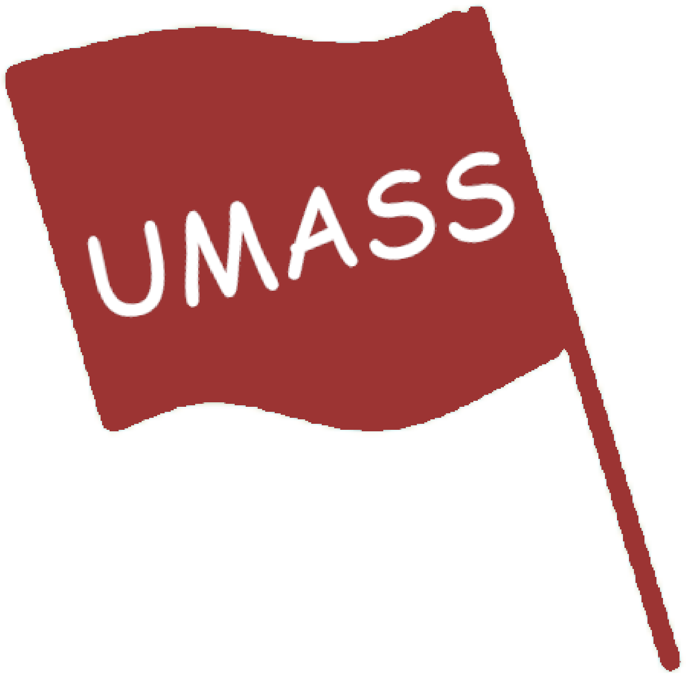
MITRE 2020 Collegiate Embedded Capture-The-Flag



Design Document (v1.0)

University of Massachusetts Amherst
a/k/a
RunDRM

Table of Contents

Fig. 1: Run-DMC showing support for UMass in 2014

Your miPod and You

Congratulations on purchasing your very first RunDRM miPod. This device builds upon the MITRE miPod, so users should familiarize themselves with that design (https://github.com/mitre-cyber-academy/2020-ectf-insecure-example and ~/Downloads/2020_ectf_rules_v1.1.pdf) before reading this document. Here, you can learn all about how the RunDRM miPod has all the same wonderful features as the MITRE miPod while providing super-secure digital rights management.

Security Summary

Since the first miPod audio media player was unveiled in 2001, pesky hackers have exploited the hard work of media producers by illegally downloading songs from the world wide web onto their devices. At long last, RunDRM's state-of-the-art ***open-source*** miPod implementation addresses the numerous security concerns about audio digital rights management. Particularly, we provide confidentiality such that songs can only be played if proper requirements are met, integrity such that songs cannot be modified arbitrarily, and authentication such that certain privileged actions are restricted for unauthorized users. Here we provide some high-level details about the RunDRM miPod's security.

Fig. 2: An early miPod advertisement

Protecting Songs

At provision time, wav-formatted songs are secured for use with our DRM module. For a given input song *SongName.wav*, our provisioning scripts creates two files: *SongName.drm* and *SongName.drm.p*. Each file is digitally signed in blocks to protect against unwarranted modifications.

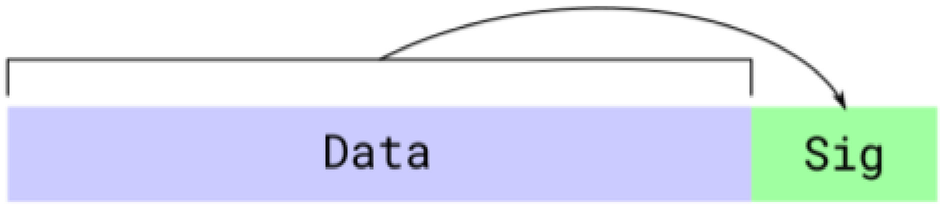


Fig. 3: computing and storing digital signatures

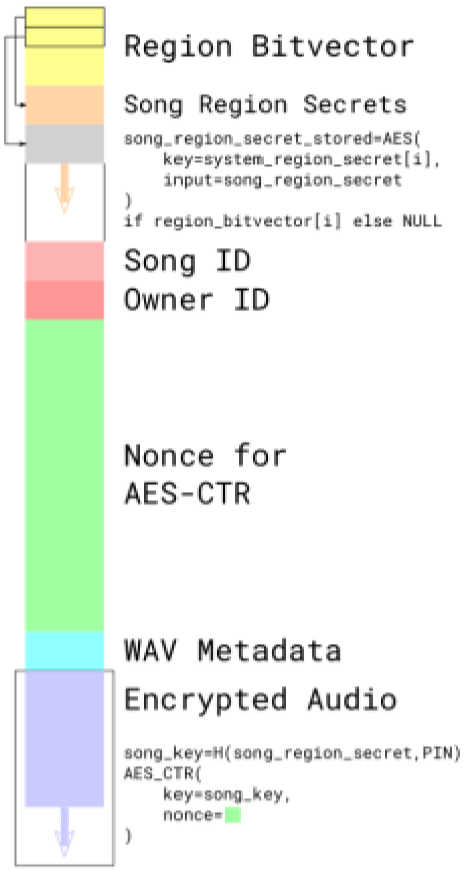
SongName.drm.p is the song preview, as indicated by a *.drm.p* file extension. These files contain the first 30 seconds of original *.wav* audio (encrypted with AES-CTR), plus some other useful metadata.

SongName.drm is the entire song, as indicated by a *.drm* file extension. There is much more going on in these files. Each file contains a bit-vector which represents the regions that song is provisioned for. These bit-vectors are followed by AES-encrypted “region secrets” which are used to derive the key for the respective song’s audio data. Each file also contains the full original audio data (encrypted with AES-CTR), plus some other useful metadata.

Song integrity is provided by using block signatures: the file is divided into 1 kilobyte blocks, and each block is signed using an ECC key when the song is provisioned.

Properly sharing songs generates a third file-type, *.drm.s*. These files contain a bit-vector representing which users the respective song is shared with, an AES key (which is itself encrypted asymmetrically) to be used to decrypt the respective song’s audio data, plus some other useful metadata. MACs are appended to the sharing files to provide integrity.

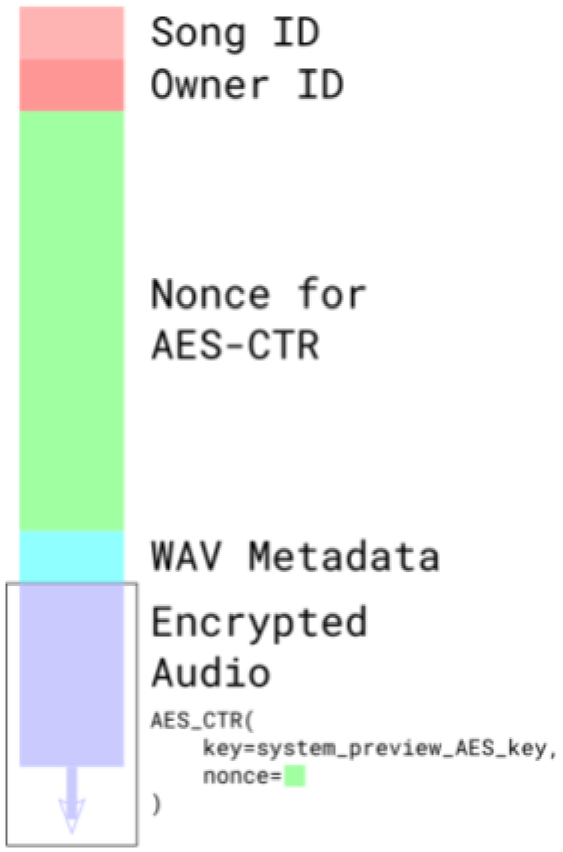
Figures 4a-4c show the structure of these three file types:



```
song_region_secret_stored=AES(
    key=system_region_secret[i],
    input=song_region_secret
)
if region_bitvector[i] else NULL
```

```
song_key=H(song_region_secret,PIN)
AES_CTR(
    key=song_key,
    nonce=🟩
)
```

Fig. 4a: the .drm file structure



```
AES_CTR(
    key=system_preview_AES_key,
    nonce=🟩
)
```

Fig. 4b: the .drm.p file structure

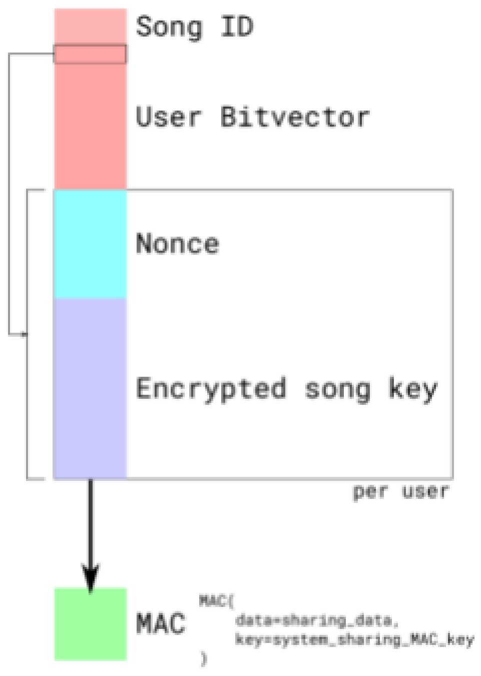


Fig. 4c: the .drm.s file structure

Region Locking

Both songs and miPods are locked to a number of specific regions. If some song is not provisioned for at least one of the regions the miPod is locked to, that song is not allowed to be played. Each region has a specific, symmetric region-key associated with it. Each *.drm* file has some random data, called a region-secret, prepended to it which is used along with the owner's PIN to derive the song's audio data key. Those region-secrets are encrypted with region-keys such that users can only obtain the region-secret if the miPod is loaded with a proper region-key.

User Accounts

Each miPod supports up to 64 different user accounts, which are secured with PINs. Usernames can be 1-15 characters (letters a-z,A-Z) long, and PINs can be 8-64 characters (digits 0-9) long. Those PINs are securely stored as Argon2id hashes so that login credentials can be validated at run time. Users can play, output to a file, and share songs which they own.

Supporting Software

The miPod wouldn't be much without the help of some supporting software. Here we explain these aspects of the design.

Provisioning Scripts

The provisioning scripts are slightly modified versions of the reference design scripts. Importantly, for each input *.wav* song we set up the file formats as described in Figures 4a-4c above. The main modification to the provisioning flow is the addition of board secrets. The `createDevice` scripts creates a system_secrets.json file.

Petalinux Application

An application running on the miPod's ARM processor is used to take user input and drive our DRM module, which resides in the FPGA. This application functions by taking user commands and sending them to a shared "command-channel" in memory for the DRM module to read. Though most operations (particularly ones which require trust) are handled by the DRM module, some unsensitive work is handled here such as servicing any `query` commands. Another important modification from the reference petalinux application is that it now waits a sufficient amount of time before sending a command, since the DRM module has been modified to ignore interrupts while it's working.

DRM Software & Hardware

To provide as much security as possible, the RunDRM miPod implements some hardware- including a Microblaze soft-processor and an AES co-processor. Specifics are discussed here.

Microblaze Application

The Microblaze core handles the miPod's control logic as well as most of the cryptographic operations (with the exception of decrypting the audio itself). Four specific functions of the miPod are particularly secure: logging in (command: `login`), playing songs (`play`), sharing songs (`share`), and un-securing songs (`digital_out`). Libhydrogen (https://libhydrogen.org), an open-source, lightweight crypto library, provides the system's cryptographic primitives. Here are some details:

`Login:` Users are authenticated by computing Argon2id hashes over the input PIN and comparing that to the user's valid hash (which is computed at provisioning time and, or course, stored in a Top Secret location).

`Play:` When attempting to play a song, the Microblaze application first checks whether this song is owned by or shared with the present user. If not, the preview file (*.drm.p*) is checked for integrity, then sent over to the AES co-processor to be played as expected. Otherwise, the next step is obtaining the full song's AES key. This can either be directly derived via a function over the song's region secret and the owner's PIN, or be extracted from its encrypted format in the respective *.drm.s* file. The region secret can only be obtained by decrypting it with a proper system-region-key, which are embedded at provision time. That key is sent to the AES co-processor. At this point, integrity checks are run over the entire song one block at a time due to memory constraints. Finally, the blocks are verified (again) and then sent to the AES co-processor one at a time, in order.

`Share:` Sharing works by securing that song's AES key with asymmetric cryptography such that only shared-with users can access it. The song's key is derived and encrypted with the shared-with user's public key. The shared-with user can then use their private key to obtain the unencrypted song key for playing. This process is shown in Figure 5. These keys are stored persistently in per-song *.drm.s* files on the SD card. The format of these files is shown in Figure 4c.

`Digital Out:` The miPod allows authorized users to "un-secure" songs. This means taking our *.drm* file format and converting it to a plain, old-fashioned *.wav* file. The process is similar to that of playing a song, except the unencrypted song data is redirected to a file.

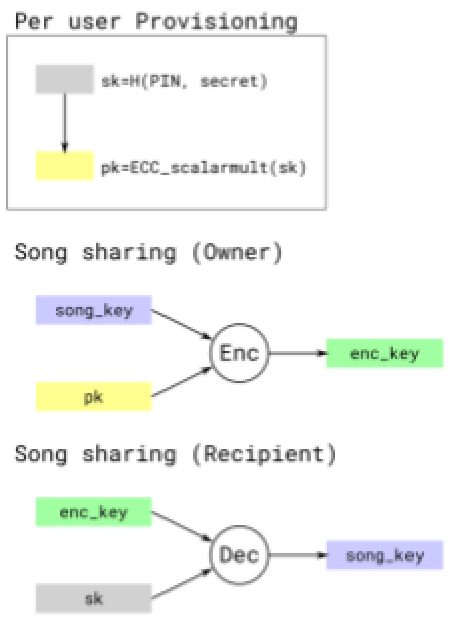


Fig. 5: song sharing scheme

Microblaze Protections

There are two Microblaze cores: one running in MASTER mode and one running in HIDDEN_SLAVE mode. They execute the instructions in parallel and produce a lockstep output that can be externally compared. If a discrepancy is detected, the lockstep comparator signals a reset to the reset manager to prevent faults from being exploited. The Microblaze core also has stack protection exceptions enabled to mitigate the impact of vulnerabilities in the Microblaze code.

AES Co-Processor

We have rolled an AES co-processor to quickly decrypt secured songs so that they can be played. The core is configured using AXI-Lite, and the encryption data itself is sent over AXI Stream. The register space for AXI-Lite configuration is shown in the following table:

Base Address	Description	External read/write
`0x00-0x1f`	Key	`-w`
`0x20-0x2f`	IV/Ctr	`rw`
`0x30-0x33`	Status	`r-`
`0x34-0x37`	Control	`-w`

The status register and control register are both laid out as follows:

`0x00`	`0x01 AXIS_TDEST`	`0x02 ctr_ready`	`0x03 key_ready`

AXIS_TDEST is set to 0x00 for analog output and 0x80 for digital output, and an AXI-Stream interconnect routes the output of the AES core either to the audio output or to a S2MM AXI DMA module for digital export.