inDRM: copy control with a personal touch

Miguel Á. Lechón*

April 1, 2015

You say "I'll just make a copy for me and a friend", but he'll make one and she'll make one and when will it end?

Don't copy that floppy
MC Double Def DP[1]

Abstract

inDRM is a digital rights management scheme. The goal of inDRM is ensuring that a small amount of human reflection accompanies the process of creating and distributing each new copy of a given piece of software. It is particularly well suited for developers that expect lots of passion but little money from their users, such as independent game developers.

Noteworthy properties of inDRM are its:

- *Broken by Design* design.
- P2P activation key generation.
- Distribution history record keeping.

Reference implementations and usage examples are available here[1].

*e-mail: miguel.lechon+inDRM@gmail.com

[1] https://github.com/debiatan/inDRM

1 Motivation

1.1 A brief history lesson

Home taping killed music thirty years ago due to music being distributed as passive, readily cloneable data. Too little was done to remedy this, and too late[2]. Computer software, in contrast, and by its own nature, needs to be active to fulfill its intended purpose, so, through the use of DRM, video games can choose to be as passive[3], aggressive[4] or passive-aggressive[5] as they so desire. This is the main reason why we have games, but not music, today.

1.2 Danger lies ahead

The current dominant threat to the computer game industry is not so much economical as it is ecological. Shastri, Morrison and White provide a concise explanation in their recent discussion[6]:

> It's a total SNAFU! Adults buy all kinds of games but never play them. Kids have time to play but behave like monomaniacs... *Minecraft* this or *LoL* that, day in, day out!

> Soon we will all realize there are no *gamers* left. And when the bubble bursts, the reeking corpses of unplayed *Steam* libraries will wash this industry away.

Indeed, according to undisclosed sources at *Valve*, the average library *backlog* (bought but unplayed games) amounts to 83% of the total number of titles each user owns[7].

1.3 Danger knocks at the door

The vast majority of game developers do not see financial return as an end, but as a way of measuring the engagement of players to their games. However, in today's prosperous economical climate, money is essentially thrown at the feet of game developers for no apparent reason, tricking them into thinking they are succeeding when, in all likelihood, no one is playing their creations.

A recent example is the case of the *Handmade Hero* project[8], where one programmer with no previous history of ever completing a commercial video game announced his plans of working five hours *per week* on a yet-to-be-started title. He successfully collected thousands of preorders during the course of the following weeks.

Will this summer last much longer? Won't some brilliant academic mind devise a clever way of avoiding this impending disaster? How can cautious game developers gauge the *real* commitment of their user base when all signs indicate they are doing a splendid job?

The answers to these questions are *no*, *yes* and *read on*.

1.4 Fear no more

The solution, of course, lies in the repurposing of the industry's proven old savior, DRM.

As long as money keeps growing on the trees of developer's backyards, the amount of game copies sold will be an inaccurate measure of consumer acceptance. Instead, I propose making players spend an ultimately much more valuable resource than money: their time. In order to achieve this, and as a condition to unlock the game, the user will be forced to engage in a short social exchange with either the creator of the game or some other fellow player.

By slightly inconveniencing potential clients, creators can expect to gain a clearer understanding of the appeal of their games and end up with a much more committed, albeit smaller, user base.

2 Cryptological interlude

`inDRM`'s strength rests on:

- the solid foundation of the MD5/4 cryptographic hash function
- the RSA public key cryptosystem signing procedure [10] with a key length of 32 bits
- the identification of computers via the MAC addresses of their primary network interfaces.

This section consists of a rather in-depth review of the first two algorithms.

2.1 MD5/4

MD5[9] takes an arbitrary string of characters, passes it through a deterministic blender and produces an unnecessarily long 128-bit value.

MD5/4 takes that value, chops it into four 32-bit pieces and XORs them together to produce a more digestible 32-bit digest.

2.2 RSA signatures

The RSA signature procedure is a popular party trick from the late 1970s, where one person comes up with three numbers `n`, `d` and `e` that satisfy the following property:

$(x^d)^e \equiv x \pmod{n}, \forall x \in \mathbb{Z}$ [2]

That person then makes `n` and `e` public, spends a few minutes teaching modular exponentiation to the crowd and claims to possess the ability to:

- Turn any message into a number `x` (using a scheme similar to MD5/4)
- Produce a number `y` that depends on `x` and that acts as that person's signature of the message, fact which can be verified by checking that: $y^e \equiv x \pmod{n}$

Of course, behind the scenes, the entertainer obtains `y` by raising `x` to the `d`th power, dividing the result by `n` and taking `y` to be the remainder of that division.

The process of finding the three initial numbers is demonstrated in [11] and can be easily accomplished today with the help from the time-tested software package `openssl`[12] by issuing the following command in any POSIX-compliant operating system:

`openssl genrsa 32|openssl rsa -text`[3]

Looking at its output, the `modulus` field is equivalent to our `n`, `publicExponent` denotes `e` and `privateExponent` indicates `d`.

3 Guided tour

Any person in possession of an unlocked instance of a game protected by `inDRM` can unlock copies for other potential players, as long as those copies descend, directly or indirectly, from that same unlocked instance.

There are several distinct phases in the distribution of a game protected by `inDRM`:

- Author `A` of a game generates a *root activation key file* and distributes it along with the game
- Potential player `P` obtains the game, tries to run it and is invited by the software to generate a *request file* and send it back to author `A` as a precondition for playing
- Author `A` receives the *request file*, throws a small party and generates a *response file* for player `P` that becomes a valid *activation key file*
- Player `P` plays the game, finds it worthwhile and hands a copy to friend `F`

[2] Which translates into Pythonist parlance as:
`(pow(pow(x,d,n),e,n) == x) == True`
for any integer value of `x`

[3] It is fashionable to use 2048 instead of 32, probably in reference to the homonymous video game[13]

- Friend **F** tries to run the game and is invited by the software to generate a *request file* and send it back to either **A** or **P** as a precondition for playing. **F** chooses to contact **P**
- **P** generates a *response file* for friend **F**, by virtue of possessing a valid *activation key file* belonging to the same *key chain* that reached **F**
- ...

The rest of this section reviews the technical details behind the generation of key files.

3.1 Root activation key file

The creator of the game generates a root activation key file and distributes it with every copy of the game. In and of itself, that key file only allows the game to be played on the developer's computer. However, it provides the basis for the generation of key requests from potential players.

Here is an example of an `inDRM` root activation key file:

```
========== inDRM key file ===========
game: Adventures in inDRMland
=====================================
nick: debiatan
location: Barcelona
date: 2015/04/01
notes: Enjoy!
mac_salt: 1e6c40ea
mac_hash: 76f0da12
hash: c738b172
signature: d38cc9a
```

Inside key files, lines starting with an equal sign are ignored. The rest are composed of a tag, followed by a semicolon and a value field. The tags and their associated functions are these:

game: Title of the game

nick: Author's name or nickname

location: Author's place of residence

date: Date of creation of the key file (in *yyyy/mm/dd* format)

notes: Notes from the author

mac_salt: Random 32-bit hex number

mac_hash: MD5/4 hash of the concatenation of **mac_salt** and the hexadecimal representation of the MAC address of the primary network interface of the computer generating the key file

hash: MD5/4 hash of the concatenation of: the preceding **signature** in the key chain (assumed to be "0" in case of the master key file), **game**, **nick**, **location**, **date**, **notes**, **mac_salt** and **mac_hash**

signature: RSA signature of **hash** (computed as $(\text{hash}^d \mod n)$, where d and n have been generated along with e as discussed in section 2.2).

For the particular example used in this section, the values of the cryptographic parameters are:

- n (modulus): 3333098473
- e (public exponent): 65537
- d (private exponent): 939472245

These values are to be embedded in the `inDRM` routines present in the game. The pair (e, n) will be used as an RSA public key in order to check signatures present inside key files. The pair (d, n) will allow a registered copy of the game to sign key file requests from new potential players through an in-game menu option labeled to that effect.

3.2 Activation key file validation

Every time a piece of software guarded by `inDRM` is run it reads the activation key file and checks that:

- the value of the **`hash`** field is correct (by recomputing it using the tag values that precede it)
- the value of the **`signature`** field is correct (by ensuring that $\mathbf{signature}^e \mod n = \mathbf{hash}$)

If these two conditions are met, the primary MAC address of the machine is checked to see if it satisfies:

$\frac{\text{MD5}}{4}(\mathbf{mac\_salt}_i + \text{MAC}) = \mathbf{mac\_hash}_i$

(where the '+' sign indicates concatenation of strings) for any $\mathbf{mac\_salt}_i/\mathbf{mac\_hash}_i$ pair present in the file. If it does, the game is allowed to run. Otherwise, the user is invited to generate a request file.

3.3 Request file generation

An invitation to generate a request file will essentially convey a message similar to this one:

```
*** Last MAC address on key file does
    not belong to this computer ***
You won't be able to play this game
unless you convince another player to
generate a key for you.
  Let's build a request file...

Please provide the following data
(or press 'enter' to skip):
Name (or nickname): _
Location (place of residence): _
Notes (message to future players): _
```

After providing (or failing to provide) the three pieces of information, `inDRM` collects the date and MAC address of the system, generates the request file and tells the user that:

```
A request file has been generated here:
*** /home/ ... /request.txt ***

In order to finish the registration
process, send that file back to
whoever shared the game with you.
That person will be able to unlock
your copy.

Think twice before sharing this game
with other people. If they ever try
playing it, they might come back
asking you to register their copies.
```

The request file consists of a copy of the original key file distributed with the game with an extra section appended at the end. Imagining that the original key was the one presented back in section 3.1 and the user provided "miguel", "barcelona" and "this

sucks" as values for **name**, **location** and **notes**, respectively, the resulting extra section could look like:

```
nick: miguel
location: barcelona
date: 2015/04/01
notes: this sucks
mac_salt: 95be1f47
mac_hash: 6051a20a
hash: 1f495ce2
signature: NO SIGNATURE YET
```

In order to run the game, the potential player will then send the request file up the distribution chain for it to be signed.

3.4 Response file generation

The example activation chain we have described consists of only the original author of the software and its first user. The request file will then be necessarily sent to the author, who will execute a registered copy of the game and select the option that allows to sign requests. That routine will check that:

- the values of all **hash** fields are correct (by recomputing them using the values of fields preceding it)
- the values of all **signature** fields, except for the last one, are correct (by ensuring that ($\mathbf{signature}^e \mod n = \mathbf{hash}$))
- the primary MAC address of the computer satisfies: $\frac{\text{MD5}}{4}(\mathbf{mac\_salt}_i + \text{MAC}) = \mathbf{mac\_hash}_i$ for some $\mathbf{mac\_salt}_i/\mathbf{mac\_hash}_i$ pair in the request file.

If all three conditions are met, the last **signature** field of the request file will be completed with the help of the private exponent **d** and the modulus **n**:

$$\mathbf{signature} = \mathbf{hash}^d \mod n$$

In our example, this would mean modifying the last **signature** of the request file to read:

```
signature: 9bc727d0
```

The resulting file will then be sent back to the potential player to be used in place of the original activation key file. The *potential player* will then stop being just *potential* and start a new life as a *real player*.

4 Properties

Improving security typically means degrading usability. Providentially, both effects are intended consequences of the use of `inDRM`.

There are several other desired properties that have gone into the design of the `inDRM` copy control scheme. This section discusses them briefly.

4.1 *Broken by Design* design

If we were to depict the concept of security as a one-dimensional horizontal segment delimited on the left by the word *insecure* and on the right by the word *secure*, `inDRM` would lie definitely to the left, in a place probably labeled as *cryptographically annoying*. Sadly, space limitations preclude even a cursory exploration of all the security flaws exhibited by `inDRM`.

4.2 P2P key generation

Players of games protected by `inDRM` can rest assured that they will be able to enjoy them for as long as they are interested in doing so. The absence of network authentication and the distributed nature of the key generation process guarantee that.

Moreover, as a side benefit of the distributed nature of the key generation scheme, players might derive some sense of belonging to a community by simple inspection of the contents of their personal key files. Intensely imaginative players, of the kind that take pleasure in trading *Steam* inventory items, may even experience `inDRM`'s compulsory chaining of signatures as a tradition that makes them part of a lineage.

4.3 Lack of attractive as a cracking target

All DRM schemes are eventually subverted, so one should not ask *if* but *when* will `inDRM` be broken. I suspect that it will not happen any time soon for mainly two reasons:

- Breaking `inDRM` poses no challenge, so no one can take pride on that endeavor
- The only benefit to be derived from breaking `inDRM` is the avoidance of a short social interaction

4.4 Distribution history record keeping

Key files have potential uses other than game distribution control. For instance, the combined **`notes`** sections of a key can be used as a simple guestbook or, perish the thought, a space for advertisements.

Another more elaborate use requires us to picture the set of activation key files associated to a particular game as a tree, its *leaf* key files being the ones that carry the most information. Getting hold of a sizeable percentage of those files by some means[4], would facilitate the identification of the principal hubs in the game's distribution network.

4.5 Simplicity

MD5 is straightforward to program[14]. The improvements that MD5/4 introduces come at the cost of three extra XOR instructions.

Implementing the standard RSA algorithm would require working with arbitrarily long integers, but the 32-bit version of it does not impose that hardship on game developers. Two additional benefits of working only with 32-bit values are that key files become smaller in size and that they fit inside the two-column layout of this article.

4.6 Involvement of players

In order to acquire a valid activation key file, the potential player must secure the collaboration of another player. After implicating that other person, the new player may feel pressured to give the game a proper try and avoid dropping it after just a few minutes of play.

[4] such as asking nicely or offering players extra game content in exchange for their keys

4.7 Future-proof architecture

As long as one registered copy of the game remains, it will continue to be redistributable and playable. Even if no registered copy remains and even if the game binary proves difficult to crack, bruteforcing the **`signature`** of a request file requires just the testing of the set of 2^{32} possible signatures.

5 Conclusion and final remarks

Game developers do not need protection from software pirates; they need instead to be guarded from the uncaring capricious money-throwing player hordes that endanger their profession. By choosing to distribute games guarded by `inDRM`, they are not only deciding to acquire a much more mature following, but are also telling the video game community that *they care*.

For an up-to-date list of games that use `inDRM`, check:

`http://blog.debiatan.net/inDRM.html`

References

[1] MC Double Def Disk Protector, *Don't copy that floppy*[5], 1992

[2] *Sony BMG copy protection rootkit scandal*[6], Wikipedia

[3] *Good Old Games*[7]

[4] *Always-on DRM*[8], Wikipedia

[5] *DRM – Software tampering*[9], Wikipedia

[6] S. Shastri, R. T. Morrison, and X. White, *More games, less time.* Journal of Bad Omens, vol. 35, pp. 75-80, Dec. 2012.

[7] "Undisclosed" means "undisclosed".

[8] *Handmade Hero*[10]

[9] R. Rivest, *The MD5 message-digest algorithm*, RFC 1321, April 1992.

[10] R. Rivest, A. Shamir an L. Adleman, *A Method for Obtaining Digital Signatures and Public-Key Cryptosystems*, Communications of the ACM 21 (2): 120–126. February 1978.

[11] *RSA encryption and decryption, a worked example*[11], Wikipedia

[12] *OpenSSL*[12], Wikipedia

[13] *2048*[13], Wikipedia

[14] *MD5 pseudocode*[14], Wikipedia

[5] `https://www.youtube.com/watch?v=up863eQKGUI`

[6] `https://en.wikipedia.org/wiki/Sony_BMG_copy_protection_rootkit_scandal`

[7] `http://www.gog.com/`

[8] `https://en.wikipedia.org/wiki/Always-on_DRM`

[9] `https://en.wikipedia.org/wiki/Digital_rights_management#Software_tampering`

[10] `https://handmadehero.org/`

[11] `https://en.wikipedia.org/wiki/RSA_(cryptosystem)#A_worked_example`

[12] `https://www.openssl.org/`

[13] `https://en.wikipedia.org/wiki/2048_(video_game)`

[14] `https://en.wikipedia.org/wiki/MD5#Pseudocode`