



# **Securing Highly Sensitive Assets with AuthenTec's SafeZone Secure Platform**

## Table of Contents

<b>Chapter 1: Introduction</b> .....	<b>3</b>
<b>Chapter 2: Threat models, Protection Levels, and Key Material</b> .....	<b>4</b>
Types of Attack.....	4
Types of key material.....	5
Key Ownership .....	7
Protection Levels .....	8
<b>Chapter 3: Asset Store – more than just Key Storage</b> .....	<b>9</b>
<b>Chapter 4: SafeZone Secure Platform</b> .....	<b>12</b>
Secure Platform EIP-123 Crypto Module.....	12
Secure Platform SafeZone Software Libraries .....	13
<b>Chapter 5: Use Cases</b> .....	<b>15</b>
Secure Storage and Device Binding.....	15
Secure Boot Confidentiality protection.....	15
Prevention of IC overproduction and feature configurability.....	15
Content Protection and Conditional Access .....	16



# Chapter 1: Introduction

Lots of devices these days rely on the use of secret data to execute their intended function. For instance, devices require secret data to:

- Get access to the service network that they are part of, for example:
  - to access a mobile network,
  - be able to decrypt Conditional Access media,
  - to access a bank's remote banking server.
- Offer certain security services to their users, such as:
  - Secure storage and access control to user passwords, licenses, or loyalty data,
  - Encryption of removable storage data such as hard disks or USB memory sticks,
  - Allow remote access to, and secure communication with, a company network.
- Ensure their own operational integrity, through:
  - Booting from authenticated and encrypted boot software,
  - Running only authenticated programs,
  - Securely storing device-specific data such as :
    - Configuration- and feature enablement settings,
    - Operational parameters (eg. Radio transmitter settings).

Apart from the fact that the items above require the protection of certain secret data from 'some' attacker, it is good to recognize that the different items represent:

- interests from different parties (mobile operator, device user, device manufacturer),
- different monetary or perceived value (free mobile access, device software theft or hacking, identity theft).

In other words, a single device may have to protect data belonging to different parties, as well as data that has different value to different people (both data owners as well as attackers), which means that the amount of effort spent to protect a piece of data, as well as the amount of effort that can be expected to be spent on attacking the data, can differ greatly. If you add to that the observation that in a lot of cases, the party whose data needs to be protected, is not the party paying for its protection, you start to get a grasp of the interesting security landscape that a lot of devices have to deal with.

## Chapter 2: Threat models, Protection Levels, and Key Material

A threat model is the term the information security community uses to determine the type of attacks that a device must be capable of resisting. This threat model is a function of:

- The resources that an attacker can spend to attack the device,
- The value (to the attacker as well as the data owner) of a successful attack, since this determines the amount of effort an attacker is willing to spend on an attack, and the amount of effort the data owner is willing to spend to defend it,
- The level of access the attacker has (or can obtain) to the device and its interfaces.

Attacks against a device can come from many different directions. In most cases, the feasibility of an attack is determined by the amount, and type, of access an attacker has to the device. This level of access may actually change over the lifetime of the device, depending on where it is located, what operational mode it is in, and what sort of secrets it holds at a certain point in its lifecycle.

A simple example is for instance the key material used to protect SIM-lock data in a mobile phone. While the device is still locked, this data is sensitive and needs to be protected, however once the phone's contract has expired and the phone has become unlocked, the SIM lock parameters no longer need to be protected.

### ***Types of Attack***

We recognize the following classes of attack vectors:

#### **Physical Attacks**

Physical attacks attempt to break device security by modifying, or tampering with its hardware. Physical attacks can themselves be subdivided into different levels according to the amount of damage the attack does to the device, and whether or not the device must (still) be operational during or after the attack. Examples of typical physical attack classes are:

- Online versus offline:
  - Passive signal probing versus active signal alteration,
  - Chip-level versus PCB level.
- Intrusive versus non-intrusive:
  - Destructive versus non-destructive.

For instance, a well known attack is attempting to read the contents of a flash chip located in the device. Since in most cases the attacker must de-solder the flash component from the PCB to do this, thus damaging the device, this is a PCB level, intrusive attack (whether it's destructive depends on the soldering skills of the attacker but in most scenarios, this is not considered a destructive attack). Since the device does not need to be operational for this attack to work, it's an offline attack. Adding a mod chip to a device, however, is an online, active, intrusive attack.

Attacking a device through one of its debugging interfaces, which requires no modification to the device PCB but does require access to the device's physical debug interface, is an example of an online non-intrusive attack.

Actually decapping a chip opens up another class of intrusive and typically destructive attacks, that can in themselves again be online or offline, and when online, be active or passive.

#### **Logical attacks**

Logical attacks are typically used to describe attacks that do not require the attacker to have physical access to the device. In almost all cases, these types of attack aim to exploit bugs or vulnerabilities in software. A typical

example of this type of attack is of course the infamous ‘ping of death’, exploiting a buffer overflow bug in the IP stack of the receiving device. Another type of attack is the scenario where a rogue, typically downloadable, application breaks the application separation enforced by the OS kernel (or even the Hypervisor).

Other examples of logical attacks are:

- Breaking, stealing or guessing passwords,
- Exploiting bugs or flaws in the OS, to allow:
  - Breaking of application separation (accessing other application’s memory),
  - Changing execution privilege level (trying to become superuser or root).
- Exploiting bugs or flaws in an application:
  - To get at the application’s data directly or to attack other applications or the OS through this application.
- A common attack is through (flawed) programs that do not check input from users or network, which can for instance lead to buffer overflow errors (the ping of death falls in this class, obviously).

Most successful attacks against a device’s secrets require a combination of smaller successful attacks of different type.

It should be clear that the different types of attacks require different attacker skills and resources. This whitepaper does not attempt to create a full classification of all attacks possible; it’s merely attempting to give an overview of the threat landscape that a typical device is in, and the diversity of things that need to be considered when designing the security mechanisms for a device. Different attacks have different cost to the attacker, and likewise, countermeasures have different cost to whoever is paying for their implementation. Good security design attempts to strike a balance between cost of countermeasures and cost associated with a successful break, versus cost of attack and potential gain to the attacker.

### ***Types of key material***

Next to the possible attack scenarios, it is important to look at the device secrets (typically called assets) themselves. Where are they located, how and when are they used, and what is the cost if an attacker successfully gets access one or more of them?

During the lifetime of the device, a device typically needs to store its assets in a location where they can survive power cycles. Thus, assets may need to be stored to a hard disk or solid state disk, to an eeprom or flash component, or to battery backed memory. Very often this type of non-volatile storage requires that the assets, before they can actually be used, are transported to a different location or memory type. Needless to say that all storage locations as well as the transmission paths for the assets, can be a target for an attack, especially if the assets are transported between different physical components (allowing a PCB-level attack).

It’s common to encrypt any device assets when they are stored in non-volatile storage (we will discuss some of the challenges of secure storage later on in this whitepaper). However when the device needs to use these assets directly, they need to be available in plaintext form; in other words, the typical ‘offline’ cryptographic protection can not be used and the device must rely on other protection mechanisms. One typical approach is to rely on the physical protection offered by a chip package. By keeping plaintext assets within the physical boundary of a chip package, the device attempts to ‘step up’ the cost of an attack by requiring the attacker to mount a physical attack against a chip package, such as decapping and probing, or side channel attacks. If the assets to be protected are of very high value then the security of the chip itself may be enhanced by adding dedicated attack prevention and detection mechanisms.

Making sure the asset stays within the physical boundary of the device, itself requires special measures; the processor handling the asset must have support of a special on-chip memory area to keep, and use, these assets. Clearly the asset should not be ‘swapped out’ to external SDRAM or put in regular stack- or heap areas that, for most systems, are located in external memory. And of course the fact that an asset is kept inside the physical boundary of the chip, primarily affects the difficulty of mounting a physical attack; logical attacks are still possible, especially in case an attacker is capable of running or influencing software running inside the device.

Another consideration for the security architecture of a device is the intended use of the asset or key material. This is often a determining factor for the value of the asset, especially for the attacker. Consider, for instance, the following scenarios for the (mis)use of key material

### **Break once, use everywhere**

Although typically a scenario that is avoided by secure systems designers, there are situations where it can be economical to use a single secret key to protect items or communication for a large number of devices. This is typically the case when resource constraints, such as device size or cost, computational power or communication setup time, prevent the use of more sophisticated keying mechanisms. Another reason to choose for this scenario is the fact that most 'more sophisticated' keying mechanisms also require a more sophisticated (costly, cumbersome) key management infrastructure.

A consequence of using a single key to protect the interests of multiple devices is of course, that if an attacker gets access to that key, it does not just affect one single device but all devices using that key. Attacking this key is also made easier by the fact that an attacker can often 'sacrifice' a number of devices to refine its attack to get access to the key. For this reason, even though the mechanism is typically selected to reduce device (or device support infrastructure) cost, this mechanism typically does put the highest security requirements on the device.

### **Identity Theft**

When a device needs to be able to prove its identity (or that of its user) to a communication partner, it uses a unique, secret key to do so. Typically the private half of an asymmetric private/public key pair is used for this. The security (or rather, trust, in this case) is based on the fact that the private key is secret and locked in the device so the receiver can trust that he is really 'talking to' the intended device. Obviously when an attacker gains access to this key he can impersonate that device, in other words steal its identity as far as the communication partner is concerned. The level of protection to provide for this type of key material depends on the answer to questions like:

- What the attacker can achieve with the stolen identity
- Does the attacker need to steal the identity of a specific device to achieve his goal, or is it sufficient for him to break any device of a given type to be successful?
- Does the attacker require access to the key material itself (for use separate from the device or to distribute to others) or is it sufficient to achieve the ability to use the key while still inside the device (attacker requires physical possession of the device and the ability to use the key material to sign his own message).

### **Breaking the Root of Trust**

This is the reverse scenario from the 'identity theft' example above. In situations where a device needs to be able to determine if a piece of data or a communication message comes from a trusted source, the device needs to know the identity of whoever created the original message so it can check the signature over the data. Again, typically one half of a public/private key pair is stored in the device for this reason, but this time it's the public part. The device 'trusts' in this public key (and thus, the signatures generated by its secret counterpart) is based on the public key being 'immutable', which means that it needs to be protected against modification by an attacker. In this case, the key does not need to be kept secret.

If the attacker can modify the value of this key then he is able to create a message that the device will 'trust', for instance allowing the attacker to run software on the device or use certain functions (or other key material) that would normally not be available to the attacker.

### **Defeating Conditional Access**

Most Conditional Access (CA) schemes use different levels of key material. There is of course the 'root' key material that is required to get access to any program belonging to the CA scheme. This key material obviously needs to be very well protected, since having access to these keys allow an attacker to get free access to the CA service – especially if the keys also fall in the 'break once, use everywhere' category. CA schemes also derive short-

lived keys, used to decrypt a limited portion of a program stream. The protection required for these keys is typically much lower since having such a key only allows access to part of a program.

### Changing SIM-lock to SIM-unlock

With SIM-lock, key material is used to verify the authenticity of certain SIM parameters, programmed into a phone by the operator, so the phone can verify that it is still using that operator's SIM card. This means that the SIM lock data is signed by some key material; in a lot of cases, using a symmetric algorithm (eg. HMAC), using a secret, symmetric, key. The challenge with this particular scenario is that the device must obviously protect the confidentiality of the signing key, but it must also protect access to the key and the resulting signature. If an attacker can get access to the key directly, he can obviously sign his own SIM card parameters and thus allow the device to accept another operator's SIM card. Even simpler, if the attacker can submit (false) SIM card parameters and have the device report the authentication value it calculated (instead of just reporting 'authentication failed'), the attacker can use the device itself to sign its own SIM parameters.

Although this sort of behavior by the device is clearly a security violation (every security architect will tell you that), actually implementing the correct behavior properly can be tricky, especially if the underlying cryptographic algorithms are provided at the primitive (eg. HMAC) level, with software actually handling the signature data. This is because at the cryptographic primitive level, symmetric-key based signature calculation and signature verification are the exact same operations, with the only difference between the two determined by how the resulting signature data is handled.

Many more examples and use cases for application and handling of key material can be provided, however the point of these examples is to provide some insight into the variety of key usage, handling, storage and access mechanisms that exist in actual device deployments. This diversity calls for a variety of protection levels and mechanisms in the device.

### Key Ownership

Just as a device is handling data from different sources and from different owners, so does the device have to protect and control access to key material that belong to different parties. On a mobile phone, for instance:

- The **Phone Manufacturer** wants to make sure his phone's firmware is protected from theft or modification. The Phone Manufacturer will typically also want to make sure that the device has the necessary security features in place to protect the Network Operator's interests.
- The **Network Operator**, especially in scenarios where the device is being sponsored, wants to be certain that the device can remain locked for exclusive use on his network for a certain period of time. He also wants to be sure that the radio parameters used by the device can not be modified by the end user. He obviously also wants to make sure that only legitimate (billable) devices can access his network.
- Modern smartphones allow users to access content with increasingly higher quality. For that reason, **Content Providers** insist that a device protects their data from copying and redistribution.
- **Banks** and **Credit Card Companies**, and obviously also the **Device User**, have an interest in making sure that the device can properly protect their mobile wallet, mobile banking login credentials, and credit card numbers, from use by an attacker, for instance after the device has been stolen.
- The **Device User** has a concern that his privacy is properly protected by the device, for instance from apps and phone malware or worse, ransomware.

Note that in some cases, the security of an item depends on the security of another – if for instance the Phone Manufacturer's firmware is compromised, it is likely that the phone as a result of that can no longer adequately protect other items such as SIM-lock or user privacy.

Even though not all data in the above list will always be protected directly by cryptography and thus, key material, it should be clear that there will be situations where the device needs to protect the key material it holds, from an external attacker, but also from persons that have regular interaction with the device (as user, not attacker) and may also have key material stored in the device itself. Thus, the device needs to be capable of recognizing different

key owners, have mechanisms to allow it to tell them apart, and offer the appropriate protection level for each key type.

What is also interesting to note here is that in a number of cases, the party paying for the creation of the security mechanisms (in this particular example, the phone manufacturer, or the mobile application processor manufacturer) is not always the party that gets hurt (most) if one of its security mechanisms fails, especially if the security mechanism in question is protecting end user data as opposed to phone manufacturer data.

## **Protection Levels**

Recognizing the different threat models, key owners, and key usage scenarios, it will be clear that the device will have to be able to deal with different levels of protection to balance usability, protection cost and security of the sensitive data it holds.

In the beginning of this chapter we have identified a number of attack scenarios; to defend against the different attack types the device can deploy different protection levels and mechanisms.

Just as the attacker has different methods at his disposal to attempt to attack the device, so does the device, to protect itself. For instance it can limit access to sensitive data through:

- Restricting the physical location of plaintext key material, eg. by keeping the data:
  - Inside a secure execution environment,
  - Inside the chip package,
  - Only temporarily visible in off-chip memory.
- Restricting access to key material:
  - Based on cryptographic mechanisms and secret- or preshared key material, such as login credentials, public/private key pairs or internally generated secret key material,
  - Based on application separation provided by an OS kernel or Hypervisor,
  - Based on (system) mode of operation (boot, debug, mission mode),
  - Based on processor mode of operation (Secure vs. Normal, Privileged vs. Restricted),
  - Based on physical device (Baseband Processor vs. Application Processor).

The device can also deploy attack detection mechanisms (anti-tampering technology) and act upon the detection of a tampering attempt by destroying sensitive data.

Obviously the different protection mechanisms provide different levels of protection against different types of attacks; for instance keeping data inside an (on-chip) secure execution environment protects the data against PCB-level attacks and logical attacks (from other SW running inside the chip) but not necessarily against intrusive- or destructive attacks.

A protection mechanism almost always comes with an associated cost, not just in terms of device complexity but also in terms of the usability of the material. In the secure execution environment example, the additional cost comprises the need for a separate (isolated) processor with its infrastructure (device complexity), but it also requires any application that needs to access or use the data inside the secure execution environment, to be ported (at least partially) to run inside that environment.

The device designer could have chosen for a setup where the sensitive data is stored in encrypted form, possibly off chip while not in use. When the sensitive data must be used, it is made available to the device in plaintext form. Clearly this mechanism provides less protection (primarily against offline PCB level attacks) but it does not require any modification to the applications using the sensitive data. If the actual usage of the sensitive data is limited, the temporary exposure of the data while in plaintext form may be acceptable compared to the additional cost incurred if the data would have been kept inside a secure execution environment at all times. Among other things, this approach does not require the application using the data to be split into a part running in the normal environment and a part running in the protected environment.

## Chapter 3: Asset Store – more than just Key Storage

The previous chapters illustrate the complexity of the security playing field, for devices containing and protecting data they cannot allow falling in the hands of an attacker. The device will have to recognize different levels of security and attempt to successfully combine these with other design constraints like useability, complexity as well as manufacturing, provisioning and maintenance cost. This requires a robust and flexible security architecture that must be adaptable to the device needs during every stage of its lifetime.

To help the device Security Architect, AuthenTec has developed a comprehensive Secure Platform to act as the foundation for a security architecture that can be tailored to meet the protection requirements the Security Architect needs. The Secure Platform provides the cryptographic building blocks, as well as the so-called ‘trust anchors’ in hardware, to allow the security architect to implement a comprehensive security architecture without having to bother with the complexities of low-level cryptographic operation and key management.

The AuthenTec Secure Platform Comprises three main components:

- The EIP-123 Crypto Module hardware, to provide the cryptographic acceleration and the ability to put a root of trust in hardware, and to provide the trusted execution environment for the Asset Store (below),
- The SafeZone Software, to allow efficient use of the hardware and to provide higher level services using, if needed, the trust anchors provided by the hardware and the Asset Store,
- The Asset Store, implemented as firmware inside the Crypto Module but also available as software component, to provide key management, storage and access control to key material, protected by hardware.

Protecting key material requires more than just the capability to store the key material behind some fence. Having key material makes no sense if the keys can not occasionally be used securely as well. However, if the protection of the key material also needs to be extended to cover the secure usage of the keys, then care must be taken that:

- The keys are only used according to the (user) access policy set by the key owner,
- Keys can only be used by the algorithm they are intended for,
- Any intermediate material produced or required by the algorithms that may contain information on the key material, must be protected as well (eg. keyed hash state, stream cipher algorithm state, derived key material).

In addition to being able to securely use key material, it is very often also required to be able to securely create and derive key material. Creating brand new, unpredictable, key material requires a source of True Random Numbers, which can only be achieved through the collection of randomness from truly random sources like thermodynamic noise. After collecting an amount of random data from such a noise source, the data needs to be correctly postprocessed to turn it into proper key material.

Finally, most systems require the ability to derive key material from a specific root key. This can have multiple purposes; for instance, deriving key material from the Hardware Unique Key allows generation of different keys that can be securely used in different algorithms and mechanisms, without leaking significant information of the HUK itself; while at the same time, the fact that the key material is derived from a key only known to the device, also makes the derived key material only known to the device and thus, any data encrypted or integrity checked using these keys, can only be processed again by this particular device. The protection of data in this way, using key material derived from a HUK, is also called ‘device binding’ but key derivation is used in numerous protocols to allow the creation of multiple keys from a single source of secret (key or just random) data.

To allow the key material created or managed by the Asset Store to survive power cycles, the Asset Store supports the export of keys with their ownership- and policy information, protected by a special Key Encryption Key. To ensure that no other device can use the key material exported in this way (even if the other device incorporates the exact same Asset Store), the Key Encryption Key itself is not stored externally but rather, derived from the HUK each time the Secure Platform is initialized. For this and other reasons, the Asset Store deploys an internal key

hierarchy that is managed and used completely by the Asset Store itself. Since all internally used key material is derived and not created from random data, there is no need to store keys in battery backed memory.

The concepts mentioned here represent the basic tools to set up a complete key storage and protection system. The SafeZone Secure Platform Asset Store, combined with the SafeZone Software components, hides the security complexities of this system from the device software developer and the device user, allowing access to the services through standard interfaces like PKCS#11. The Asset Store provides the flexibility to support key handling at a variety of security levels, depending on the security- and usage requirements of the system.

The Asset Store supports a large variety of key types. For instance:

- Keys can be generated using the TRNG. This creates key material that is only known to the device, but needs to be stored in non-volatile memory to survive power cycles.
- Keys can be derived from a root key, stored in on-chip non-volatile memory. This allows key material that is only known by the device (assuming the root key is secret) but that does not need to be stored to survive power cycles – whenever the key is needed it is simply recreated.
- Keys can be imported, either plaintext or as wrapped key material. This obviously allows sharing of key material. Once a key has been imported inside the Asset Store, the original key material can be removed from the device so the shared key is only available inside the Crypto Module. This can be used to support eg. personalization of a device (where the key material is imported in a controlled way or in a secure environment) and to limit the exposure of key material in an uncontrolled environment, from which it can be removed after importing into the asset store.
- Keys can be provided directly for use in the cryptographic algorithms present in the Crypto Module. This allows less sensitive key material (that does not require the protection of the Asset Store, or for reasons of usability, need to remain accessible by the host processor software) to be used for hardware acceleration by the hardware implementation of the associated cryptographic algorithm.

The following figure shows some of the relationships that can exist between the different types of keys, and how they can be used. It also gives an indication of the security level that they can represent.

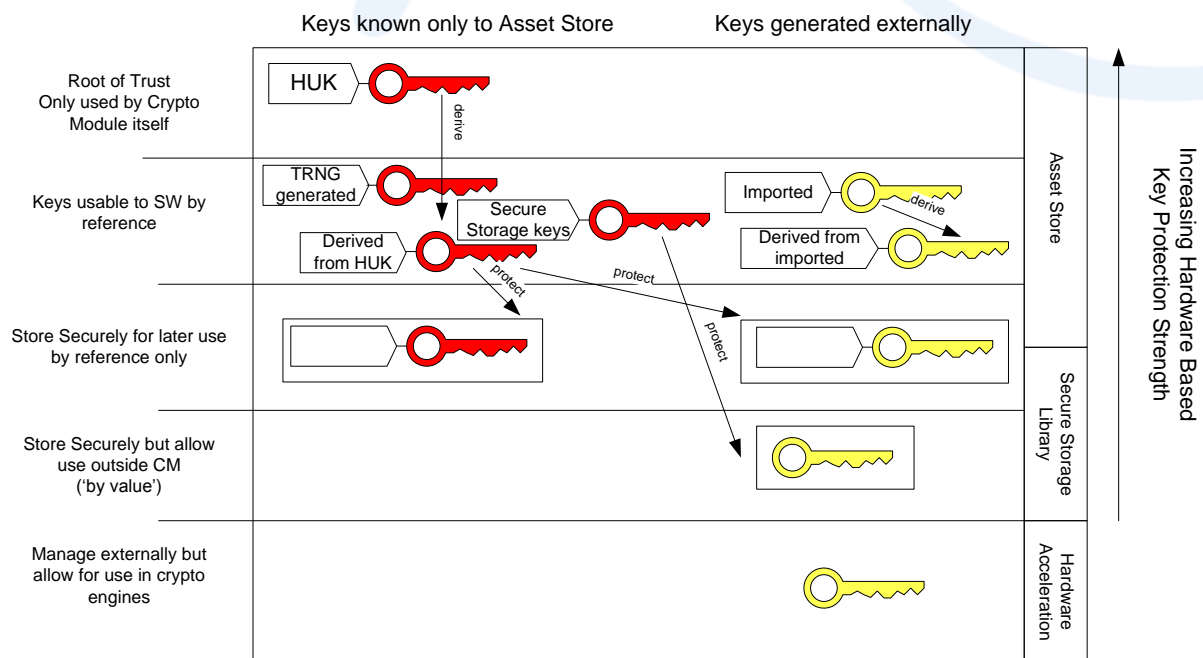


Figure 1 Key Types, Usage and Relative Protection levels supported by Asset Store

Keys with a label attached to them are keys that are managed by the Asset Store and thus have ownership- and usage policy attached to them.

The actual security provided for, and required by, a key depends on many device specific factors. However, the flexibility in key types offered by the Asset Store allows the Security Architect to choose the most appropriate setup. The chapter on use cases gives some examples of how the flexibility of the Asset Store can be applied to meet the security requirements of some real world applications.

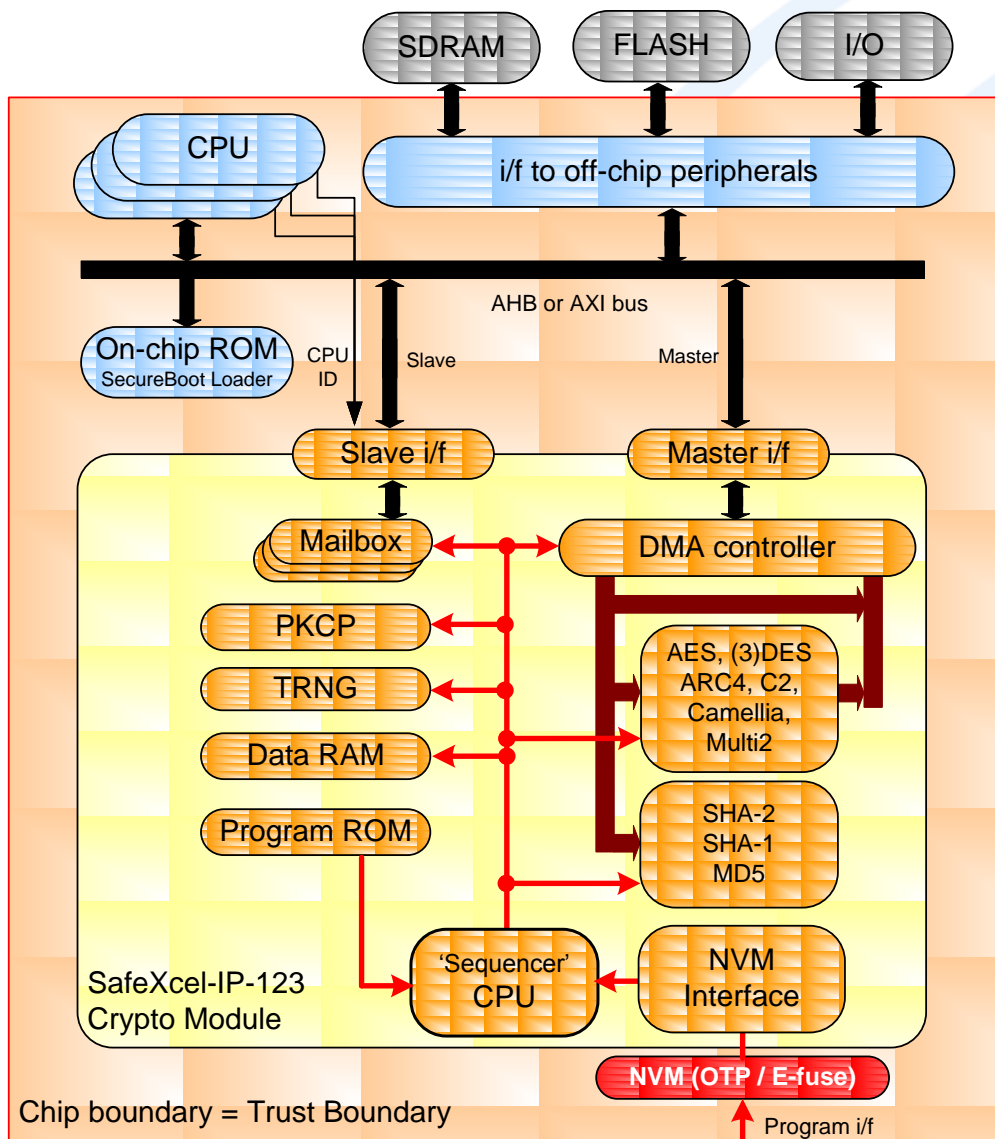


# Chapter 4: SafeZone Secure Platform

AuthenTec's SafeZone Secure Platform is designed to support data protection at the various levels mentioned in the previous chapter. It comprises hardware and software components that can be combined in different configurations to allow the right balance of data protection, usability, and cost.

## Secure Platform EIP-123 Crypto Module

The hardware component is shown in Figure 2. When present, it can provide the capability to keep key material on-chip and out of reach of any of the (application) processors on the chip. It also provides the capability to internally generate secret key material, eg. by using the inbuilt True Random Number Generator. Furthermore it can internally derive and manage key material from the hardware root of trust provided by the Hardware Unique Key (HUK), as part of the extensive key protection technology called the 'Asset Store', explained in more detail in the previous Chapter.



AuthenTec HW IP component

Customer Foundry IP component

Customer IP component

Customer external component

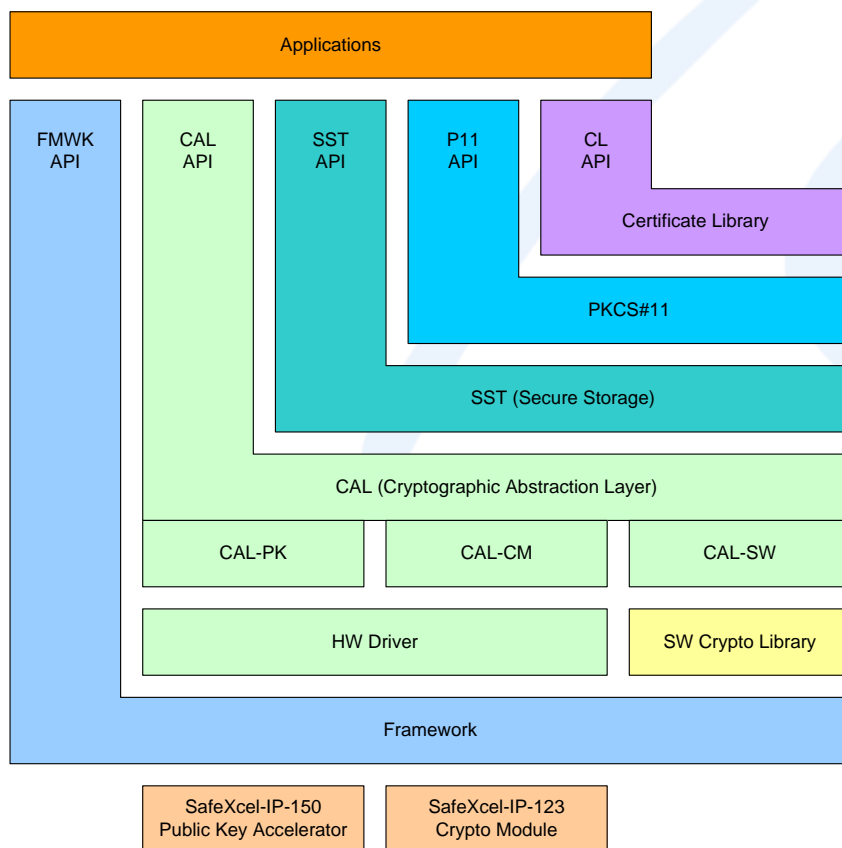
Figure 2 SafeZone Secure Platform Hardware Component: the Crypto Module

Some of the key features of the Crypto Module Hardware are:

- ROM code controlled operation- not possible to influence from external processors,
- All key material used by the algorithms present inside the module – no external key exposure,
- Ability to cooperate with multiple hosts using separate mailbox (control) interfaces, and support priority/Quality of Service support,
- Built-in public key and TRNG support,
- Low power – support for active clock control,
- Low gatecount,
- Flexible support for built-in hardware crypto- and hash algorithms.

### **Secure Platform SafeZone Software Libraries**

In addition to the hardware component, AuthenTec SafeZone Secure Platform also comprises multiple software components, either used in combination with the hardware component, or used as a software-only solution.



**Figure 3 Secure Platform SafeZone Software Libraries**

The software components can make use of both the security- and key storage services offered by the hardware component, as well as the cryptographic algorithm acceleration services. For instance, the hardware- and software components work together to implement the two ‘design choices’ mentioned at the end chapter 2, as follows:

Certain high-valued keys can remain locked inside the Secure Platform hardware component, only to be used ‘by reference’ by the Software components. For less sensitive (let’s call it ‘user key’) material it is also possible to just store the key material securely, encrypted using a storage key. This storage key itself is considered a highly valued key and is thus only used by reference by the software because it is itself is locked inside the hardware component.

The software can then request the hardware component to unwrap the (user) key, so the resulting key material can simply be used by the software directly.

Both hardware and software components also work together to provide access control to key material. In particular the PKCS#11 and Secure Storage components are capable of extending mechanisms used by typical operating systems to identify applications and execution contexts, to distinguish who/what application is requesting access to certain key material. The owner of a certain key (set at the time of the initial creation of the key) can specify an access- and usage policy for his key, which is stored with the key and enforced, by both the hardware and software components, over the lifetime of the key. It will be obvious that for the software components this functionality depends on the proper operation of the Operating System. The Hardware component, however, is also able to extend this mechanism to support key access separation between physical processors, based on the 'CPU ID' signals depicted in Figure 2.

The Key storage and access services offered by the hardware, however, have more sophisticated key management- and protection capability, implemented in the Firmware of the embedded 'Sequencer' CPU. This functionality goes by the name of 'Asset Store' and is explained in more detail in the previous chapter.

## Chapter 5: Use Cases

### ***Secure Storage and Device Binding***

This is the most straightforward use case for the Asset Store. In situations where the device needs to store data in a location that is easily accessible by an attacker, such as off-chip Flash, the device needs to be able to provide confidentiality- and integrity protection to that data. This is typically referred to as 'Secure Storage'. In addition the device typically also needs to make sure that no other device, even one of the same type as itself, can read that data, a feature referred to as 'Device Binding'.

Device binding is achieved by making sure each device uses its own unique keys for the confidentiality- and integrity protection. At least one unique key (the 'Hardware Unique Key') must be present in the device. This HUK can then be used by the Asset Store as the root key for the derivation of key material for encryption- and integrity protection. The protection offered by the Asset Store over (for instance) a software-only solution is that the storage key material is guaranteed to remain on chip. This means that passive attacks such as SDRAM data bus probing will not allow an attacker to succeed. He will have to take control of the software running in the chip to have a chance of using the key material. This software can of course itself be protected using secure boot. Note that because key material protected by the Asset Store is never allowed outside of the hardware boundary of the Crypto Module, the attacker can (request the) use of the keys but he can not distribute them. The device must be operational to use the keys.

### ***Secure Boot Confidentiality protection***

Secure Boot solutions typically rely on the so-called 'immutability' of on-chip ROM code in combination with an on-chip public key. In short, the on-chip code uses the on-chip public key to verify the signature over the software image (stored, again, in some off-chip location like Flash). Of course the Crypto Module hardware allows the acceleration of the cryptographic operations but for this particular stage of secure boot, it does not provide additional security. This changes if the software image also needs to be confidentiality-protected. In this case the software image is encrypted using some secret key, which implies that the device stores this secret key securely as well. This is obviously where Secure Storage comes in again. And by using the ability of the Asset Store to provide access control to key material, the system can be set up to only allow access to the boot image confidentiality key while the device is executing from ROM code, to prevent the use of the secure boot confidentiality key by an attacker running his own software on the device.

### ***Prevention of IC overproduction and feature configurability***

Fabless chip developers are sometimes confronted with the situation that their foundry overproduces their device to sell the pirated copies under its own brand in countries that are 'less sensitive' to Intellectual Property protection. One way of preventing this practice is by requiring each device to be activated before it becomes (fully) functional. The activation process requires the device to register itself with the original chip developer (IP owner), so the IP owner can sign the activation code and return that code to the device. This allows the IP owner to register exactly how many devices are 'activated' while leaving unactivated devices inoperable. To allow this mechanism to work securely, a number of features are required:

- The device must be able to recognize the IP owner's signature,
- The device must be able to store the fact that it has been activated, in such a way that it is unique to this device and cannot be used to activate other devices,
- The device must have a unique identity that cannot be predicted or read out by an attacker,
- The device must be able to activate/deactivate its functionality based on the presence/absence of the activation data.

The first bullet is typically achieved through the use of an immutable public key, such as the key in ROM used by Secure Boot. We've also already covered how to achieve secure boot and device binding (second bullet).

Assuming the presence of a HUK, the requirement for the device to have a unique identity is also addressed. Care should be taken, however, that the secrecy of the HUK is properly guarded- since the HUK is being used for

multiple purposes already. For this reason the Asset Store provides strong key derivation mechanisms, allowing the use of key material derived from the HUK rather than the HUK itself.

Activation and deactivation of device features can be achieved in different ways; for instance the device may refuse to boot completely (combining the activation code with secure boot), or by only allow access to part of its hardware (using the activation code to control enable/disable bits in hardware, controlled by the secure hardware domain provided by the Crypto Module.

Obviously the activation concept can be extended to allow partial activation or feature configurability, allowing a single mask set to yield different device types with selected feature (sub)sets

### ***Content Protection and Conditional Access***

Most content protection schemes already recognize the fact that the way key material is used, influences the amount of protection that can be provided for a given key. For this reason, such schemes typically distinguish between key material that is used to identify a user or allow (conditional) access to a service, and keys used to actually decrypt (sections of) actual content (such as a TV show, Ringtones data, or a 'Video on Demand').

In addition, certain content protection schemes require the device to store a (secret) device key that it can use to prove that it is indeed compliant to that protection scheme. This key typically also represents the identity of the device in relation to that protection scheme.

The key material that is considered more valuable, such as the material representing the device or user identity, content protection scheme membership, or service subscription, also requires higher levels of protection. Typically they can only be handled in what is called a trusted execution environment, and they must be protected against board level attacks such as probing of flash device replacement. These key protection requirements can only be addressed by handling and using the key material on chip, in a module separate from the main (application) processor, which often runs untrusted software and is thus vulnerable software based attacks. This is obviously one of the scenarios the Crypto Module and Asset Store are designed to support. The key hierarchy and flexibility offered by the Asset Store also allows the 'content descrambling' keys to be exported from the Asset Store for use by the application that handles the content, to allow convenient integration with existing applications. The key management firmware inside the Crypto Module is, or can be made, aware of the key management scheme used by the content protection scheme and can therefore decide which keys can and which keys can not be allowed to be given to the external application.

In some cases, even various keys that are created by the device while in operation (such as derived keys, media keys) as well as license and usage control data must be protected and stored by the device, which is in itself a Secure Storage use case.

Combined with these key storage- and handling requirements, content protection schemes require a device to provide what is called 'operational integrity', which means that the device must be capable of checking that it is indeed running the software that it is intended to run, unmodified – in other words: a requirement for Secure Boot.

With this example in particular we see a lot of security and key management requirements combined, with a number of mechanisms working together to provide a complete protection scheme. The Asset Store allows different mechanisms to base their root of trust in a single value, the HUK, without compromising that value. By taking the key handling operations away from the system processor the key material is guaranteed to remain on-chip and out of reach of the main processor, unless the key was explicitly allowed for use by the processor. By being aware of specific key management and key derivation schemes, the Asset Store can enforce, in hardware and from a trusted environment, the security requirements put on the key material.