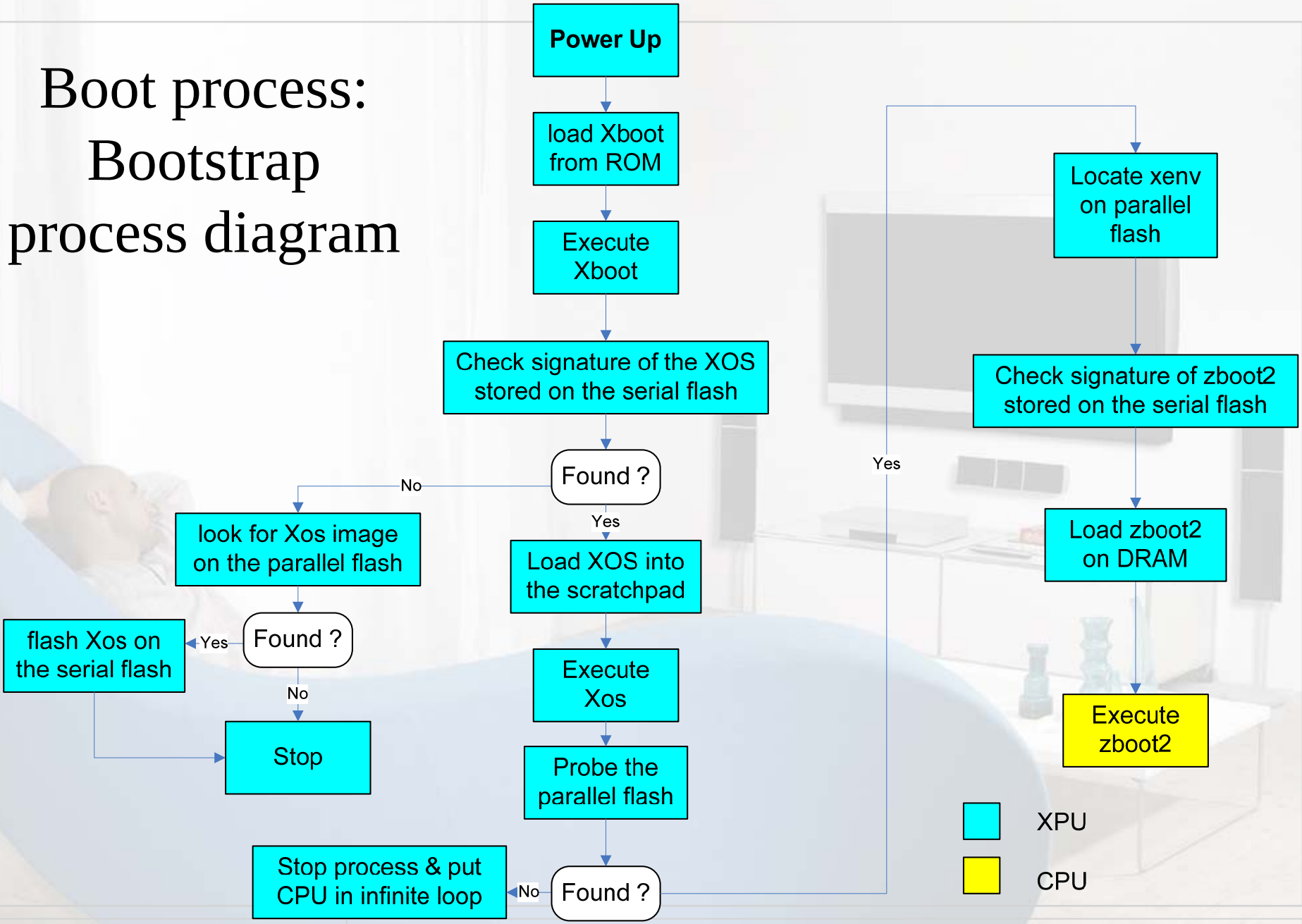
Secure SDK components

- Key hierarchy: Elaborate key hierarchy that let customers sign and encrypt their xtasks and OS kernels
- Secure Data Path: Create a secure data path inside the chip to protect the decrypted compressed data from being tampered.
- Xrpcs: Remote procedure calls between xpu and cpu and between tasks
- CipherAPI: xtasks can access to accelerated and romed cipher algorithms
- FIFOs models: xtasks are seamlessly integrated in the Hardware Library model.
- Embedded Serial Flash: xtasks can store private keys in the embedded serial flash without spying on each others
- Random Number Generator

Boot process: General framework



Boot process: Bootstrap process diagram



Secure Boot Loader

- SMP8630 contains two RSA 2048 public keys and two AES 128 bit keys (serial and parallel flash).
- xboot is located on masked rom inside the SMP8630
- xboot initializes the xpu
- xboot tries to boot first from serial then from parallel flash.
- xboot decrypts the content of the flash and then verify its signature (RSA-PKCS#1).
- The content of the flash is xos (secure OS).

Secure Boot Loader

- When xos starts, it tries to find zboot on the parallel flash by reading its signature.
- zboot is signed and can be encrypted and will run on the main CPU.
- zboot initializes the main cpu and loads the main bootloader (for CE, or YAMON for Linux).
- Main bootloader loads OS

XOS

- xos runs in kernel mode inside the instruction scratchpad of the xpu
- Multitasking OS with up to four xtask running in user mode
- Able to process secure remote procedure calls
- The timer may interrupt an xtask to schedule another
- xtasks interface with xos through system calls
- xos has the ability to upgrade itself in serial flash from a properly authenticated image in memory

xtasks

- 2 memory models:
- Small: all the code and data is loaded on the scratchpad (max: 12kB of code and 16kB of data)
- Small: code and data is deciphered and reciphered in place
- Small: context switch has an overhead of 10% because of encryption/decryption
- Large: code and data are located on DRAM

Xos ciphers

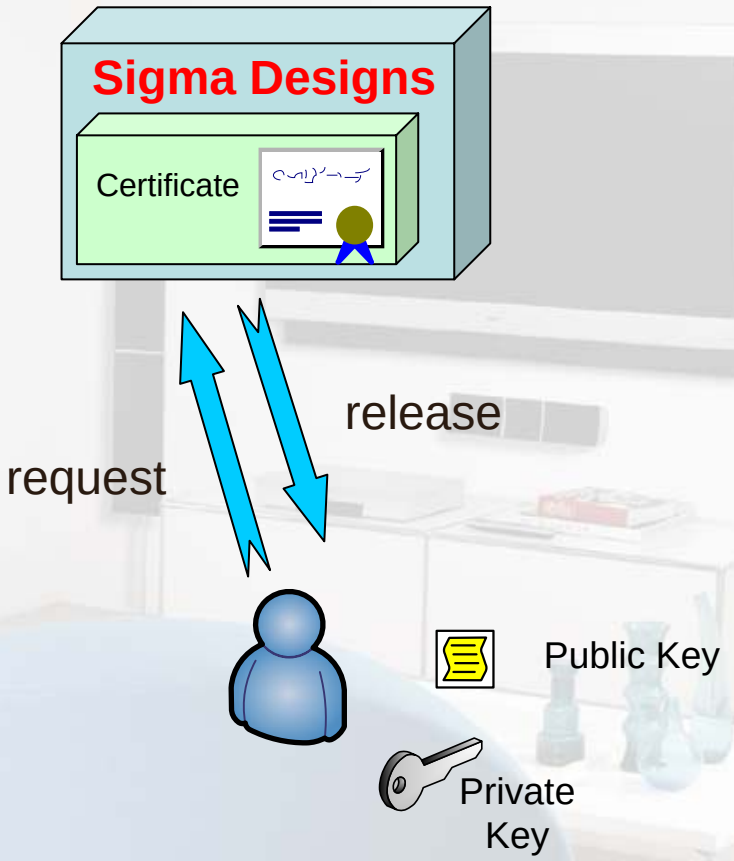
- Cryptography functions include:
- Hardware 3DES/DES block
- Hardware AES block
- Hardware RC4 block
- Software and rom-hosted RSA implementation
- Software and rom-hosted SHA-1 implementation

Certificates

- All the code running on the SMP8634 is signed and optionally encrypted.
- Sigma Designs is a certificate authority for the SMP8634 Public Key Infrastructure
- Sigma Designs issues public key certificates to customers and partners which states that Sigma Designs attests that the public key contained in the certificate can be trusted by the SMP8634.
- Sigma Designs also allows binding of public keys to the chips itself.

Certificate Generation

- Customer generates Public/Private Key pair
- Customer sends certificate request to Sigma Designs with Public Key
- Sigma Designs signs the Public Key using the Root private key and generates a certificate
- The certificate is released to the customer
- The customer uses the private key to sign code



Certificate and SDKs

- SDKs will include facsimile certificates and private keys for development chips
- For production chips, customers will have to generate unique certificates (for zboot2 and linux image)

Key Hierarchy

- Sigma RSA Root Private Key : [SigmaRSAPrivateKey]
- Sigma RSA Root Public Key : [SigmaRSAPublicKey]
- Sigma AES 128 Root Key : [SigmaAESSymmetricKey]

- XOS RSA Private Keys: [XOSRSAPrivateKey(n)]
- XOS RSA Public Keys: [XOSRSAPublicKey(n)]
- XOS AES Symetric Keys: [XOSAESSymmetricKey(n,Certificate)]

- SRDS : Sigma RSA digital signature (PKCS#1) using [SigmaRSAPrivateKey]
- CRDS : Customer RSA digital signature (PKCS#1) using [CustomerRSAPrivateKey]

Key Hierarchy

- [SigmaRSAPrivateKey] is stored in a FIPS Level 2 card and cannot be accessed by anyone. This is a 2048bit RSA key.
- [SigmaRSAPublicKey] is hardwired inside the chip.
- [SigmaAESSymmetricKey] is also hardwired inside the chip. This is a 128 bit key.
- [XOSRSAPrivateKey(n)]: When flashing XOS, we also burn a SEK block that includes 7 RSA Private Keys [XOSRSAPrivateKey(n)] 2048 bits.
  The public key parts are [XOSRSAPublicKey(n)].
  [XOSRSAPrivateKey(n)] are stored on the serial flash encrypted using [SigmaAESSymmetricKey]. Only XOS can directly access these keys.
- [XOSRSAPublicKey(n)]: These keys are not stored on the chip.

Key Hierarchy

- [XOSAESSymmetricKey(n,Certificate)]: 7 AES keys that depend on the CertificateID of the xtask that want to use them.

  SK = 128 lsb of SHA-1([XOSRSAPrivateKey(n-7)])

  MSG = 128 lsb of SHA-1(Certificate)

  [XOSAESSymmetricKey(n,Certificate)] = AES ecb of MSG using key SK

- These keys are not directly stored on the chip but can be recomputed by xos internally easily from the above formula.
  The [XOSAESSymmetricKey(n,Certificate)] is delivered to the customer when a certificate is generated.

XLOAD

Protected binaries signing:

1 - Customer generates a RSA 2048 key pair:
[CustomerRSAPrivateKey] and [CustomerRSAPublicKey]

2 - Customer sends [CustomerRSAPublicKey] to Sigma for Signing.

3 - Sigma attributes a new KeyID to the customer and signs
Certificate=(KeyID|KeyType|XOSKEYId|[CustomerRSAPublicKey])
using [SigmaRSAPrivateKey], resulting in SRDS(Certificate).

This certificate is sent back to the customer with
[XOSRSAPublicKey(XOSKEYId)] and
[XOSAESSymmetricKey(n,Certificate)]

XLOAD

4 - Customer writes an xtask and compiles it into task.bin

5 - Customer signs the task using [CustomerRSAPrivateKey]: CRDS(task.bin).

6 - Customer generates a random AES session key [SessionAESSymmetricKey] and IV [SessionCustomerIV]and encrypts task.bin : [SessionAESSymmetricKey](task.bin)) (padding is 0)

7 - Customer encrypts [SessionAESSymmetricKey] using [XOSRSAPublicKey(XOSKEYId)]: [XOSRSAPublicKey(XOSKEYId)]([SessionAESSymmetricKey]).

XLOAD

8 - Customer creates a protected binary :

| Certificate | SRDS(Certificate) | CRDS(task.bin) |[XOSRSAPublicKey(XOSKEYId)]([SessionAESSymmetricKey]) |

[SessionCustomerIV] | [CustomerAESSymmetricKey](task.bin) |

In short:

task.xload = | Certificate | CertificateSignature | TaskSignature | Encrypted Session Key | IV | Encrypted Task |

XLOAD

Clear xload format

As a special case, we use XOSKEYId=0xff to code the fact that the Task is not encrypted, but simply digitally signed:

task.xload = | Certificate | CertificateSignature | TaskSignature | Clear Task |

XLOAD

Certificate:

Included in the certificate:

CertificateID : 16 bits (but < 2048) starts at 0. Each certificate has a unique keyID. Sigma maintains a list of all the certificates that have been generated.

CertificateType: 8 bits. Indicates the certificate type

0: cpu Bootloader (zboot), cpu zone, used by ALL customers
1: cpu code, cpu zone, used to sign cpu kernels and applications
2: xtask1, xpu zone, used to develop and release SDK DRM implementations
3: video microcode, protected risc zone, used by Sigma Designs only
4: audio microcode, protected risc zone, used by Sigma Designs only
5: transport demux microcode, protected risc zone, used by Sigma only
6: irq handler running on xpu, xpu zone, used by Sigma Designs only
7: xtask2, xpu zone, used for Sigma Designs DRM implementations.
8: xtask3, xpu zone
9: xtask4, xpu zone
0xff: xos update

XLOAD

Certificate (continued)

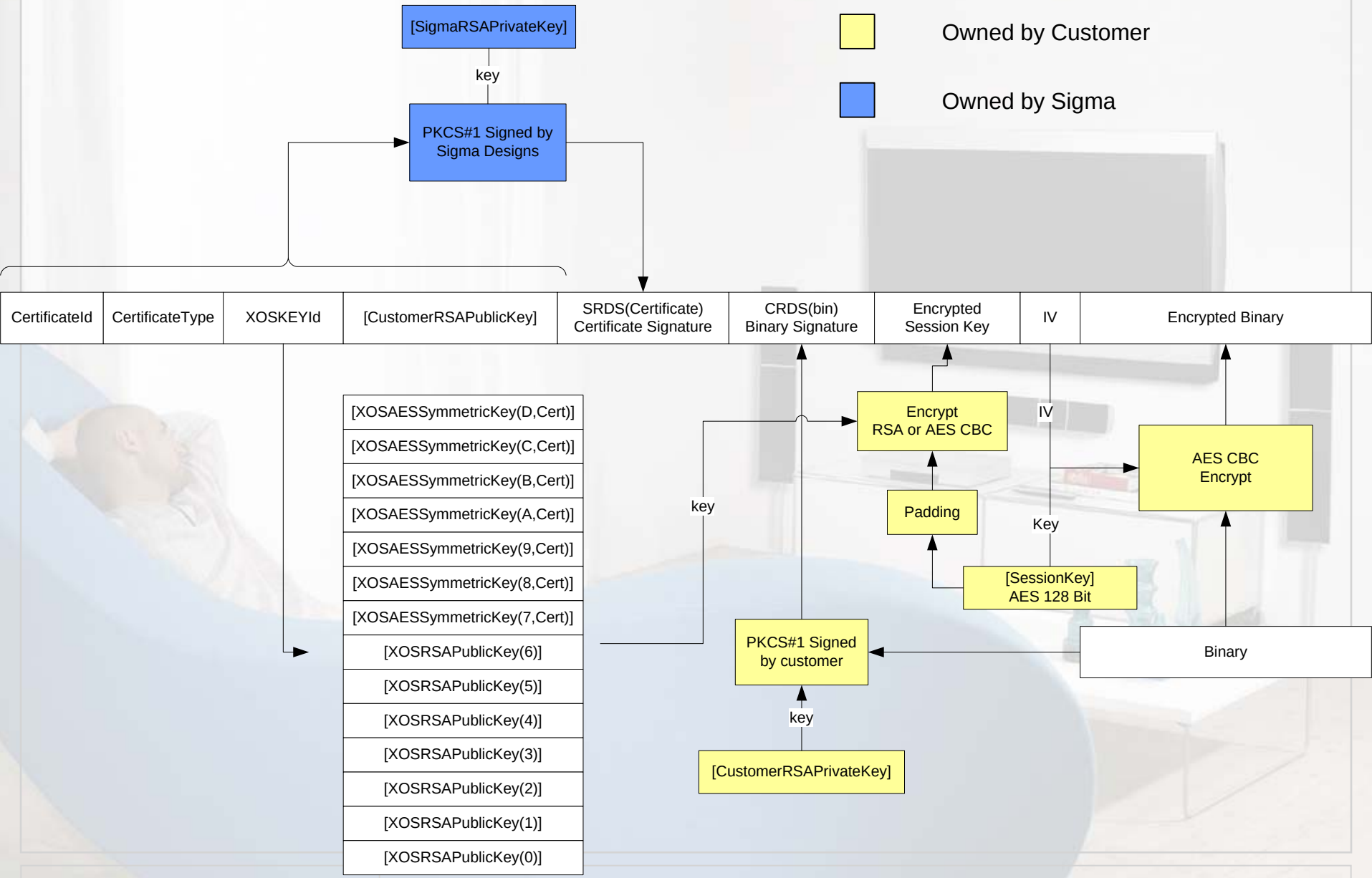
- XOSKEYId: 8 bits, indicates how the session key is encrypted.

  0..6: use a RSA encryption using [XOSRSAPublicKey(n)]

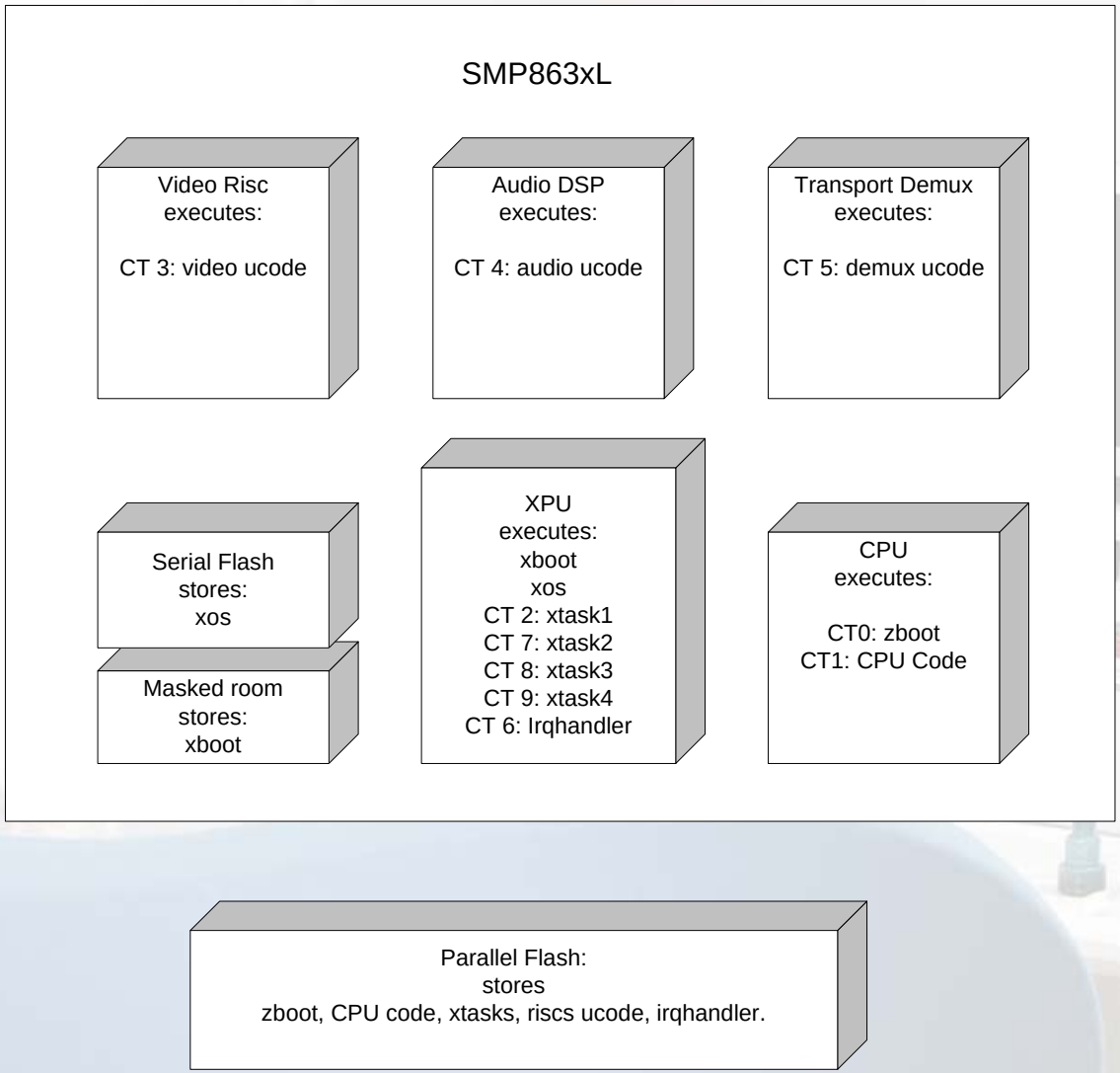
  7..13: use an AES encryption using [XOSAESSymmetricKey(n,CertificateID)] (*)

  0xff: no encryption

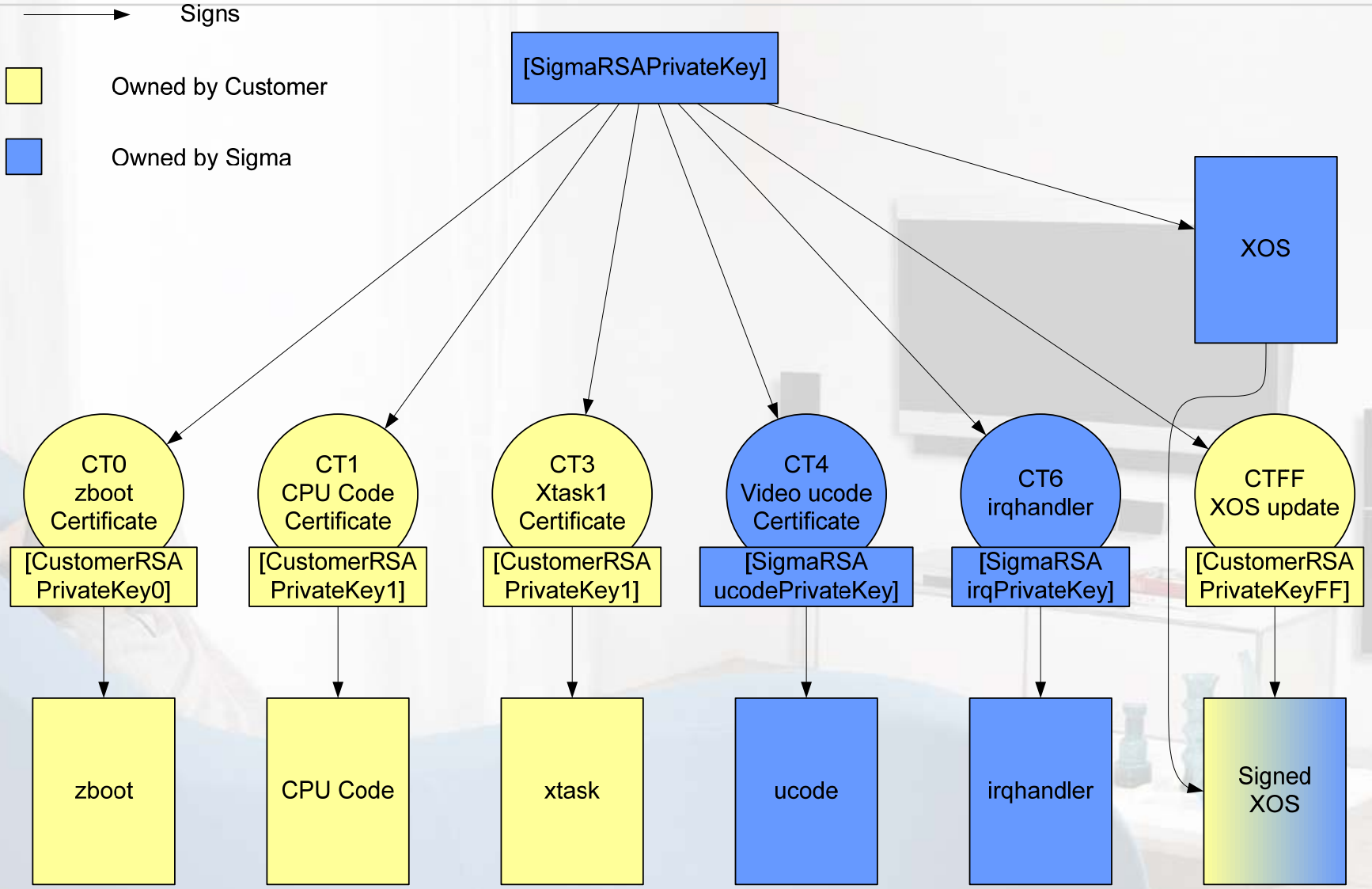
XLOAD


[SigmaRSAPrivateKey]
key
PKCS#1 Signed by Sigma Designs
Owned by Customer
Owned by Sigma
CertificateId
CertificateType
XOSKEYId
[CustomerRSAPublicKey]
SRDS(Certificate) Certificate Signature
CRDS(bin) Binary Signature
Encrypted Session Key
IV
Encrypted Binary
[XOSAESSymmetricKey(D,Cert)]
[XOSAESSymmetricKey(C,Cert)]
[XOSAESSymmetricKey(B,Cert)]
[XOSAESSymmetricKey(A,Cert)]
[XOSAESSymmetricKey(9,Cert)]
[XOSAESSymmetricKey(8,Cert)]
[XOSAESSymmetricKey(7,Cert)]
[XOSRSAPublicKey(6)]
[XOSRSAPublicKey(5)]
[XOSRSAPublicKey(4)]
[XOSRSAPublicKey(3)]
[XOSRSAPublicKey(2)]
[XOSRSAPublicKey(1)]
[XOSRSAPublicKey(0)]
key
Encrypt RSA or AES CBC
IV
AES CBC Encrypt
Padding
Key
[SessionKey] AES 128 Bit
PKCS#1 Signed by customer
Binary
key
[CustomerRSAPrivateKey]

Certificate Types

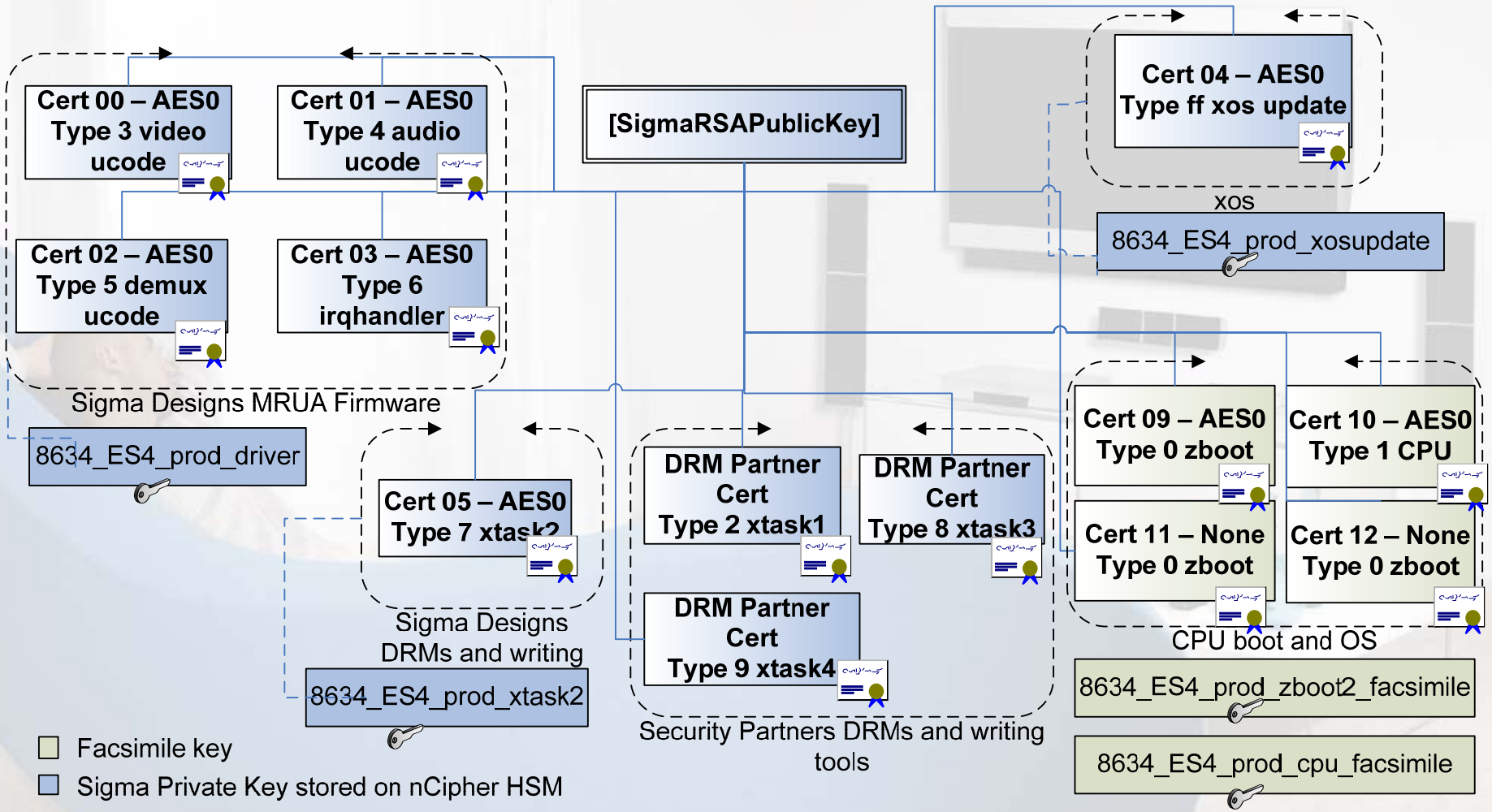


CT: CertificateType



CT: CertificateType

Certificate Hierarchy



Certificate Binding

xos can store for each certificateType a default certificate hash.

This forces certain binaries to be signed with a unique key.

When used in production, this prevents the software in the box to be replaced.

We plan to include the following functionalities:

- Bind a certificate type with a certificate.
- Unbind a certificate.
- Transfer the ownership of the binding to another certificate.

The unbinding or change of ownership can be global or chips specific (based on the serial number).

Binding for RevA

Cert Type	Cert Number	XOS KEYId	Notes
0: zboot			Customer should request one and bind SMP8634 to it
1: CPU			Customer should request one and bind SMP8634 to it
2: xtask1			DRM Partner or Sigma Designs
3: video ucode	00000	AES0	Production driver video ucode
4: audio ucode	00001	AES0	Production driver audio ucode
5: demux ucode	00002	AES0	Production driver demux ucode
6: irqhandler	00003	AES0	Production driver irqhandler
7: xtask2	00004	AES0	Sigma Designs Production DRM
8: xtask3			DRM Partner or Sigma Designs
9: xtask4			DRM Partner or Sigma Designs
ff: xos update	00004	AES0	Production xos update

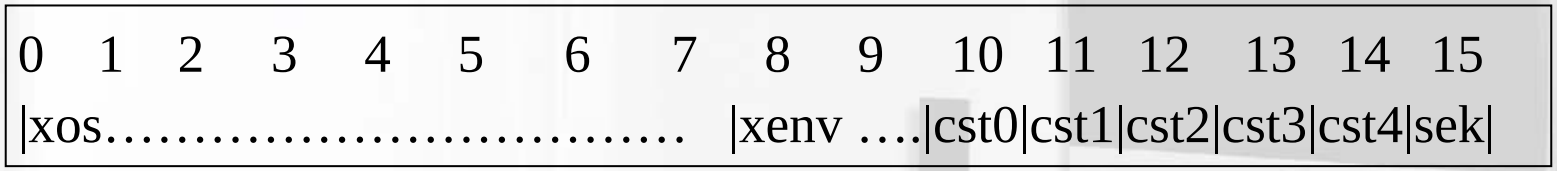
Key Hierarchy

XOS protected binary loading:

- CPU sends rpc with protected binary and address to load.
- XOS extracts the certificate and verify its signature.
- XOS consults the default certificates hash list at position certificateType to check if there is a hash. If there is a hash,
  - XOS hashes the certificate and verifies it is equal to the stored hash. If not, XOS refuses to continue.
- XOS verifies that the loading address is compatible with the certificate type.
- XOS decrypts the session key
- XOS decrypts the binary and loads it at the load address
- XOS verifies the signature of the binary.
- If the signature fails, wipes the decrypted binary.

Flash

Flash map in 4kB sectors (total 64kB)



- xos stands for the encrypted xos image
- xos env is the kernel own environment. It is 4KB, and duplicated for accidental power-loss-during-update recovery.
- cs0..cs4: five pages for different partners
- sek currently holds the seven [XOSRSAPrivateKey(n)]

Flash

Flash access:

- We reserve 12kB of flash for XOS + 20kB of reserved area for extension + 4kB of flash mapping information + 4kB of xos private keys(sek). This leaves 28kB free. We partition per page: we have 7 4kB pages on the flash that can be used by xtasks.

XOS has the following services:

1 - xos_uapi_xenv_format: Format a page and change the ownership to the originating task.

2 - xos_uapi_xenv_chown: Change page ownership.

3 - xos_uapi_xenv_set/get: Save and Get records in the page. Records can be ReadWrite, ReadOnly and OTP