11 A Binary Magic Trick, Angecryption

by Ange Albertini and Jean-Philippe Aumasson

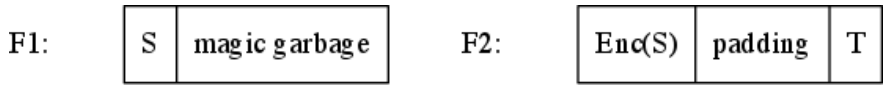
This PDF file, the one that you are reading right now, contains a magic trick. If you encrypt it with AES in CBC mode, it becomes a PNG image! This brief article will teach you how to perform this trick on your own files, combining PDF, JPEG, and PNG files that gracefully saunter across cryptographic boundaries.

Given two arbitrary documents S (source) and T (target), we will create a first file F_1 that gets rendered the same as S and a second file $F_2 = AES_{K,IV}(F_1)$ that gets rendered the same as T by respective format viewers. We'll use the standard AES-128 algorithm in CBC mode, which is proven to be semantically secure[13] when used with a random IV.

In other words, any file encrypted with AES-CBC should look like random garbage, that is, the encryption process should destroy all structure of the original file. Like all good magicians, we will cheat a bit, but I tell you three times that if you encrypt this PDF with an IV of `5B F0 15 E2 04 8C E3 D3 8C 3A 97 E7 8B 79 5B C1` and a key of "`Manul Laphroaig!`", you will get a valid PNG file.

11.1 When the Format Payload can Start at Any Offset

First let's pick a format for the file F_2 that doesn't require its payload to start right at offset 0. Such formats include ZIP, Rar, 7z, etc. The principle is simple:



First we encrypt S, and get apparent garbage $Enc(S)$. Then we create F_2 by appending T to $Enc(S)$, which will be padded, and we decrypt the whole file to get F_1. Thus F_1 is S with apparent garbage appended, and F_2 is T with apparent garbage prepended.

This method will also work for short enough S and formats such as PDF that may begin within a certain limited distance of offset 0, but not at arbitrary distance.

11.2 Formats Starting at Offset 0

We had it easy with formats that allowed some or any amount of garbage at the start of a file. However, most formats mandate that their files being with a magic signature at offset 0. Therefore, to make the first blocks of F_1 and F_2 meaningful both before and after encryption, we need some way to control AES output. Specifically, we will abuse our ability to pick the Initialization Vector (IV) to control exactly what the first block of F_1 encrypts to.

In CBC mode, the first 16-byte ciphertext block C_0 is computed from the first plaintext block P_0 and the 16-byte IV as

$$C_0 = Enc_K(P_0 \oplus IV)$$

where K is the key and Enc is AES. Thus we have $Dec_K(C_0) = P_0 \oplus IV$ and we can solve for

$$IV = Dec_K(C_0) \oplus P_0$$

As a consequence, regardless of the actual key, we can easily choose an IV such that the first 16 bytes of F_1 encrypt to the first 16 bytes of F_2, for any fixed values of those 2×16 bytes. The property is obviously preserved when CBC chaining is used for the subsequent blocks, as the first block remains unchanged.

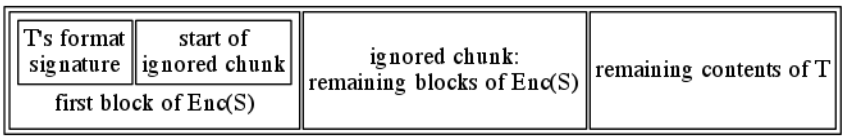
So now we have a direct AES encryption that will let us control the first 16 bytes of F_2.

Now that we control the first block, we're left with a new problem. This trick of choosing the IV to force the encrypted contents of the first block won't work for latter blocks, and they will be garbage beyond our control.

[13] "IND-CPA" in cryptographers' jargon.

So how do we turn this garbage into valid content (that renders as T)? We don't. Instead, we use the contents of the first block to cause the parser to skip over the garbage blocks, until it lands at the ending region which we control. This trick is similar to the one I used to combine a PDF and JPEG in Section 3, and it's a damned important trick to keep handy for other purposes.

F2:



Let's take a look at some specific file formats and how to implement them with Angecryption.

11.2.1 Joint Photographic Experts Group

According to specification,[14] JPEG files start with a signature `FF D8` called "Start Of Image" (SOI) and consist of chunks called segments. Segments are stored as

$$\langle marker : 2\rangle\langle variablesize(data + 2) : 2\rangle\langle data :?\rangle$$

In a typical JPEG file the SOI is followed by the APP0 segment that contains the JFIF signature, with marker `FF E0`. The APP0 segment is usually 16 bytes.

So we need to insert a COMment segment (marker `FF FE`) right after the SOI. As we know the size of S in advance, we can already determine the start of F_2, and then the AES-CBC IV. T will then contain the APP0 segment, and its usual JPEG content.

11.2.2 Portable Network Graphics

PNG files are similar to JPEGs, except that their chunks contain a checksum, and their size structure is four bytes long.

A PNG file starts with the signature "`\x89PNG\x0D\x0A\x1A\x0A`" and is then structured in TLV chunks.

$$\langle length(data) : 4\rangle\langle chunktype : 4\rangle\langle chunkdata :?\rangle\langle crc(chunktype + chunkdata) : 4\rangle$$

These are typically located right after the signature, where an IHDR (ImageHeaDeR) chunk usually starts.

For F_2 to be valid, we need to start with a chunk that will cover the $len(S) - 16$ garbage bytes of $Enc(S)$. We can give it any lowercase chunk type,[15] and luckily, at the end of the chunk type, we're right at the limit of 16 bytes, so no brute forcing of the next encrypted block is required.

At that point of F_2 the uncontrolled garbage portion may start. We then calculate its checksum, append it, then resume with all the chunks coming from T. Our F_2 is now composed of (1) a PNG signature, (2) a single dummy chunk containing $Enc(S)$, and (3) the T chunks that make up the meaningful image. This is a valid PNG file.

11.2.3 Portable Document Format

PDF may include dummy objects of any length. However, we need a trick to make the signature and the first object declaration fit in the first 16 bytes.

A PDF starts with "%PDF-1.5" signature. This signature has to be entirely within the first 1024 bytes of the file, and everything after the signature must be a valid PDF file. Because the uncontrolled portion of the file appears as a lot of garbage after the first block, it needs to be enclosed in a dummy stream object.

[14] JPEG File Interchange Format Version 1.02, Sept. 1, 1992

[15] If the first letter in the type field of a PNG block is lowercase, then that chunk will be ignored by the viewer, which interprets it as a custom dummy block.

```
1 0 obj
<< >>
stream
```

Unfortunately, the PDF signature followed by a standard stream object declaration take up 30 bytes. Choosing the IV only gives us 16 bytes to play with, so we must somehow compress the PDF header and opening of a stream object into slightly more than half the space it would normally take.

Our trick will be to truncate both the signature and the object declaration by inserting null bytes "`%PDF-\0obj\0stream`". The signature is truncated by a null byte,[16] and we also omit the object reference and generation, and the object dictionary. Luckily, this reduced form takes exactly 16 bytes, and still works!

Now the uncontrolled remainder of $Enc(S)$ will be ignored as a valid but unused stream object. We then only need the start of T to close that object, and then T can be a valid PDF. So F_2 is a valid PDF file, showing T's content.

11.3 Conclusion

Provided that the format of our source file tolerates some appended garbage, and that the file itself is not too big, we can encrypt it to a valid PNG, JPEG or PDF.

This same technique can work for other ciphers and file formats. Any block cipher will do, provided that its standard block size is big enough to fit the target header and a dummy chunk start. This means we need six bytes for JPEG, sixteen bytes for PDF and PNG.

An older cipher such as Triple-DES, which has blocks of eight bytes, can still be used to encrypt to JPEG. ThreeFish, which can have a block size of 64 bytes, can even be used to encrypt a PE. The first block would be large enough to fit the entire `DOS_HEADER`, which allows you to relocate the `NT_Header`s wherever you like, up to `0x0FFF_FFFF`.

So you could make a valid WAV file that, when encrypted with AES, gives you a valid PDF. That same file, when encrypted with Triple-DES, gives you a JPEG. Furthermore, when decrypted with ThreeFish, that file would give you a PE. You can also chain stages of encryption, as long as the size requirements are taken care of.

[16] This part of the trick was learned from Tavis Ormandy.