Discretix
embedding security solutions

Challenges in Designing Content Protection Solutions

Hagai Bar-El
hagai.bar-el@discretix.com

Contents

1 Scope

This essay lists some of the challenges that are encountered when developing content-protection products.

The focus is on the "designing" part, rather than on general difficulties of content protection, such as: legal issues, scheme design issues, fair-use issues, marketing issues, user acceptance, moral considerations, etc. Enough can never be written on all these subjects, although more than enough has already been. In this essay, I deal mostly with the decision points and the difficulties that are encountered by someone who tries to make a living off designing and implementing content protection products. Our main focus is security.

This is the place to say that there are plenty of issues, probably enough for an essay ten times longer than this one. However, I am addressing the most typical ones, hard ones, and ones that are interesting to discuss. Specifically, I address issues of robustness, of deciding on which scheme to support, and of the frequent need to extend the protected data-path, beyond what was intended by the scheme designer.

I try to share a company's wisdom here. The following is some of what we learned and some of what we decided in our march towards establishing a sellable and effective content protection solution.

2 Robustness Issues

Content protection is all about security. Any content protection system must be designed with security as its first goal. This applies not only to the design of the scheme (which is beyond the scope of this essay), but also to every step of the implementation. It is one of the most significant challenges. The actual content being protected may not be the biggest asset that a security system has ever protected, but many properties of content protection systems make them one of the most challenging security applications. Consider the following:

- The user is part of the threat model. In most security systems the designer can at least assume that the valid user can be trusted by the system and only protect against external adversaries.
- The system protects valuable assets that have a large set of possible opponents, including quite resourceful ones.
- The system is installed on equipment of which cost is a major constraint. Content needs to be protected by consumer devices, not by high-end vaults such as ATMs. This applies in particular to physical security that is hard to afford on low-end devices.
- The system is highly distributed. When an ATM is found to be vulnerable, it can be called back, or visited in a timely manner. A compromised device can only, and only sometimes, be revoked from obtaining future content.
- Content protection systems involve complex data flows, including analog outputs, various connectivity options, and various storage mechanisms, all of which make them hard to design and to implement securely.

This chapter lists some of the difficulties that have to be addressed for assuring the robustness of the solution. Please note that I purposely do not address design considerations on the actual protection scheme and protocol, because these are a given for the developer.

2.1 Key storage

To protect content against reaching everywhere, it must be cryptographically bound to some entity. Binding is done using some key that only this entity has possession of. Therefore, the device, representing that entity or being that entity, must be able to store keys securely.

The key used for binding is the bare minimum. Most content protection schemes utilize a larger set of permanent keys, all of which have to be securely stored. Examples are domain keys, group keys, and class keys. Secure storage may also be required to store other pieces of secure information, such as links information (such as in *Marlin*), and rights objects that are cached, so they do not need to be processed per each playback.

2.1.1 The approach taken

Secure storage is one of the most essential intellectual property pieces that has ever been developed in Discretix. Two secure storage modules were designed (patents are currently pending). These two secure storage systems are utilized depending mostly on the type of data involved.

The first module is designed to protect keys and objects that are processed entirely by the hardware-based secure storage module. Keys that are stored in this secure storage module are not available to any entity on the device; not even to the operating system's kernel. These keys are never made available. When they need to be used, the actual cryptographic computation is carried out by the hardware-based secure storage module, after authenticating the requesting party. Authentication grants the right to make use of the key, not to retrieve it. This approach is somewhat similar to the approach taken by some smartcards when storing private keys.

The second secure storage module is made to allow protection of information that is to be processed by a trusted, or by an untrusted, application. Following successful authentication, internal content keys are loaded into the cryptographic engines and allow the data to be encrypted or decrypted as it is being streamed from, or to, the application. The application does not possess the keys that are used to encrypt the content, but following proper authentication it does get the right to access the protected data.

Clearly, stronger protection is provided by the first module, and this is the only module used when protecting long term device keys. The second module is useful when the data that is securely stored is to be processed by application code that shall not be made permanent, and that shall not become part of the secure storage module itself.

The design trend today is towards including as much functionality as possible in the secure storage module. As part of this process, we program the secure storage module to recognize and process various forms of rights objects.

Other than protecting the secrecy of keys and data, the secure storage modules also assure data integrity by cryptographic means. This is a valuable feature because the introduction of erroneous data is often the first step towards the exploitation of a buffer overflow. Obviously, this is not a sufficient line of defense against this threat, as malformed data can be introduced by other means as well.

2.2 Handling a shared environment

In some cases, the security model of the content protection module has to live with a shared environment in which it runs. Running in a shared environment has two immediate implications on the content protection module:

- Platform resources are shared with other applications. Other applications may thus be able to reveal data leftovers in public RAM and in CPU registers, and may read and/or modify data that is stored in secondary memory.
- If the content protection implementation is split into several trusting components that do not form a single process, a difficulty may occur when such processes need to authenticate each other. For example, if the media player is running as a separate process from the content protection code, the content protection module may experience difficulty in assuring that plaintext content makes it only to an unmodified, authorized, player.[1]

The complexity of the shared environment problem is directly derived from the type of applications that run on the platform; more specifically, from the level of trust that the content protection system has in these applications.

2.2.1 The approach taken

A closed platform A closed platform, for the purpose of this discussion, will be considered as a platform that only executes code that is known to not be able to cause damage to the security model of the content protection system. This does not mean that new code cannot be introduced into this system, but it does mean that *arbitrary* code cannot be introduced into the system. Every piece of code that is executed has been validated to not circumvent the content protection scheme.

Most consumer devices follow this model. They have a firmware image that is shipped with the device and in most cases this firmware image is never changed. In the cases of a firmware change, the update is done by the manufacturer and contains only manufacturer's code, or code provisioned on its behalf. Most consumer devices, as of today, do not run code that comes from arbitrary sources. We do not see this as something that is likely to change because allowing users to introduce highly-privileged arbitrary code also raises strong usability concerns.

Code from arbitrary sources can still be introduced without breaking the closeness of the platform, if this code is not native code but code that runs within some *sandbox*, such as when using interpreted languages. However, to determine that the relevant security assumptions of a closed system are met, one must be certain that any arbitrary code that is introduced to the system shall not be trusted[2] by the content protection module. To be sure of that, a close look is required not only on the sandboxing technique that is used, but also on limitations of the virtual machine that runs the code and/or that enforces the restrictions on it.

The above paragraph discussed a platform that is open to arbitrary code but yet can be considered as a closed platform. The opposite is also possible. In some cases, it is advisable to treat a platform as an open one, even though it presumably cannot receive and execute arbitrary code. This is advisable when relying on the inability of an attacker to introduce arbitrary code by utilizing buffer overflows, or other programming glitches in run-time, is deemed too risky.

In closed platforms our job is easier. All that needs to be done is to put the mechanisms in place to actually enforce the assumption on the pre-approval of the code and on its integrity. After all, having the system designed in a way that it supposedly uses only factory-issued

[1]This problem resembles a group of problems called "what you see is what you sign" in Smartcards. The latter is about the inability of a smartcard to determine the details of the transaction that it is asked to sign by an application running on a host, that is presumably untrusted.

[2]The term *trusted* is defined as the ability of one entity to violate the security model of the other.

code is one thing, and making sure that the system actually rejects other code, and prevents alteration of existent code, is another.

The typical solution is a *secure boot* mechanism. This mechanism runs a checkup (that cannot be overridden) upon each boot, and assures that no code has changed and that no new code was illegally introduced. If such illegal alteration is detected, the device is typically programmed to hang or to enter a special recovery mode that cannot be used for launching any useful attack.

Another component of the solution is a mechanism for secure software updates. The introduction of the secure boot mechanism prevents typical software update methods from working. Therefore, secure alternatives are designed to address the cases in which the software code needs to be altered, or replaced, after the device has left the factory.

An open platform An open platform is one that does not comply with the requirements of a closed platform, as described above. An open platform is considered, for the purpose of this discussion, to be a platform that allows arbitrary code to run in a mode that may potentially violate the content protection scheme. Our aim, when dealing with open platforms, is to make sure that this potency is removed, that is, to assure that arbitrary code does not gain the capability of violating the security model of the content protection scheme.

To achieve this, the common approach is that one must form “islands” in the system. These “islands” are sub-systems running approved code, that are not accessible by non-approved code, and that shall optimally be the only system components that handle protected content when it is in its unencrypted form, as well as perform other security-critical operations.

Forming the separation If the platform is open, which is the assumption here, the isolation must be based on a hardware component. A complete discussion of the rationalé behind this requirement is beyond the scope of this essay. Briefly, an open platform implies that all software code can be replaced or modified to accommodate the opponent’s needs, and so a hardware component is the only way to obtain assurance of something that the system does also when it is being attacked.

There are a few products in the market for forming these “islands”, and with which Discretix integrates its content protection products:

ARM TrustZone
: is ARM’s offering of a hardware-based trusted execution environment that is based on ARM11 processors. In addition to secure native code, the trusted mode can be used to run STIP applets (interpreted code).

Discretix CryptoCell SSDMA
: is part of CryptoCell 4 and above. It forms a secure execution environment that is based on a low-end processor and thus is suitable only for executing simple code in its trusted mode.

Discretix Gatekeeper
: is a complementary part of CryptoCell 6 that provides a processor-independent trusted mode, based on any one of a few existent processors. The trusted mode can be used to run the same native type of code that the processor runs when not in trusted mode.

A separate processor
: can be used to run the code that needs to run in isolation.

Discretix ESC
(temporary name) is a framework that allows isolated code to run on the embedded flash chip which has its own processor. This is a variant of the separate processor option.

Defining the isolated code Once the facility is in place to run some of the code in isolation, this part of the code needs to be defined. We need to draw the line between code that runs in the secure-mode cage and code that runs normally. The code that is run within the secure mode is hereafter called "Level-1 code" whereas the code that does not run within the secure mode will be referred to as "Level-2 code".

Level-1 code must be fundamentally simple, and easy to assure. Most importantly, it shall not rely on the behavior of any Level-2 component to retain its robustness. The modules that comprise the Level-1 code are the root of trust of the system. Level-2 code calls routines of Level-1 code, usually to obtain core security services, while performing the rest of the operations that are required by the application.

The system security is designed around an ever-standing assumption on the trustworthiness of Level-1 code. This code has its authenticity and integrity assured[3], was validated for robustness, and runs uninterruptedly. The level of trust that the system has in Level-2 code, provided that Level-1 code is not compromised, is one of the following:

1. The system does not trust Level-2 code, given Level-1 code is not compromised. That to say, no possible compromise or replacement of any of the modules of Level-2 can violate the security of the system.
2. The system exercises trust in Level-2 code independently of Level-1. Even if no Level-1 code is compromised, a compromise of Level-2 code can violate the security of the system, and thus cause damage, but damage that is smaller[4] than the damage that could have been caused had the opponent compromised a module of Level-1.
3. The system has trust in Level-2 code as above, but such that a compromise of Level-2 code may, in some cases, lead to damage of magnitude that is similar to the magnitude of damage that can be caused by a compromise of Level-1 code.

The third type of trust is seldom the case with well-designed systems. It indicates that the distinction between the two levels, in terms of the trust of the system in them, is useless. If a Level-2 code compromise causes as much damage as a Level-1 code compromise, then there is no point in having Level-1 at all.

The case in which Level-1 code is compromised is not discussed, because it is obviously a scenario that will violate the security model of the system. Consider the following:

- If the system has no trust in Level-1 code, but does have trust in Level-2 code, then there is no point in isolating Level-1 code at all.
- If the system has no trust in any code of either levels then it is not a security system, because there is no code that its compromise causes damage to the security model. If an opponent cannot cause damage, no matter how well he succeeds, then there is no security challenge.

What we form is a distinction between modules requiring two different levels of robustness. The required level is based on the level of trust that the system has in each component, while

[3]Such assurance can be achieved by mechanisms such as *secure boot.*
[4]damage — usually measured in terms of cost to the system.

also following the independence rule between the levels. This independence — the lack of trust of Level-1 code in Level-2 code, is mandated; the separation cannot really be enforced in practice otherwise.

In the optimal case, the system does not have any trust in Level-2 code (trust option #1). That is, compromising the less-guarded part of the code can cause no violation to the security model of the entire system. This is not always the case, however.

Minimizing trust in Level-2 code Level-2 code does not run in isolation from the rest of the system as Level-1 code does. Making Level-2 code modules untrusted by the overall content protection system (in other words, making compromised Level-2 code unable to violate the security of the system) is not trivial. An easy step forward is making the potential damage following a Level-2 code compromise smaller than that of a Level-1 code compromise. To achieve this, we define Level-1 modules as the ones handling the storage and usage of long term keys, such as device and domain keys, and credentials that the device needs to authenticate itself. Also, we develop modules in this level to handle the renewability mechanisms that naturally use permanent keys and code.

The code for handling long term keys (storage and cryptographic processes) can be isolated. It can utilize interfaces that do not require trust in the customers of the module, and the code is relatively simple and can, in some cases, be implemented in hardware. As long as Level-1 code is not compromised, a compromise of Level-2 code cannot possibly lead to the exposure of the long term keys.

The content protection system still has trust in its Level-2 code because if it is compromised then content keys can be let out, plaintext content can be exported, and, if other content-protection operations are done by this level, various other risks may apply, such as by changing the copy control information, by alteration of metering data, etc.

To reduce the scope of damage that may be caused by attacks against the Level-2 code, all content protection actions could be done by Level-1 code. Unfortunately, this is easier said than done. The following difficulties are met when attempting to include the entire code base for content protection and playback as Level-1 code:

- Content decoding and playback requires code that is too complex to assure. Recall that a bug in Level-1 code can lead to the complete collapse of the system.
- Having the codecs and drivers run as Level-1 code implies that all drivers and codecs need to be fully trusted and harder to update.
- Running the codecs and drivers as Level-1 code implies a strong performance penalty on most platforms that allow such code separation.[5]

One could consider including the entire content protection agent as Level-1 code, while leaving the playback modules (media player, codecs, and drivers) as Level-2 code. The independence requirement is met because a compromise to the playback modules cannot result in a compromised content protection application. However, in this case as well, a compromise of the Level-2 components still has strong implications on the security of the system. A rogue or compromised player can request the DRM agent (in Level-1) for plain content, presumably to play it back, and record this plain content in the clear, for illegal distribution.

Regretfully, there are no known magic solutions here. All remedies from that point on are platform specific. On most platforms, the entire content flow cannot be implemented as

[5]On ARM TrustZone, for example, each invocation of secure mode costs roughly 200 CPU cycles, twice.

Level-1 code without introducing strong undesirable side-effects. On the other hand, as long as plaintext content is also handled by Level-2 code, it cannot be assured that content is not leaked out following a Level-2 code compromise.

Platform-specific cures can sometimes be found, though.

In some hardware configurations, the content protection agent passes decrypted content to the player just so this player passes this content further down to a multimedia coprocessor. In some of these cases, the multimedia coprocessor has its own cryptographic capabilities. If this is the case, then a secure association can be formed between the multimedia coprocessor and the Level-1 code that consists of the content protection agent. This allows us to form an encrypted tunnel and thus eliminate all trust in the Level-2 components that form the player, and perhaps in some other components as well. The multimedia coprocessor still needs to be trusted, but this coprocessor is typically a closed system, so it is a good security trade-off.

In some other cases, proprietary OS kernel tricks can be played to make sure that the player and the codec are restricted from performing certain OS-level operations. The blocked operations are those that are required for content leakage but which are not useful otherwise. For example, a codec can often be prevented from writing to memory addresses that do not belong to the memory-mapped playback device. The player may also be prevented from utilizing secondary memory in write mode.

Some operating systems provide processes with the ability to determine the identity and signature of other processes to which they provide services. When this feature is available, it can be used by the content protection agent to determine the identity of the player to which it provides plaintext content.

2.3 Renewability

We strongly believe that a content protection system that does not offer adequate renewability is broken by design. Years of security experience teach that the only commercial systems that are never broken are those systems for which there are no persistent adversaries. Unfortunately, content protection systems are not such. There are too many opponents, the system is too complex and its operation environment is too untrusted, for us to believe that whatever we do will retain its attack-resistance forever. We believe that a truly secure content protection system is only a system that is capable of reacting to *successful* attacks. This is what renewability is all about.

There are three levels of renewability:

keying renewability
: is the ability to replace compromised keys;

implementation renewability
: is the ability to fix the actual implementation of the content protection scheme, was it found to be broken;

scheme renewability
: is the ability to fix the actual content protection scheme, if the scheme is found to be broken.

Discussing implementation, the third level of renewability is out of scope.[6] An implementer

[6]For more information on introducing renewability into standardized content protection schemes, the reader is referred to a Discretix contribution to the Open Mobile Alliance (OMA) Mobile Broadcast group (BAC-BCAST), *Challenges of Standardizing Renewable Broadcast Security*, which can be found at `http://www.discretix.com/PDF/Challenges_of_Standardizing_Renewable_Broadcast_Security.pdf`

cannot do much against a broken protection scheme. However, we believe that great care must be taken when addressing the other two types.

2.3.1 The approach taken

Renewability of the implementation is addressed by several techniques that allow for update of essential code parts, also through insecure channels. These methods can be implemented so they are able to replace almost every non-ROM component in a secure way. Generally speaking, a good implementation renewability mechanism is such that can replace any component of the system that is at some time to be found broken. Determining in advance what the to-be-broken components are is mostly an act of guessing. Therefore, a general guideline is to build the renewability mechanism in a way that it can replace as much as possible of the system. This is, of course, true only as long as the update can be done securely. It shall always be kept in mind that if the renewability mechanism ends up being broken, then the scope of possible damage may easily become as wide as the scope of possible renewability. Renewability is a strong sword against attacks, but it is a double-edged one.

The renewability mechanism that was designed in Discretix is based on a small core module that runs as Level-1 code and that is for the most part implemented in hardware. The code is small enough to be thoroughly tested and makes use of key material that is not used for other purposes (and thus which is less likely to leak).

The module responsible for renewability has several security objectives to address, and it addresses them. The main ones are hereby listed:

- The updates must be verified to assure that they come from an approved source and that they were not modified in transit.
- The updates should be encrypted to protect their confidentiality. Security through obscurity is a bad practice for itself, but is a desired add-on in this case. We may assume that peer-reviewers receive the code for review by other means. More importantly, an update may also include keying information that needs to be kept private, so updates shall be encrypted.
- There must be some protection in place against taking the update objects out of context, for example:
  - It shall not be possible to mix up updates that were meant for different models.
  - It shall not be possible to use cached update packages to make the system revert to an older code version.
- Devices that were not updated even though they needed to should be prevented from obtaining access to new assets.

2.4 Buffer overflows and other programming glitches

Buffer overflows account for at least 80% of the software flaws; some say more. A single buffer-overflow vulnerability, a single format string attack, or any one of other possible programming errors, can make an implementation vulnerable to attacks that can be widely disseminated using *exploit code* in a matter of days. Such coding errors can practically kill a content protection product even if it was designed securely from all other aspects.

2.4.1 The approach taken

Unfortunately, there is no known way to just scrap off all potential buffer overflow flaws in an automated way. Such flaws can only be eliminated by giving the right skills to programmers and by reviewing and re-reviewing every piece of code written. In Discretix the following three-tier approach is taken to prevent products from being subject to buffer-overflows and to results of other coding glitches:

1. All programmers go through secure coding training.
2. Code that is more sensitive in terms of secure coding is manually checked by a professional code reviewer.
3. QA tests include black-box stress-tests that are configured to detect possible buffer overflows before they are exploited.

Beyond that, the separation of the code into two parts, Level-1 and Level-2, provides a fire-walling effect, assuring that even if a code module of Level-2 is completely taken over, it cannot circumvent the security model of Level-1 code, protecting the more valuable assets of the system.

2.5 Random number generation

Most cryptographic systems are heavily reliant on good random number generation. Random numbers are used for internal key generation (both for symmetric and asymmetric keys), for salt generation and for nonces that are required to assure freshness of messages. Unfortunately, good random numbers are hard to come by when dealing with deterministic machines.

Pseudo-Random Number Generators (PRNGs) come for the rescue by allowing random values (referred to as "seeds") to be expanded into practically endless amounts of random numbers. These mechanisms, however, as robust as they may be, still rely on seed values that have to be random[7]. Some good source of randomness is thus required, and the resulting random values should be used to seed a safe PRNG circuit.

The quality of random numbers may vary, depending on the use. In some cases the random values do not even need to be random, but just to not repeat (other than by the valid chance). In other cases, the random values are used to generate keys of very short lifetime. In these cases, a mediocre random seed that is fed into a PRNG, that runs for a while before generating the key, may be adequate. This is because brute-forcing the key through the seeding process will take longer than the lifetime of the key. In other cases, however, the *quality* of the random numbers, hence, their resemblance of a truly random sequence in which each value is equally likely to appear, is a real must and shall not be taken lightly.

2.5.1 The approach taken

. . . So we do not take it lightly.

As a design principle, we do not take the shortcuts described in the previous paragraph. The random number generation component is treated as a one-size-fits-all component and is thus designed by the most stringent requirements. Also, the random number generation

[7]When we specify that numbers are *random* we do not necessarily claim randomness by all means, as the definition of randomness is somewhat philosophical. What we mean is that an opponent cannot discriminate between these numbers and numbers that are generated at random, in which each possible value has the same probability of occurring.

component is permanent (often hardware based), and thus has to accommodate high levels of security as well, as required by some schemes, and as may become required by future schemes.

The PRNG module is a full FIPS-approved PRNG. We eliminate using Linear Feedback Shift Registers (LFSRs) and other linear-strength shortcuts as complete PRNGs. All PRNG designs are based on presumably-secure ciphers and hash functions.

The biggest problem, as usually, remains the seed source. We accommodate two types of seed sources:

- A Discretix IP-based digital-only seed generation circuit. This is a hardware component that generates high quality seed values using digital logic only.
- A platform-specific seed source that can be wired into the PRNG. Such a weak source may easily become the weakest link in the security chain of the system. Therefore, extensive statistical tests are carried out in the lab, to determine whether the output from this source can indeed be used as a reliable random seed. Moreover, the quality of the incoming seed is checked in real-time by the PRNG itself. The PRNG refuses to seed with values that are boldly weak.

The output of the PRNG, which consists of the pseudo-random numbers that are to be used, is constantly monitored to assure that the PRNG did not somehow get stuck in a faulty state in which it produces constant values. If such a case is detected, the system is reset so these non-random values are not actually put to use.

If good seed values are too scarce, the PRNG can be programmed to use the secure storage component so that it can seed less often. This is done by retaining the internal state of the PRNG across power-cycles. In the most extreme case, the PRNG can be seeded only once in its lifetime, during one of the manufacturing phases, and retain its internal state from that moment throughout the life-cycle of the device. This is a good solution for the cases in which there is absolutely no reliable seed source on the platform itself. Special integrity measures are used to assure that an opponent cannot use the stored state to influence the next random values that are generated.

2.6 The analog hole

The "analog hole" is one of the biggest barriers to the success of content protection. Whereas the digital content is protected, this digital content eventually ends up in an analog form. It is inevitable — it can only be consumed in analog form.

The only way that is known in the art for somehow protecting analog content is using watermarks. Watermarks are human-senses-wise unnoticeable alterations that are introduced to the content. These alterations embed encoded information such as the copyright owner, the copy control information, or the identity of the licensee. Such watermarks can be used for the following purposes:

- To embed information about the copyrights, the identity of the copyright owner, etc.
- To deter potential pirates by allowing a trace-back to the violators of content license.
- To embed information that can be used by players to determine if they are allowed to play the content or not. For example, a watermark embedded in a film while being projected at a cinema can be detected by a player. The player will subsequently refuse the play the film since it was illegally recorded in the cinema hall using a camcorder.

- To embed copy control information that can instruct a client-side content protection agent on how to process the content, such as: playback only, copy-once, low-resolution playback only, etc.

Watermarking is a relatively new area of information security, and is subject to a lot of research. Robust watermarking schemes are still hard to come by. Such schemes need to resist a wide range of attacks, such as distortion, analog manipulations, combination of content sample variants from several sources, and more. As of today, some of these challenges are still to be met. For example, most watermarking schemes cannot yet cope with a large number of pirates colluding against the protection of a title, each one bringing in his own copy of the pirated title, so to form a joint pirated title object which cannot be traced back to any of the pirates. One other difficulty is that the watermarking scheme is such that needs to be publicly-detectable, that is, the detection of the watermark shall not require knowledge of secrets. Many professionals in the field claim that a publicly-detectable watermark cannot possibly be secure, because effectively removing it is just a matter of iterations of trial-and-error against a readily-available oracle.

Mitigation of the analog threats to content is beyond the scope of the implementation designer. The content protection scheme is responsible for adopting a certain watermarking scheme in order to be able to address this complex threat.

The analog hole is a threat that the implementer faces, but can do nothing against.

3 Fragmentation Issues

One of the biggest hurdles we face when implementing content protection schemes is that there are so many of them around. For the mobile environment alone there are at least five different protection schemes, without a single one being an obvious winner. The strong market forces behind each one of these schemes and the strong political forces behind each one of the proposed standardized schemes, makes one believe that these various approaches to content protection are not likely to converge into one agreed solution; not during the lifetime of the author, anyway.

In some cases, the divergence is a matter of political and market forces alone. In many other cases, however, the divergence is a straightforward result of the different set of requirements and the different priorities that each content protection designer has set, such as:

- Maximal code size for the client
- Processing capabilities of the client platform
- Typical bandwidth consumption
- The implication on the BOM of the client device and its associated cost
- Time to market
- Support for fair-use, and the very definition of fair-use
- Suitability for unconnected devices
- Desire for centering the scheme on the user's PC, on the set-top-box, on the network operator's server, etc.

Other than the race among various specific content protection schemes, there is a higher-level debate on open-standard-based content protection schemes versus proprietary solutions. The recent MPEG-LA saga raised some doubts about the meaningfulness of open standard solutions as free-to-all designs, but distinction is still made between schemes that are open for anyone to implement versus designs that come as pre-implemented turn-key solutions.

Even in this limited-scope dilemma there is no reason to believe that only one approach will survive. Both standard and proprietary solutions have their pros and cons, and no one is by far better than the other for just any service provider. There is a large criteria set that needs to be considered when choosing between a standard versus a proprietary solution, for example:

- Standards prevent fragmentation, increase competition, and thus reduce cost.
- Proprietary solutions might lock the customer to the vendor by introducing high costs of replacement.
- A standard solution is less likely to serve the interest of a single company.
- Many professionals claim that a standard solution is inherently more secure, and give convincing reasons.
- Many professionals claim that a proprietary solution is inherently more secure, and give other convincing reasons.
- Proprietary solutions have one owner who is responsible for the functionality of the system with no way to shift liability.
- Proprietary solutions are clean of IPR issues. Once the product is bought, it presumably has no hidden costs.

To sum up the issue, there is no single winning content protection scheme, and there is not likely to be one in the near future. We must assume that we may have at least one standard and one proprietary scheme, and we are very much likely to have more than one of each. More importantly, even if we take that we are to be left with one (or two) winning schemes, we are still helpless in trying to determine which one (or two) it would be.

. . . But we need to implement something.

3.1 The approach taken

Waiting for the market fog to clear off is seldom an option, and it is certainly not an option in this case. Designing and implementing software takes time, and designing and implementing hardware takes more time. The hardware is as fixed as hardware can be, and the software is embedded, so both need to be produced in high quality from the start; this translates to extended development time. Development must start before the problem of what to implement can be grasped to its full extent. The situation is thus one that calls for an educated guess.

The approach taken by Discretix is the *multi-scheme* approach. The basic idea of this approach is that it is better to pause for a moment and look at content protection from a higher altitude than to rush into implementing a suggested scheme that at one moment seems to be most promising. We found no point in holding until the market stabilizes – it has been almost two years since that point in time and the market has not stabilized yet. It may never. We did take some time, however, to extend the design beyond the one or two schemes that were considered.

The result is a multi-scheme architecture. This architecture utilizes common building blocks and a common security infrastructure, and implements various schemes on top, using software modules that can be added or removed at will. Such modules today are available for OMA DRM V2.0, for CPRM, for WM DRM, and soon for the emerging OMA-BCAST

1.0. Adding support for other schemes is significantly easier than implementing the client side from scratch; many building blocks can be reused.

There are advantages to the multi-scheme approach. It reduces development time, as many common components were only implemented once. This also saves on code size. It allows extendability of the system to support newly emerging content protection schemes. Understandably, it increases the robustness of the solution, because many of the security components are hardware based, were designed and thoroughly tested once, and serve all schemes. It would not be rational to expect each and every content protection implementation to utilize its own hardware-based secure storage, for example, or to implement its own secure data path to the multimedia coprocessor.

Most importantly, this approach to content protection design eliminates the need to make a firm decision on something that cannot be safely decided yet, and thus saves us from having to bind the success of a product line to results of guesswork. We still need to decide which content protection schemes to support, but the multi-scheme approach allows us to:

- implement more schemes for less added code-size and for less added security risks, per-scheme, and,
- be able to enhance the capabilities of the device with new schemes, using a secure update mechanism, without calling the device in.

4 Extending the Data Path

In some cases, implementation requires that the data flow that was assumed by the content protection scheme designer, is extended. In some other cases, it is desired to extend the data flow for usability reasons.

Following are a few examples for cases in which the data-flow needs to be extended beyond the original design of the content protection scheme.

- Separating processing modules that are atomic by the design of the content protection scheme, e.g. separating decoders from content management modules or running part of the scheme on a host processor and part on a multimedia coprocessor.
- Expanding storage of sensitive material onto external devices, for example, introducing removable flash cards as a medium for content mobility. In some cases, the content protection scheme supports this out of the box. In other cases, the content protection scheme may be able to utilize removable storage devices as *dumb storage* only.
- Introduction of external playback devices. The device that obtained the content is often assumed to be the only device that plays it. This limitation prevents users from playing protected content on external playback devices.

Extending the data path beyond the path which the scheme was designed to protect is a tough problem. Digital content is knowledge (information in bits), and knowledge can be duplicated. A must for content protection is thus end-to-end protection (via encryption) of content from issuer to consumer, so bits are never seen (and thus obtained), just consumed. Such end-to-end tunneling relies on some sort of a security association between the endpoints. The types of endpoints are scheme specific. If a product extends or complicates the data flow beyond what the scheme supports — it must extend the security association as well.

4.1 Extending out of the device

Extensions that are done for usability purposes typically extend the data flow to components that are physically external to the device. One example was given of allowing content mobility using removable cards. Another example is allowing content to be played back on a device that is not the device which was used for license acquisition. Some content protection schemes support these use-cases out of the box. We are concerned about the ones that do not.

Let us look at OMA DRM V2.0 as an example. This content protection scheme draws a security association between a server of a rights issuer and a client application running on the device. The content is encrypted, and the rights issuer's server provisions the content key to the DRM client on the device, using a protocol called ROAP. If one wants to allow the device, to which the content was bound, to send the content to be played back on another (compliant) device, or store the rights to use the content on a removable media, it necessarily extends the data flow beyond the structure that the standard supports. Retaining the protection for the content in these scenarios requires additions that are not part of the standard, as well as possibly some legal settlement against the rights owner.

To achieve the above, both the source and the destination devices need to support a common content protection scheme, or two content protection schemes that inter-operate on the above functionality. The content protection scheme does not need to be the same scheme that was used to acquire the content or the license. It can just as well be a "gluing" scheme that is used just for securing the link.

As of today, some content protection schemes are ready for introducing external playback devices (playback devices that are not the devices that were used to acquire the license). Examples for such are *Marlin* and *Windows Media DRM*. Marlin is the most flexible scheme we have seen in terms of playing content on various devices, according to a fair-use policy. OMA DRM V2.0 does not yet allow playback on devices for which the rights were not initially purchased.

When the content protection scheme does not allow for playback on external devices, separate "gluing" schemes may be used. Various schemes such as the HDCP, or the less common SVP, may be used to establish a security association between devices.

4.1.1 The approach taken

Discretix has its own tunneling technology that allows two devices to exchange content securely following mutual authentication, even if this is not supported by the content protection scheme. However, we by no means suggest that this is the only scheme that shall be implemented for this purpose. Since interoperability is a key issue, we offer support for other schemes as well.

The approach we take to extend the data path into external playback devices is similar to the overall multi-scheme approach discussed earlier in this document. That is, we strive to provide a generic platform that can support a large set of content protection schemes and that can be easily enhanced to recognize new schemes. We see this as the only way to assure (to the extent possible) that interoperability will be achievable with all devices.

When designing content protection for consumer electronics we also realize that when such a device is met by a mobile device for content playback, it is likely that the mobile device has a narrow set of supported schemes. We therefore find it important to make sure that consumer electronics devices can interact with mobile devices using the to-be-defined OMA DRM extensions, or using other DRM schemes that are used on mobile devices, such as Windows Media.

The need for supporting mobile DRM schemes on consumer electronics has another aspect to it. Playing back content from mobile devices is different from playing back content coming from set-top boxes or disc players. When a TV-set (for example) plays back protected content that is received from a disc player, it receives the raw content in an encrypted form, such as by HDCP, deciphers the content and plays it. Convergence with mobile devices is likely to be different. The raw content will not necessarily be streamed from the mobile device. The mobile device will not necessarily have the capability of streaming high definition content at adequate speeds to the TV-set. Moreover, it is not granted that the mobile device will actually store the same content file that is to be used when playing on a large TV set. A more likely scenario is that the mobile device will have in its local (or removable) storage a reduced version of the content, suited for the screen size and audio quality of a mobile device. When a mobile device connects to a TV-set for content playback, only the rights are transferred from the mobile device, and the TV-set executes the full DRM scheme, including the content download, to use the license arriving from the device to decipher the high-fidelity content object and play it.

The conclusion that was reached is that the consumer devices that are to receive content from mobile devices are likely to be expected to support the native DRM schemes that are used by the mobile devices. Some DRM schemes already allow such extensions, such as CPRM or Marlin. Some other schemes, such as OMA DRM V2.0, still have it in the works.

If the mobile device does not support a scheme that allows an extension to be made towards external playback or towards rights storage devices, a plug-in can be added, as long as it is supported on both sides. Discretix has implemented such plug-ins that allow OMA DRM content to be securely played on external devices, as well as for OMA rights objects to be mobilized using removable flash cards that support the required security functionality.

4.2 Extending within the device

Implementation considerations may also lead to the need to extend the data path, typically within the device. This can occur if the underlying implementation involves separate physical units that form a single logical unit in the scheme-level. The "content protection agent" is often considered by the scheme to be a single logical unit. If its implementation spans over more than one physical unit, such as in a multi-chip solution, one must address the data that is passed in the clear, if there is such.

In some cases a multimedia coprocessor is used. Plaintext content may be transferred over a visible bus from the host processor that executes the content protection scheme, to the multimedia coprocessor that runs the codecs and/or that speeds up the media display and/or playback. This problem of plaintext content over visible bus lines can grow even bigger if the multimedia coprocessor happens to use shared memory of the main host as a scratchpad for intermediate values during its computation.

Another example comes from mobile broadcast. Three separate physical entities may be responsible for the playback of received content. The first is the DVB chip that mainly receives the radio signals. Second is the host processor, or an application processor, that runs the broadcast security scheme and extracts and processes traffic keys. The last is a decryption module, which may be run on separate hardware, to perform speedy decryption of the broadcast stream. In other broadcast security configurations there is a DVB chip to receive the data, a SIM card to perform some of the key management operations, a host processor (or an application processor) to control the system and to perform some other part of the key management, and decryption is done either by the host processor or by yet another hardware component dedicated for this purpose. Some mobile broadcast schemes are suited for the separation of these components — some are not. For example, in some cases a secure channel exists between the smartcard and the descrambler; this is not always the case, though.

4.2.1 The approach taken

In cases where data may be going in the clear, and if the content protection scheme does not provide its own remedy, an implementation level solution is sought for either by making sure the added path is not exploitable (e.g., by using a non-traceable bus, or by passing only encrypted data over the bus), or by protecting the data using encryption, where native scheme protection is missing.

Protection with cryptography is very effective, potentially robust, and potentially fits everywhere. However, it has many prerequisite requirements from the components, that are not always easy to fulfill. Extending the protection using encryption within the device may require encryption and decryption capabilities, key exchange capabilities, secure storage capabilities, and secure provisioning.

What we do in practice is examine the entire data path and determine what needs to be protected. Then, we check what capabilities each component has. We attempt to build some secure solution using the existent capabilities, and if this is impossible, we add our own IP that will enable the extension to be properly secured.

The functional requirements of the extension may vary in accordance to several parameters, such as the need to protect against bus-level Man-In-The-Middle (MITM) attacks or against replacement of physical components.

5 Summary

Designing a content protection product is not easy. There are several design issues that must be resolved when designing a content protection system, some of which are never encountered when designing other security products. Some of these design issues were presented here.

The first tough decision to be made is which scheme must be supported. There are many schemes, and one may not have a clue as to which scheme will eventually be the most useful; besides the fact that there will probably be no single one scheme. When the scheme is implemented, several security challenges are met, such as the need to store keys securely, the need to address the situation of a platform that may run malicious code, the need for random number generation, and the need to support renewability. Last but not least, after the scheme to be implemented is chosen, practical usability and integration considerations may require extending the data flow beyond what the scheme naively supports. When such cases are met, additional security measures must be deployed.

This paper presented the above issues and the approach that was taken when designing the content protection products in Discretix. Many other issues were, and will be, raised. Many other solutions were, and will be, deployed. Addressing the complex challenge of content protection is probably a never-ending project.