SECURE IMPLEMENTATIONS OF CONTENT PROTECTION (DRM) SCHEMES ON CONSUMER ELECTRONIC DEVICES

Discretix

trust ▪ embedded

Contents

Introduction

Today's connected devices – including web TVs and hybrid STBs – embody a new era in consumer electronics, allowing unparalleled access to content and unlimited options to consumers. In this new age of over-the-top TV (OTT-TV), device manufacturers and service providers require robust content protection schemes that are approved by studios, yet are flexible enough to support multiple business models and new use cases. Companies such as Netflix are driving requirements for highly secure DRM implementations on mobile platforms. Recently, Netflix wrote a blog entry explaining that they cannot support Android-based platforms at this time because of insufficient security mechanisms for these platforms.

The need for secure DRM implementations is real, and it is happening now. Secure and robust DRM implementation, as defined by content owners, is mandatory in order to enable premium content licensing by the service provider. Such robust implementation requires an in-depth understanding of the security vulnerabilities of today's connected devices. This White Paper explains what is required to build a secure DRM solution. The first part of the White Paper develops a DRM threat model. The second part presents some approaches to a secure DRM solution and describes the Discretix secure DRM solution.

DRM Threat Model

In order to develop a DRM threat model, we start by reviewing a typical DRM flow. From this, we identify the assets in a DRM system. Finally, we define against what attacks the DRM assets should be protected.

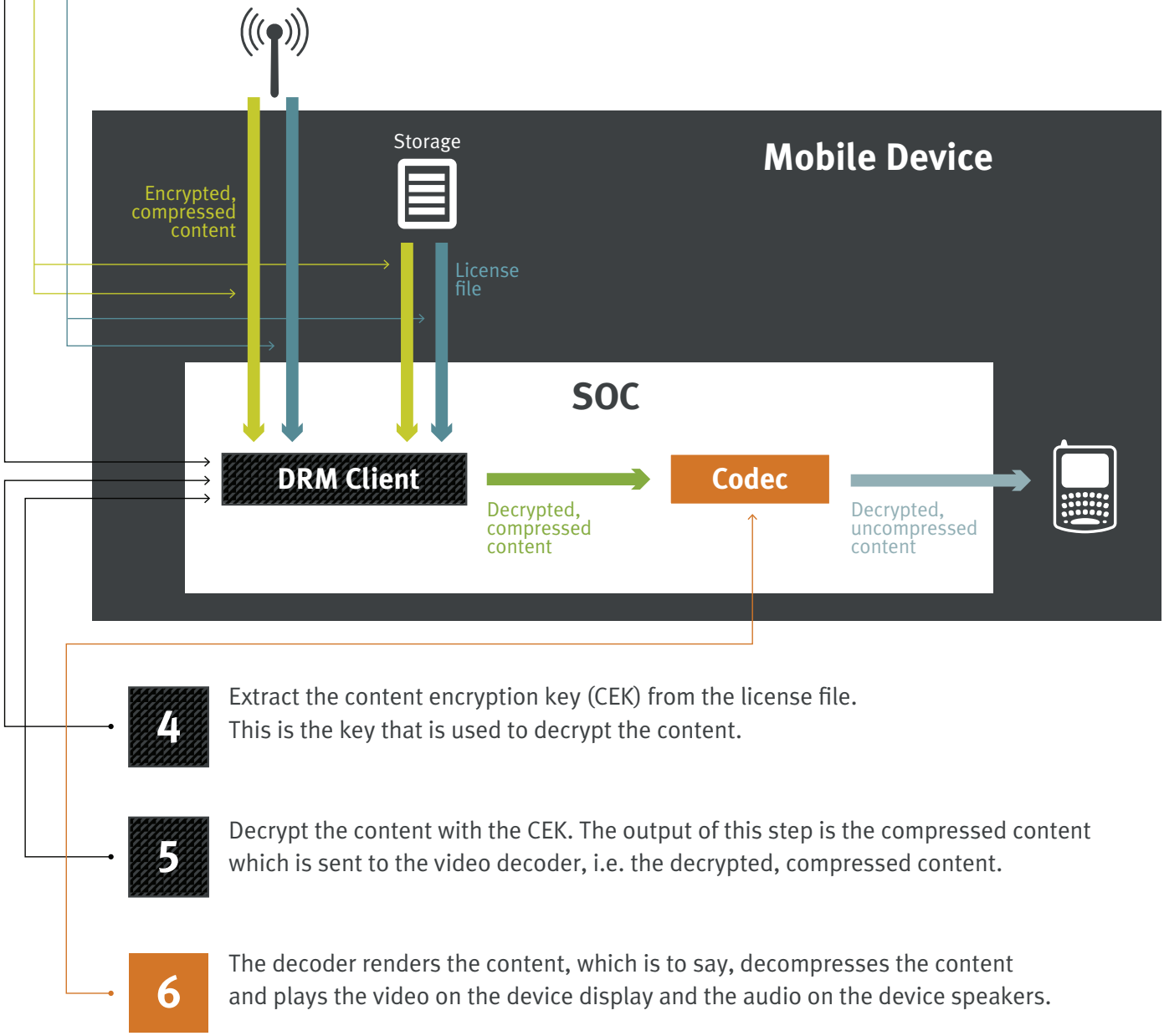
DRM Flow

Figure 1 illustrates a typical DRM Flow.

1 Get DRM-protected content. The content is compressed content that is also encrypted as defined by the specific DRM scheme used.
The content may be streamed on-line, or may be stored on the device storage.

2 Get the DRM license file associated with the content.
The DRM license file is obtained from a DRM server prior to consuming the content.
The license file may be stored on the device storage.
If the license file is not found, the DRM client will try to get it on-line from a DRM server.

3 Process the DRM rules from the license file.
An example of a DRM rule is that the content may be played only five times, as in the case of content that was loaded for rental.



4 Extract the content encryption key (CEK) from the license file.
This is the key that is used to decrypt the content.

5 Decrypt the content with the CEK. The output of this step is the compressed content which is sent to the video decoder, i.e. the decrypted, compressed content.

6 The decoder renders the content, which is to say, decompresses the content and plays the video on the device display and the audio on the device speakers.

Figure 1: DRM Flow

DRM Assets

The DRM flow described above illustrates the elements of the system that must be protected in a secure DRM system, i.e. the DRM assets.

1. The first set of assets that must be protected is all DRM private keys and license files that are permanently stored on the device storage. One example is that of a DRM scheme that includes system-wide keys which must be stored securely on the device. A second example is a private key that uniquely identifies the device in a DRM scheme ecosystem.
2. The second set of DRM assets is temporary keys. The most common example of temporary keys is CEKs that are extracted from license files during run-time.
3. The third set of DRM assets is the decrypted compressed content data that is passed from the DRM client to the codec. This is actually the “native” form of the content.

Note that DRM-protected content files are not included in the list of DRM assets. They are protected by means of the encryption as defined by the DRM scheme.

Threat Model

The threat model defines what kinds of attacks are considered relevant and which should be defeated or at least hindered by the protection mechanisms employed in a solution. Thus, the threat model defines the attacks that are considered in the scope of the solution. In general, the threat model will exclude attacks that are more expensive to mount than the value of the assets that may be obtained, while also considering potential attack scalability. There may be other reasons to exclude specific attacks from the threat model.

In a DRM system, there are two types of attacks that are considered in the scope of the threat model.

1. Software attacks. This encompasses all attacks mounted by software techniques, including attacks mounted by an attacker that has root privileges in the system.
2. Hardware attacks. This includes attacks mounted by monitoring the device hardware ports. In some cases, this may also include probing inter-chip signals on the device PCB.

Protection of Assets

Having defined the list of DRM assets that must be protected and the list of attacks against which they must be protected, we can summarize the mechanisms used to protect the assets. The following table summarizes the protection mechanisms used to protect the assets.

Asset	Protection Mechanism
DRM keys and license files	Secure storage
Temporary keys	Protected execution
Decrypted, compressed content	Secure content path

Although decrypted, compressed content must be protected, its protection has normally been considered out of scope for DRM requirements, or at most, has been considered a "soft" requirement. However, as noted above in the introduction, we have recently seen a push for high-security DRM implementations. The meaning of high security is exactly the added focus to protect decrypted, compressed content as a strict requirement.

It is important to understand some details of the protection mechanisms listed in the table.

1. Secure storage. This is an encrypted, integrity-protected database that is stored in the device permanent storage. Access to database items is controlled by some form of authentication, which in the case of DRM is the authentication of the calling application.
2. Protected execution. This is an execution environment that is protected from the main OS running on the device, i.e. programs running on the main OS cannot access the code and data memory spaces of the protected execution environment.
3. Secure content path. This is a data path from the DRM client to the codec that cannot be accessed by the main OS.

Each of these mechanisms can be implemented using software or hardware techniques. The implementation details are described following the next section which presents some security infrastructure requirements.

Security Infrastructure

Our discussion to this point has used a standard security methodology to derive the required mechanisms for a highly-secure DRM implementation. However, the discussion has focused on DRM-specific details without considering general system security aspects. In addition to the requirements discussed above, there are some fundamental security modules that must be present in any secure system. These security modules are required for the protection of the entire system including all of the DRM assets listed above.

This section provides an overview of these security infrastructure modules.

Secure Boot

The first security infrastructure module is Secure Boot. Secure Boot verifies the integrity and authenticity of the system code. It checks that the code has not been modified since device manufacture and also that the code is a valid version that originated from the device manufacturer. Secure Boot should support authorized software updates from the device manufacturer.

Secure Boot is essential for any secure system to ensure that the security mechanisms embodied in the code are not modified by an attacker. In our context, one such attack could be to modify the DRM code to copy all decrypted content to a removable memory card. In order to protect the Secure Boot code itself from modification, this module is normally stored in ROM in the SOC. This forms a Root of Trust for the system.

Secure Debug

The complexity of devices such as smartphones and tablets requires debug capabilities. Debug must be supported in the field as well as during the software development phase. However, since debuggers enable the reading of all memory locations as well as CPU registers, this capability can be used to extract secret information from the system. An attacker can start a DRM operation and then initiate a debug session to read out keys or content.

Secure debug prevents this attack by restricting in-field debug to authorized debuggers. Before a standard debug session can be started, the secure debug module authenticates the debugger to verify that it is allowed to enter the debug session.

Life Cycle State Management

A device goes through a number of life cycle states before being released to the field. The following table presents a simplified life cycle state flow.

Life Cycle State	Associated Functionality
Chip Manufacture	Full debug capability allowed. Enable secure provisioning of chip vendor secrets.
Device Manufacture	Most debug capability allowed. Enable secure provisioning of device manufacturer secrets.
Secure (in Field)	Only authorized debug sessions allowed. Add user secrets via security protocols.
Failure Analysis	Full debug capability allowed. All secret information is wiped from the device.

Since the life cycle state determines the security level of the device, it must be kept secure. The life cycle state is encoded in an on-chip NVM.

Cryptography and Secure Storage

A secure system must include cryptographic engines as well as a secure storage mechanism. The cryptographic engines must run in a secure environment and provide sufficient performance. More details about cryptographic engines are provided in the following sections.

The secure storage mechanism is described above and is further described in the following sections.

Software-based DRM Solution

Figure 2 presents a diagram of a high-security, software-based DRM solution. The DRM client and Video codec are compiled together into one monolithic DRM-codec object. In addition, the DRM-codec object includes software cryptographic engines and software-based secure storage.

A number of software techniques are employed to protect the DRM-codec object from reverse engineering and other software attacks. The internal program flow and data objects of the DRM-codec object cannot be accessed by any processes running in the OS, including malware with root privileges. Before we examine the details of the software protection techniques, we can see how this solution addresses the DRM requirements.

1. All DRM keys and license files are stored in the secure storage data base. They can only be accessed by the DRM client.
2. When the DRM keys and license files are loaded by the DRM client into its internal memory space, they are protected from eavesdropping by any other processes. The DRM client extracts CEKs from the license file to decrypt the DRM content. The CEKs are located in the system memory space but they are unreadable by other processes because of their obfuscated format (see the following section, Software Protection Techniques).
3. The decryption of the DRM content is done by the software crypto engines that are integrated with the DRM-codec object.
4. The decrypted, compressed content is sent from the DRM client to the codec using obfuscated means that are difficult to reverse-engineer.

The resulting solution protects all of the DRM assets according to the requirements of a high-security solution.

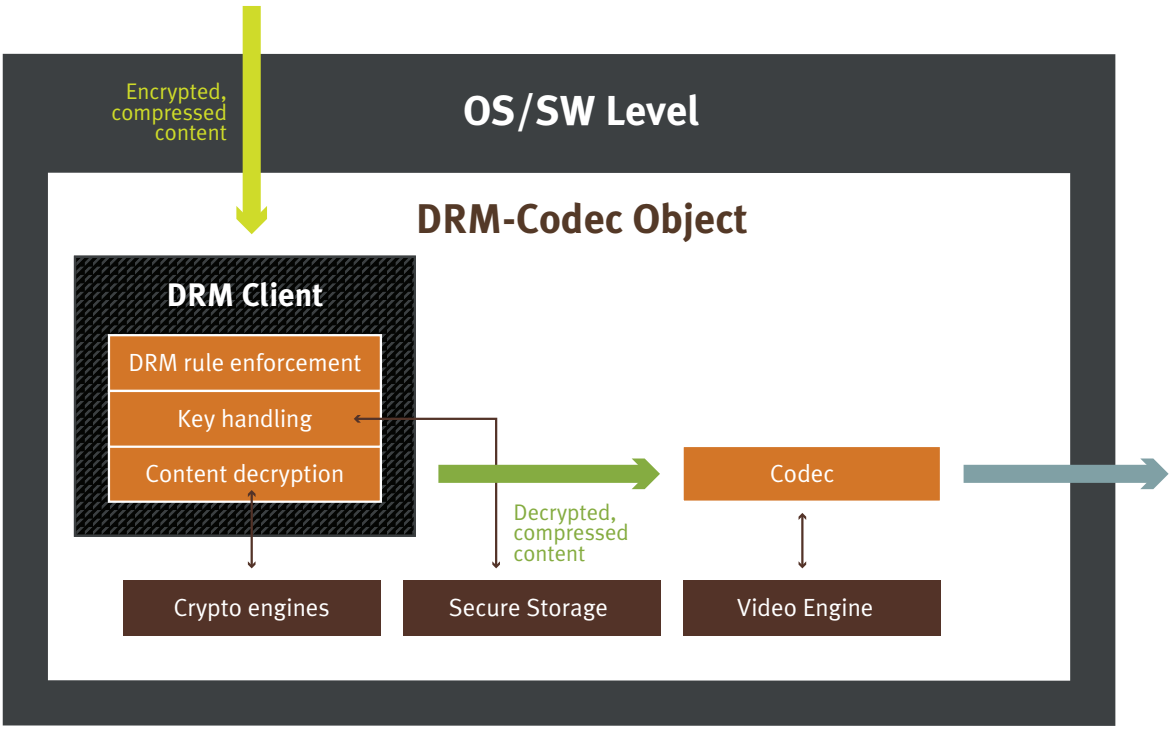


Figure 2: Software-based DRM Solution

Software Protection Techniques

A number of techniques must be used in order to make a robust software-only solution.

The first technique is software obfuscation. Software obfuscation is a method by which the control flow of a module and the data handled by the module are scrambled. First, some operations in the source code are replaced by obfuscated routines. After the module is compiled, the binary code is further scrambled. The first step is called source code obfuscation and the second step is called binary obfuscation.

By its nature, software obfuscation adds significant complexity to the module's operations in order to hide the original program flow and the plain data. This complexity results in a much larger binary image as well as slower execution. Some analysis is needed to identify the security-sensitive parts of the module and apply obfuscation only to those parts. This is important in order to limit the effects of obfuscation on code size and performance.

The second software technique is variability. Variability means that the functionality of a software module is modified according to an initial configuration setup. A software module is configured to function differently in different devices or even in the same device from power-on to power-on. This makes it difficult for an attacker to generate repeatable exploit code that is scalable, i.e. that can be made widely available to others.

The third software technique is renewability. The module is designed so that the core security operations can easily be replaced with an over-the-air update. If the core security functionality of the module is compromised by an attack, an update is released to change this core functionality to another method.

Putting it All Together

The above described techniques are employed together to provide a robust, software-only DRM solution. First, the DRM Client and the Video codec are compiled together as one monolithic object. This object is secured using source-code and binary obfuscation. The resulting DRM-codec object provides a secure environment that protects the DRM assets.

1. The secure storage data base is encrypted and integrity-protected by keys that are difficult to obtain from outside the DRM-codec object.
2. The DRM keys, license files and CEKs are opened only by the DRM-codec object.
3. Content decryption is done by an obfuscated cryptographic routine.
4. The decrypted, compressed content is passed from the DRM client to the Video codec within the DRM-codec object.

In addition to software obfuscation, the robust DRM solution must also employ Variability and Renewability techniques in order to “future proof” the solution, and protect it against the generation of exploits. Variability makes it difficult for an attacker to reverse-engineer a software implementation in a way that produces a repeatable result. This makes a specific implementation more robust. If a specific implementation is hacked, a new scheme is introduced by using the Renewability feature of the solution. Now the attacker must start all over again in attempting to hack the solution.

Hardware-Assisted DRM Solution

Figure 3 presents a diagram of a high-security, hardware-assisted DRM solution. The hardware-assisted solution is based on a Trusted Execution Environment (TEE). The TEE is separated from the main OS by hardware. This ensures that the main OS, and malicious applications running over it, cannot read from the secure memory which is dedicated to the TEE and cannot observe the program flow of the TEE CPU. Some examples of hardware-based TEEs are Discretix CryptoCell® and TI M-Shield™ Mobile Security.

The DRM client is partitioned into security-critical and non-critical parts. The security-critical part of the DRM client is executed in the TEE and the non-critical part is executed by the main OS.

For premium content that requires high bandwidth, a hardware codec is normally used in the solution. The hardware data engine of the codec is secured by placing it in the TEE. The codec control code may run over the main OS.

This solution addresses all of the DRM requirements.

1. All DRM keys and license files are stored in the secure storage data base. The data base is encrypted and integrity-protected with hardware keys that are only accessible from within the TEE.

2. When the DRM keys and license files are loaded by the DRM client into the TEE secure memory space, they are protected from being read by any processes running on the main OS. CEKs are extracted from the license file and only exist in the clear in the TEE secure memory space.

3. The decryption of the DRM content is done by hardware crypto engines that are located in the TEE. The CEK is loaded by the DRM client directly from the TEE secure memory space into the crypto engine and cannot be read from the main OS.

4. The decrypted, compressed content is sent from the hardware crypto engines in the TEE to the codec data engine in the TEE. This data transfer is protected from the main OS by the TEE hardware protection.

A hardware-assisted solution is generally considered to be more secure than a software-based solution. As such, it is easier to get a hardware-assisted solution certified as a robust DRM implementation for premium content.

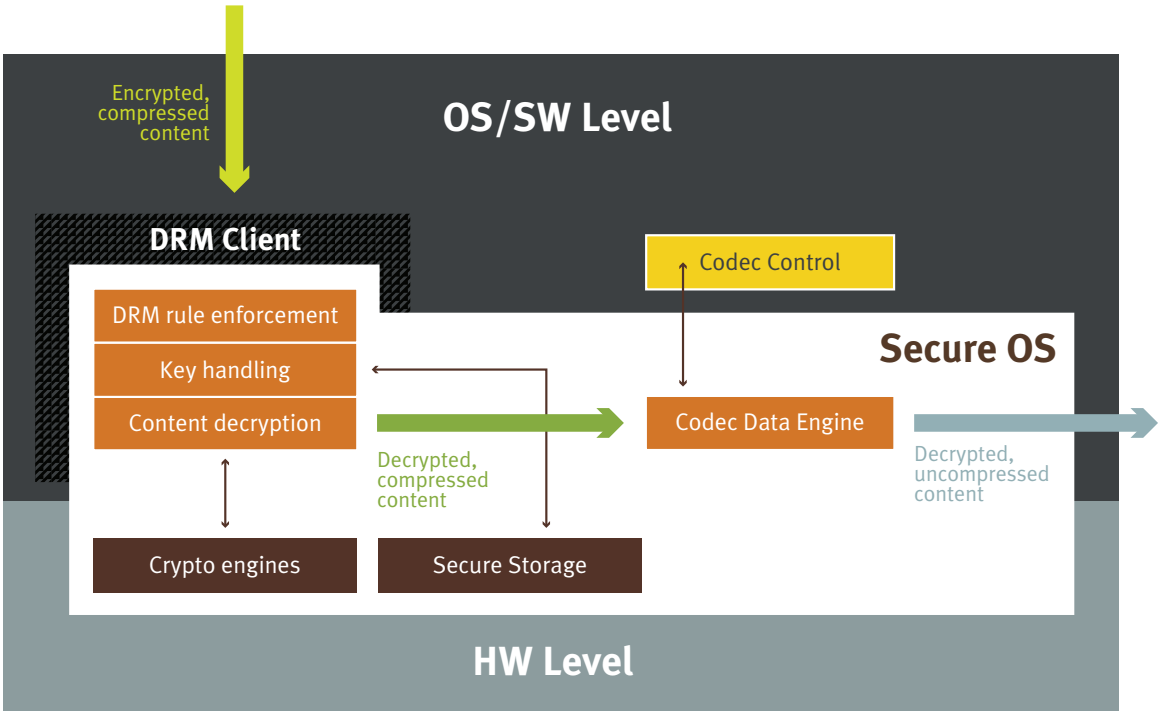


Figure 3: Hardware-assisted DRM

Discretix Secure DRM Solution

The Discretix multi-scheme DRM client is an embedded client that supports multiple DRM schemes while utilizing available security infrastructure mechanisms in a target platform.

Discretix Hardware-assisted DRM

Figure 4 presents a block diagram of the Discretix PlayReady client implemented on a platform with hardware-based security functionality. The hardware security features on the platform include a secure processor and RAM which comprise a TEE, hardware cryptographic engines and hardware-based secure storage as well as a secure content path. The PlayReady client is partitioned into non-critical and security critical parts. The non-critical part of the PlayReady client runs in the main OS and interfaces with User applications such as the Media Player and the Browser.

The critical part of the PlayReady client runs in the TEE. This includes the PlayReady black box and the security sensitive Discretix Middleware modules such as Secure Storage and Content Decryption. The Middleware modules are closely integrated with the hardware-based security functionality of the platform. For example, the Secure Storage will use a device-unique Root Key as the basis for deriving the encryption and integrity-protection keys. The Content Decryption module uses the existing hardware cryptographic engines in the platform.

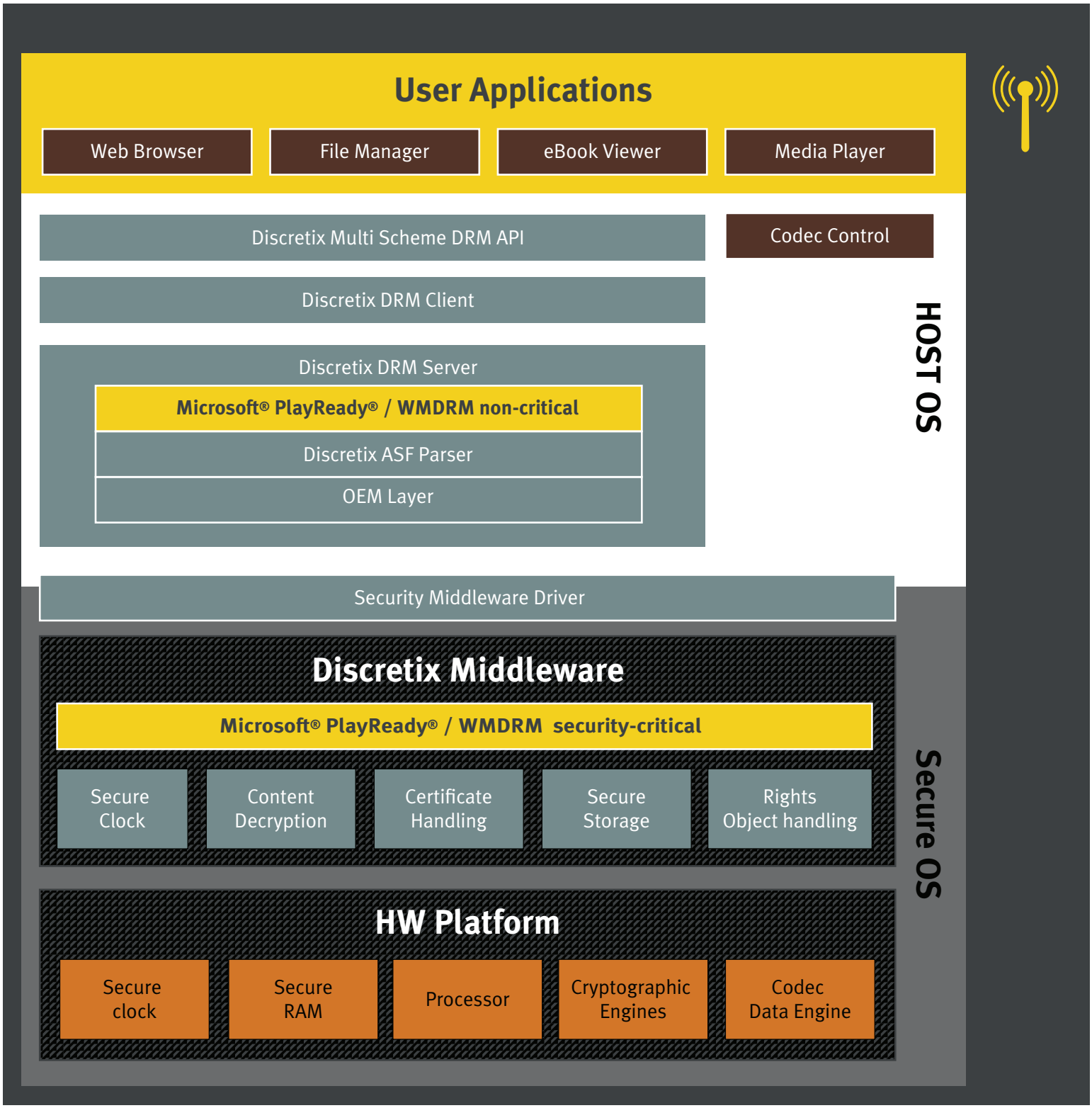


Figure 4. Discretix Hardware-assisted PlayReady Client

A secure content path is implemented by including the Codec Data Engine in the Trusted Environment. The Content Decryption module outputs the decrypted content directly to the Codec Data Engine within the TEE. The decrypted content is protected from interception by any processes running in the main OS.

The result is highly secure PlayReady client that meets the most stringent security requirements for a DRM solution.

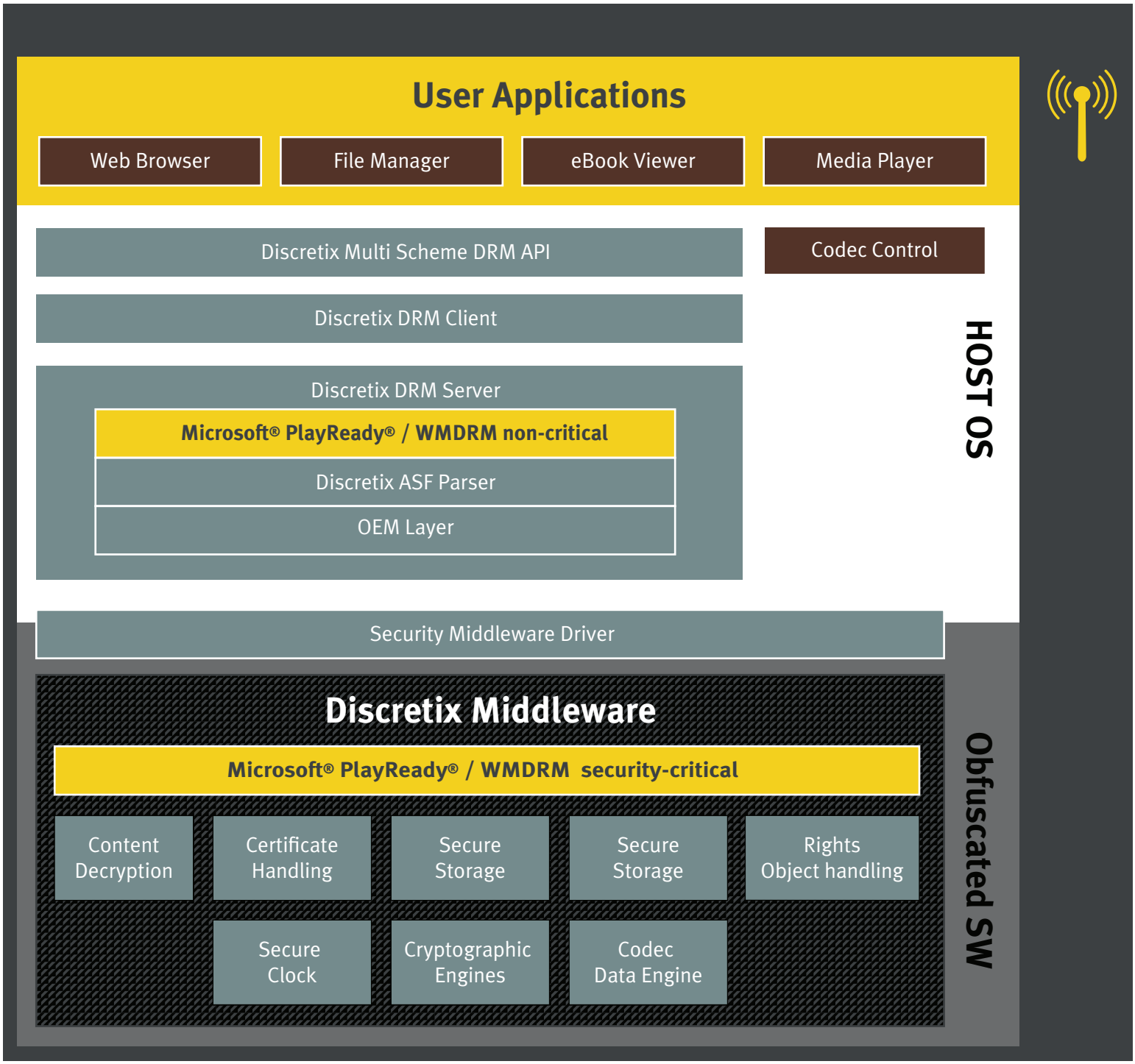


Figure 5. Discretix Software-based PlayReady Client

Discretix Software-based DRM

Figure 5 presents a block diagram of the software-based Discretix PlayReady client implemented on a platform without hardware-based security functionality. As in the case of hardware-assisted DRM, the PlayReady client is partitioned into non-critical and security critical parts. The non-critical part of the PlayReady client runs in the main OS and interfaces with User applications such as the Media Player and the Browser.

The critical part of the PlayReady client is "hardened" by applying to it the software obfuscation techniques described above. Security functionality such as Secure Storage and Content Decryption are implemented in software and included in the obfuscated object. A secure content path is implemented by including the codec data engine in the obfuscated object. All data passed from the Content Decryption module to the Codec Data Engine is obfuscated. Other process running in the OS cannot obtain any useful information or content by reading this data stream.

The result is PlayReady client that is secured by software obfuscation techniques on platforms that lack hardware-based security features.

Summary

The world of content is experiencing exciting new developments with the availability of premium content on multiple, high-performance devices. In order to enjoy this premium content, there is a strong requirement from the content owners for high-security implementations of DRM schemes on these devices.

This White Paper develops the threat model for a DRM solution. We presented two approaches to high-security DRM implementations: software-based and hardware-assisted. Each of these solutions addresses the high-security DRM threat model. We pointed out that hardware-assisted solutions are generally considered to be more secure than software-based solutions.

In conclusion, the White Paper presents details of hardware-assisted and software-based Discretix PlayReady DRM clients. Discretix PlayReady is ported on multiple platforms to provide DRM solutions to meet the most demanding environments.

For further information about Discretix DRM solutions and other products, please visit our website at **www.discretix.com** or contact us at marketing-dx@discretix.com.

Glossary

CEK	Content encryption key
DRM	Digital rights management
NVM	Non-volatile memory
OTT Delivery	Over-the-top delivery – refers to direct streaming of content over the Internet
OS	Operating System
PCB	Printed circuit board
ROM	Read-only Memory
SOC	System on Chip
TEE	Trusted Execution environment

www.discretix.com marketing-dx@discretix.com
Tel: +972 (73) 255 8800