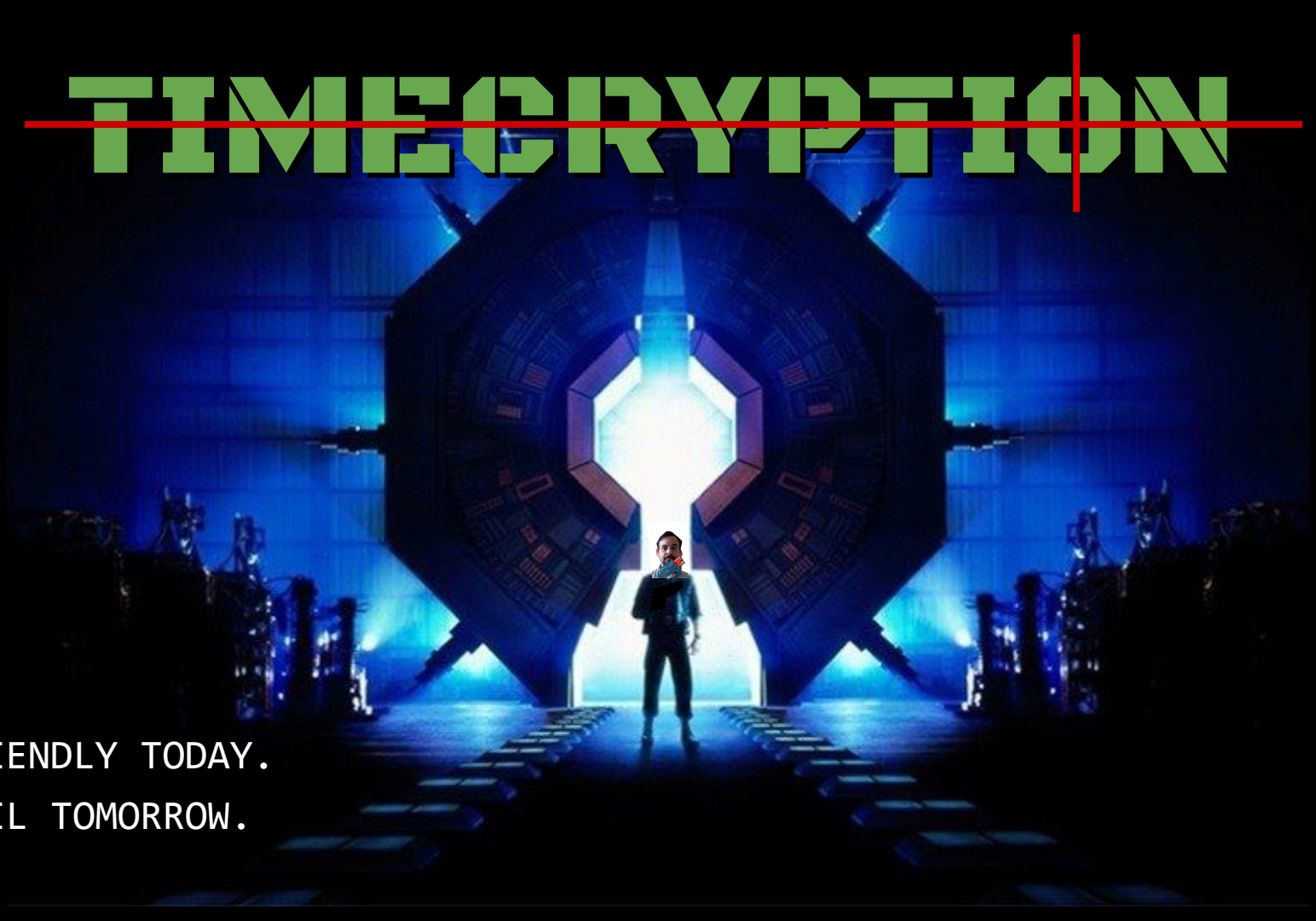
TIMECRYPTION

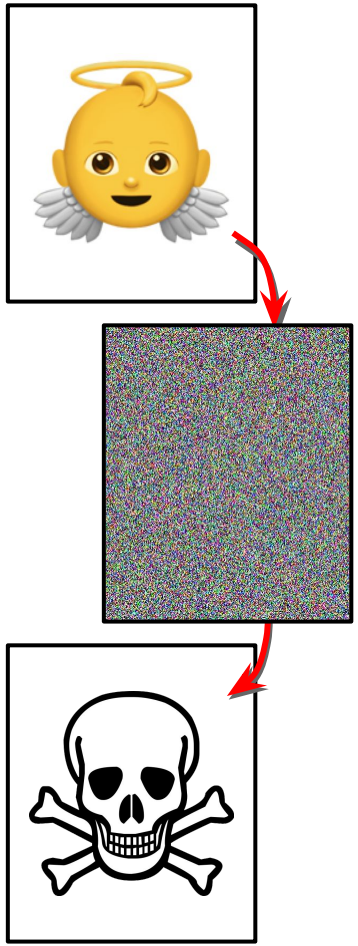


FRIENDLY TODAY.

EVIL TOMORROW.

TIMECRYPTION

CLEAN NOW, MALICIOUS LATER



A talk by


Ange Albertini


Stefan Kölbl

A.K.A.

Abusing one-time pads with binary polyglots.

About Stefan Kölbl

- Doing Cryptography research and engineering for 10 years
- Made several cryptanalysis tools (CryptoSMT, Solvatore)
- Contributed to several crypto designs, e.g. Gimli, Skinny, SPHINCS+

Professionally

- Now Information Security Engineer @ Google
- Worked as Postdoc and Security Consultant
- PhD on symmetric key cryptography

About Ange Albertini

- Reverse engineering since 1989
- Author of Corkami
- 6 years at PoC or GTFO*
- Occasional drawer, singer
- File Formats For Ever

Professionally

- 13 years of malware analysis
- 2 years of Information Security Engineer at Google

My license plate is a CPU.
My phone case is a PDF doc.
My resume is a Super NES/Megadrive rom PDF

*https://github.com/angea/pocorgtfo/blob/master/README.md

Honest trailer

THE CURRENT SLIDE IS AN

HONEST TALK TRAILER

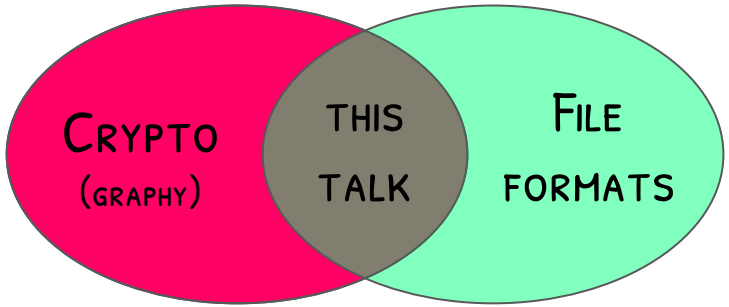
A CORKAMI ORIGINAL PRODUCTION

No new cryptographic finding.

Yet another use of file formats tricks.

GCM mode is standard

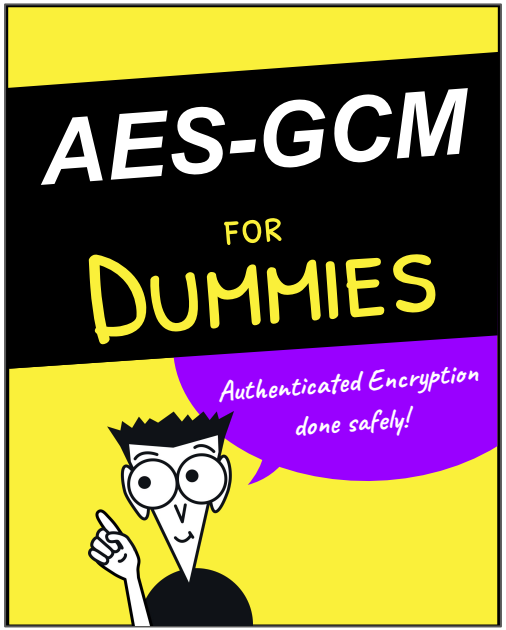
Let's raise awareness!



How to Abuse and Fix Authenticated Encryption Without Key Commitment

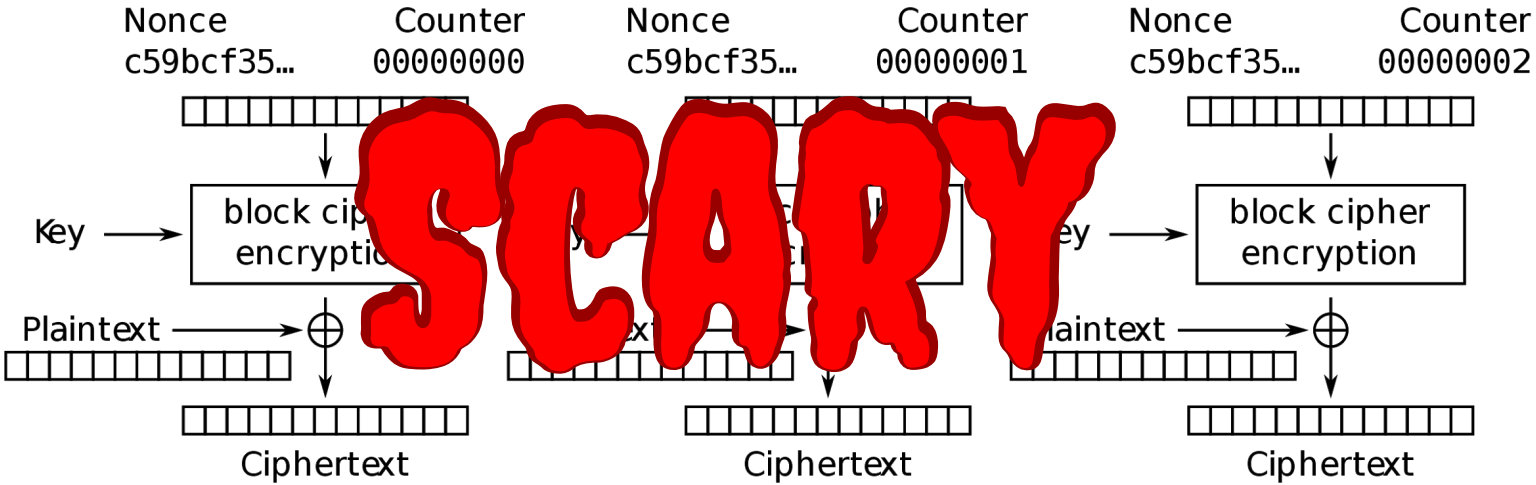
Ange Albertini, Thai Duong, Shay Gueron, Stefan Kölbl, Atul Luykx, Sophie Schmieg

Cryptology ePrint Archive: Report 2020/1456 - last revised 11 Jun 2021



THIS TALK IS A VULGARISATION OF OUR PAPER

https://eprint.iacr.org/2020/1456



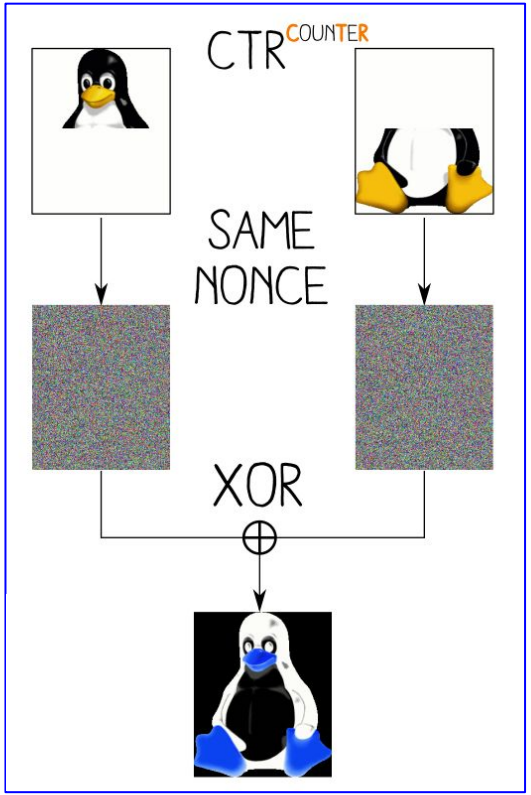
How you may know the CounTeR mode (from wikipedia)...

CTR MODE OF OPERATION

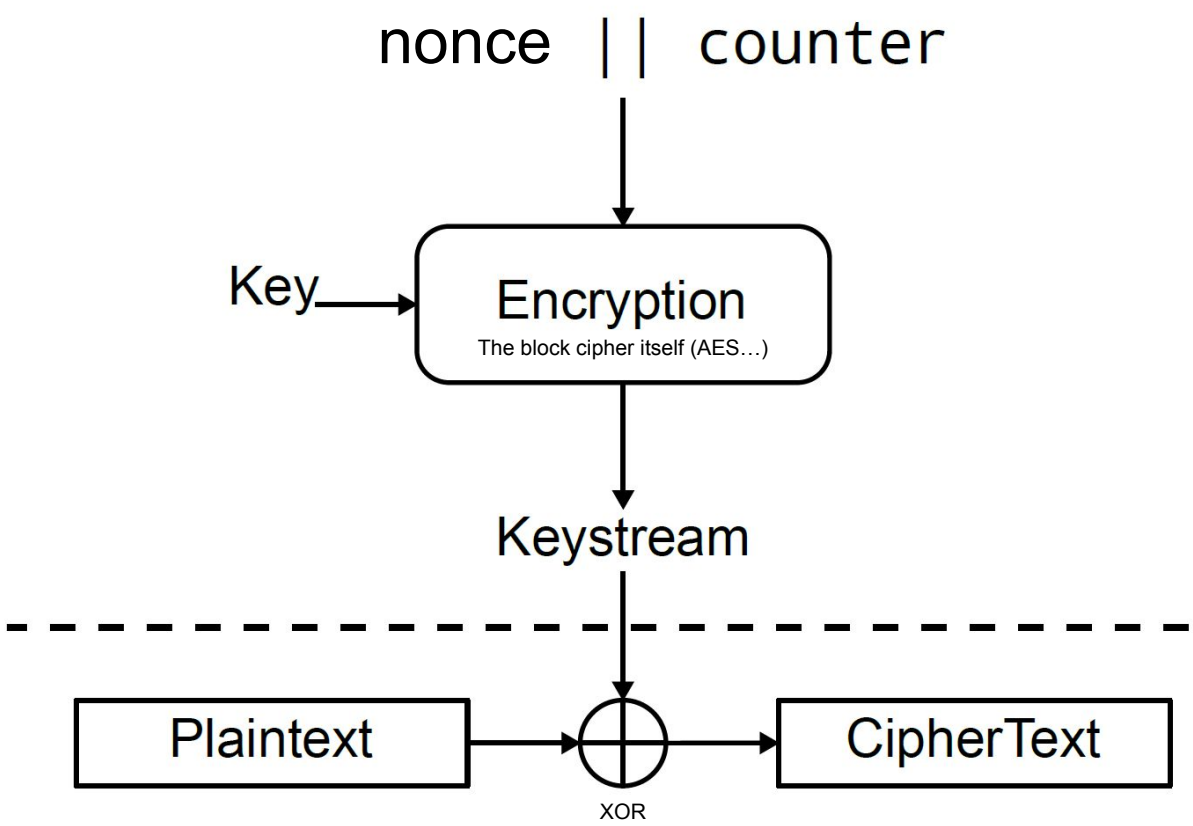
- Needs a "used-once" number (a nonce)
- Generates a keystream from (Nonce, Key)
  with the same length as the plaintext
- Ciphertext = Keystream xor Plaintext

-> CTR acts as a one-time pad

> In cryptography, the **one-time pad** (**OTP**) is an encryption technique that cannot be cracked, but requires the use of a one-time pre-shared key the same size as, or longer than, the message being sent. In this technique, a plaintext is paired with a random secret key (also referred to as *a one-time pad*). Then, each bit or character of the plaintext is encrypted by combining it with the corresponding bit or character from the pad using modular addition.

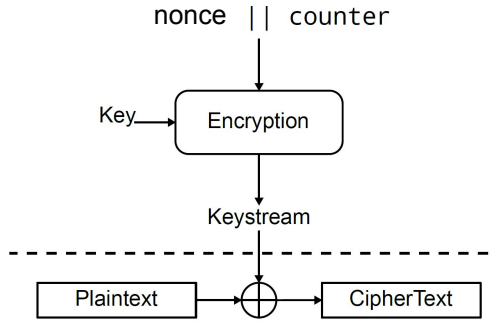


The risks of nonce reuse
(you end up xoring with the same keystream)



High-level view of CTR mode

Remarks



- [Nonce || Counter] are the blocks being encrypted
- The block cipher is used to generate a keystream, independently of the plaintext and ciphertext
  -> parallelizable

CTR mode is a xor with a keystream

CTR decryption and CTR encryption are the same operation

Recap on CounTeR mode

CTR turns a block cipher into a stream cipher

Cipher decryption isn't used

Cipher encryption is just used to generate a keystream

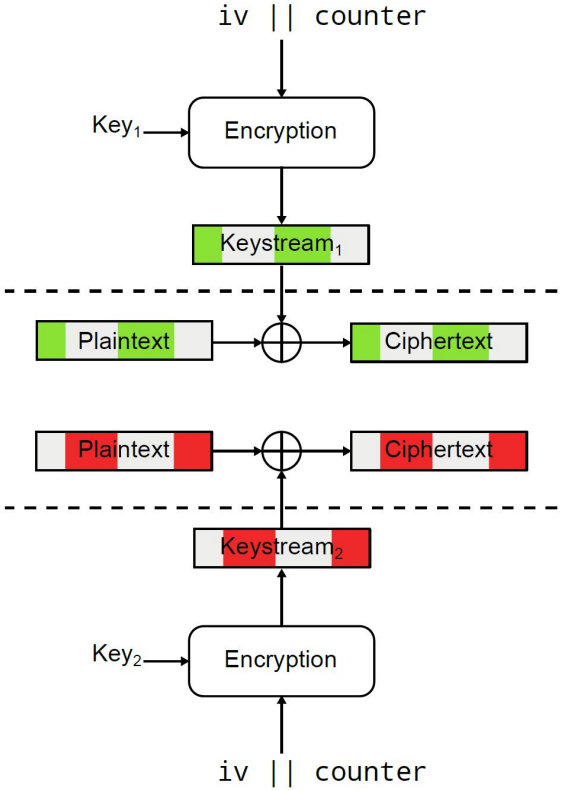
CTR decryption is the same as CTR encryption:
a xor with this keystream

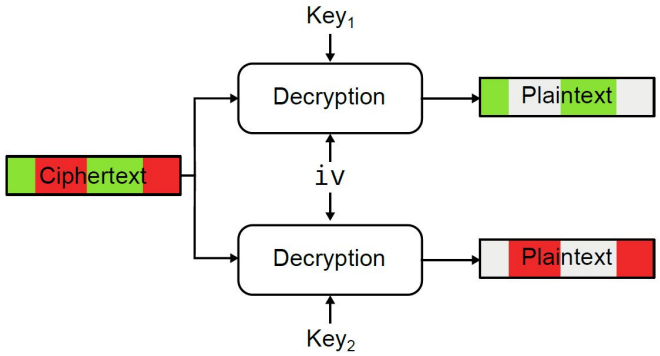
What if we decrypt with a different key?

We just end up xoring
with a different keystream

Craft a ciphertext that gives meaningful plaintexts for different keystreams

An "ambiguous" ciphertext





Make valid contents co-exist by encrypting and slicing

Craft a ciphertext?

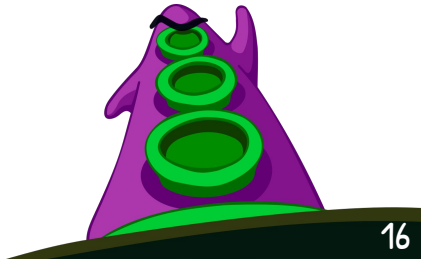
We can freely modify the ciphertext

The keystream is set by (Nonce, Key)
  Plaintext and ciphertext aren't involved

For a given keystream [which is set by (Nonce, Key)]
  if we change ciphertext bytes, we set the plaintext bytes
  [it's just a xor against a known keystream]

Several valid contents in the same file?

Binary polyglots !!!



Mitra: a binary polyglot generator

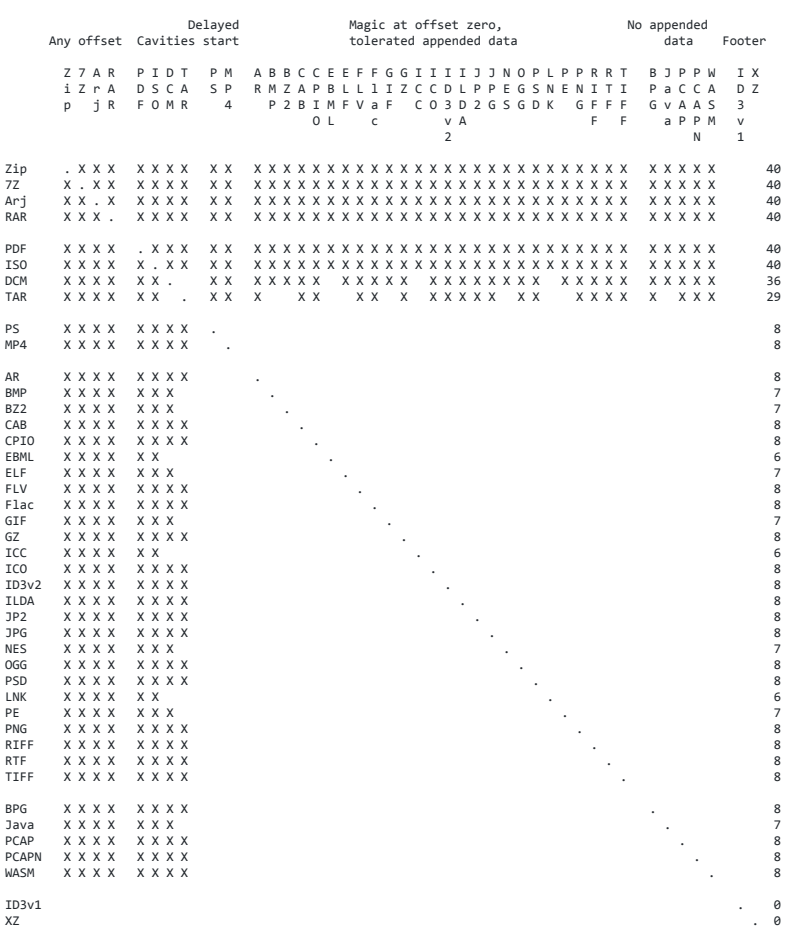
https://github.com/corkami/mitra

Not a parser nor a validator!
Just knows the bare minimum
about each format

- Takes 2 files as input
- Identifies file formats (40+ supported)
- Tries different layouts: concatenation, parasites, cavities, zippers...
- Generates binary polyglots

How does it work? -> 

278 FORMAT COMBINATIONS



Only 278 ?

Several sub-formats per container format

Several layouts per combinations

=> 700+ different kinds of polyglots

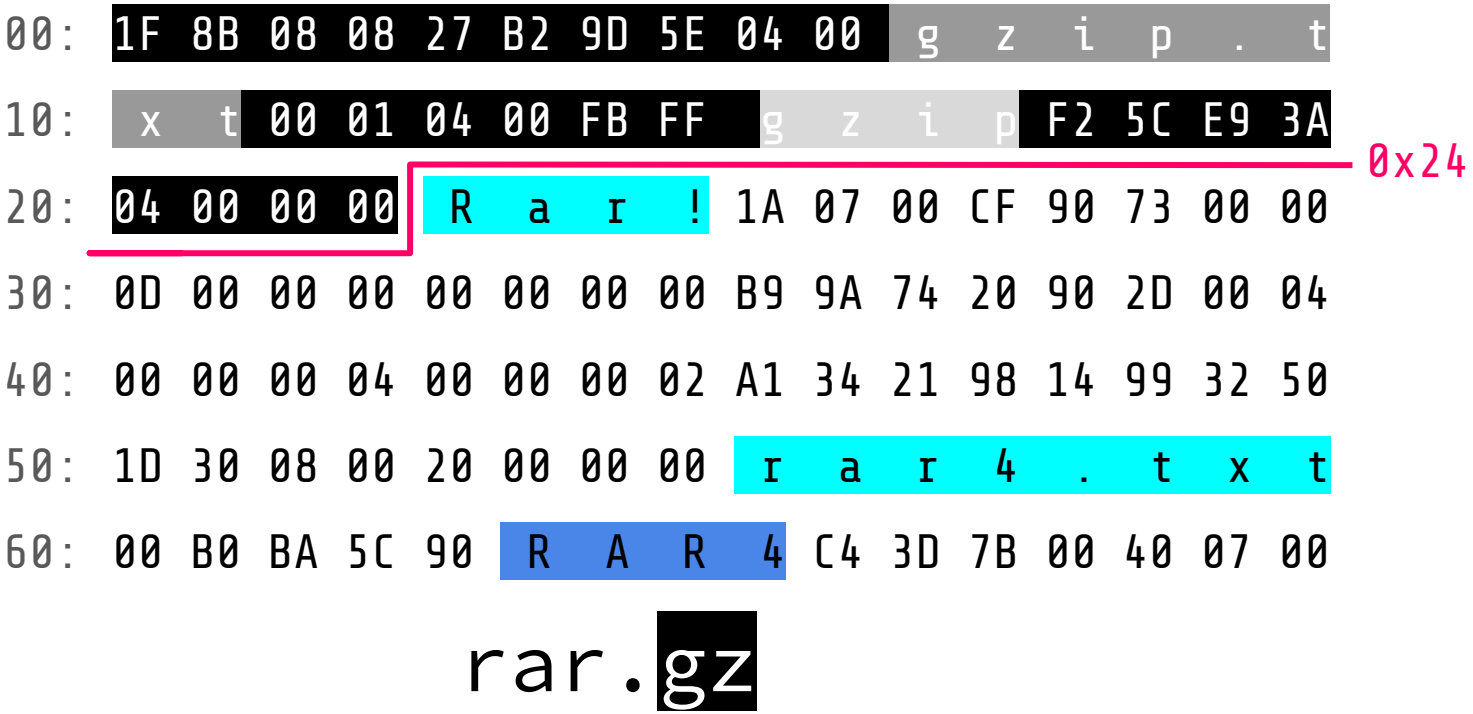
(that's just how much I tried TBH)

Format	Subformats
ZIP	JAR, DOCX, APK...
Riff	Wav, AVI, Ani, SfBk...
ISOBM	MP4, JP2, QT, M4V...
Ogg	Vorbis, Flac, Opus...

Mitra use

Just run the script (Check mini or tiny PoCs for input files)

```
mitra>mitra.py in\gzip.gz in\rar4.rar
in\gzip.gz
File 1: gzip
in\rar4.rar
File 2: RAR / Roshal Archive

Stack: concatenation of File1 (type GZ) and File2 (type RAR)
Parasite: hosting of File2 (type RAR) in File1 (type GZ)

mitra>_
```



Actually named: S(24)-GZ-RAR.a7bccab6.rar.gz

the list of hex offsets where the contents change from one format to the other

A TINY GZIP/RAR POLYGLOT BY CONCATENATION

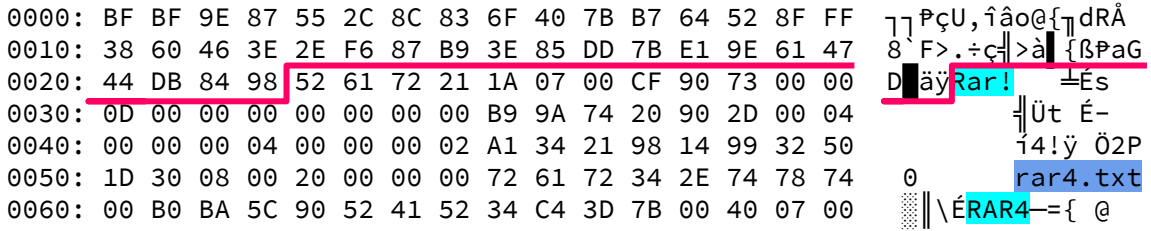
Depending on where you start reading, you'll get different results

mustangstrombonehead**line**feed**back**hand**rail**road**side**show**down**turn**over**book**case**work**shop**

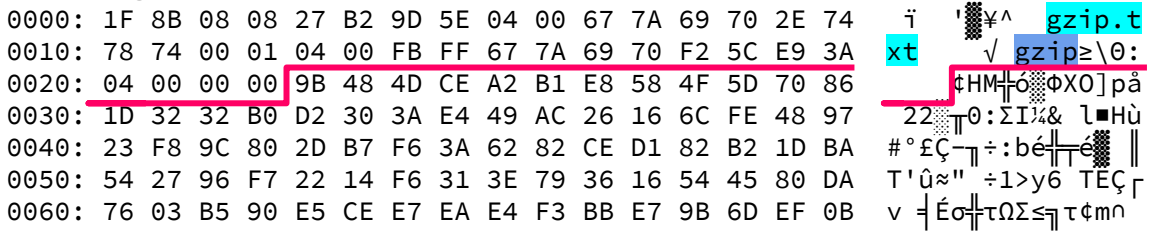
CTR TOOLS FOR MITRA

```
mitra\utils\ctr>brioche.py "S(24)-GZ-RAR.a7bccab6.rar.gz" gzip-rar4.ctr
Generated output: gzip-rar4.ctr
Tests:
 openssl enc -in gzip-rar4.ctr -out output1.rar -aes-128-ctr
    -iv 00000000000000000000000 -K 4e6f773f00000000000000000000000
 openssl enc -in gzip-rar4.ctr -out output2.gz  -aes-128-ctr
    -iv 00000000000000000000000 -K 4c3474337221212100000000000000

mitra\utils\ctr>_
```

output1.rar



output2.gz



2 VALID FILES FROM THE SAME CIPHERTEXT

More than polyglots

- 2 related files coming from the same ciphertext
- 1 format is hidden when the other is in clear
  -> hides malicious payload
  -> bypass polyglot blacklisting (Adobe Reader)

```
$ unrar vl output1.rar

UNRAR 5.61 beta 1 freeware      Copyright (c) 1993-2018 Alexander Roshal

Archive: output1.rar
Details: RAR 4, SFX

 Attributes      Size    Packed Ratio    Date    Time   Checksum  Name
----------- ---------  -------- ----- ---------- -----  --------  ----
    ..A....         4         4 100%  2020-01-18 19:08  982134A1  rar4.txt
----------- ---------  -------- ----- ---------- -----  --------  ----
                    4         4 100%                              1
```

```
$ gzip -t output2.gz

gzip: output2.gz: decompression OK, trailing garbage ignored
```

GCM

Here comes a
new challenger!

You do **NOT** want unauthenticated encryption

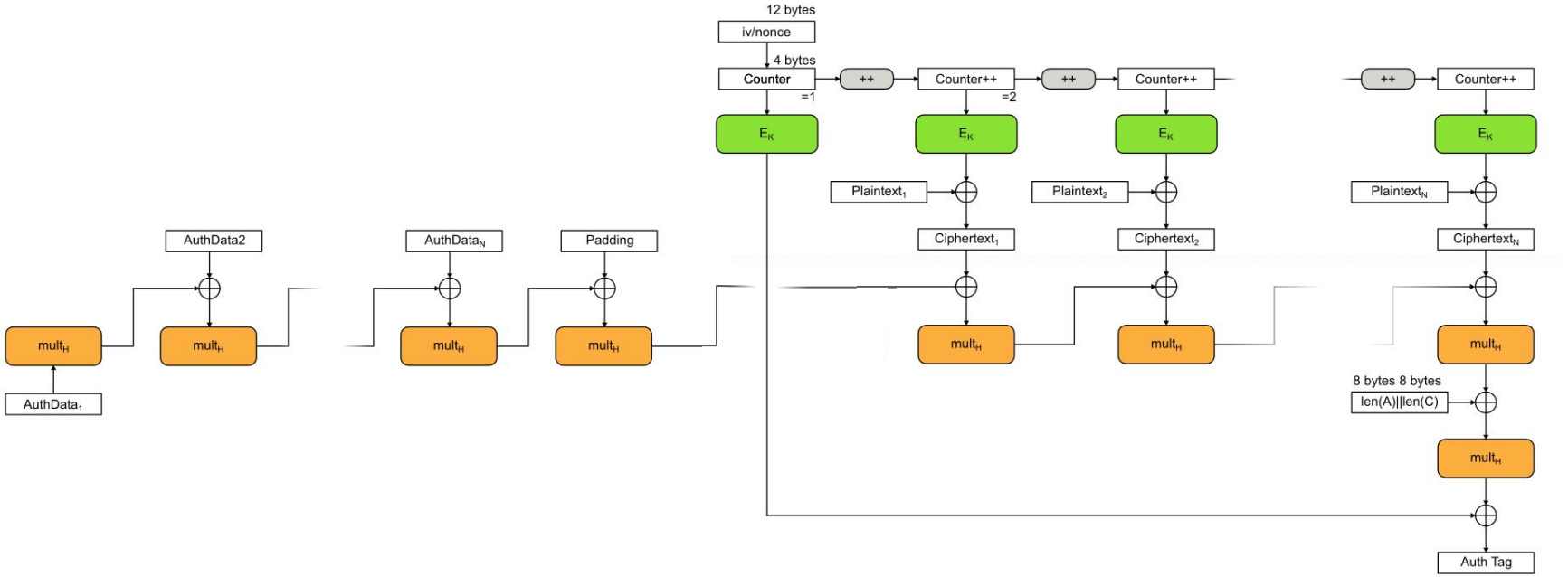
CBC padding oracle attacks anyone ?

https://en.wikipedia.org/wiki/Padding_oracle_attack

https://github.com/google/security-research/security/advisories/GHSA-f5pg-7wfw-84q9

Authenticated Encryption = AE

Authenticated Encryption w/ Associated Data = AEAD

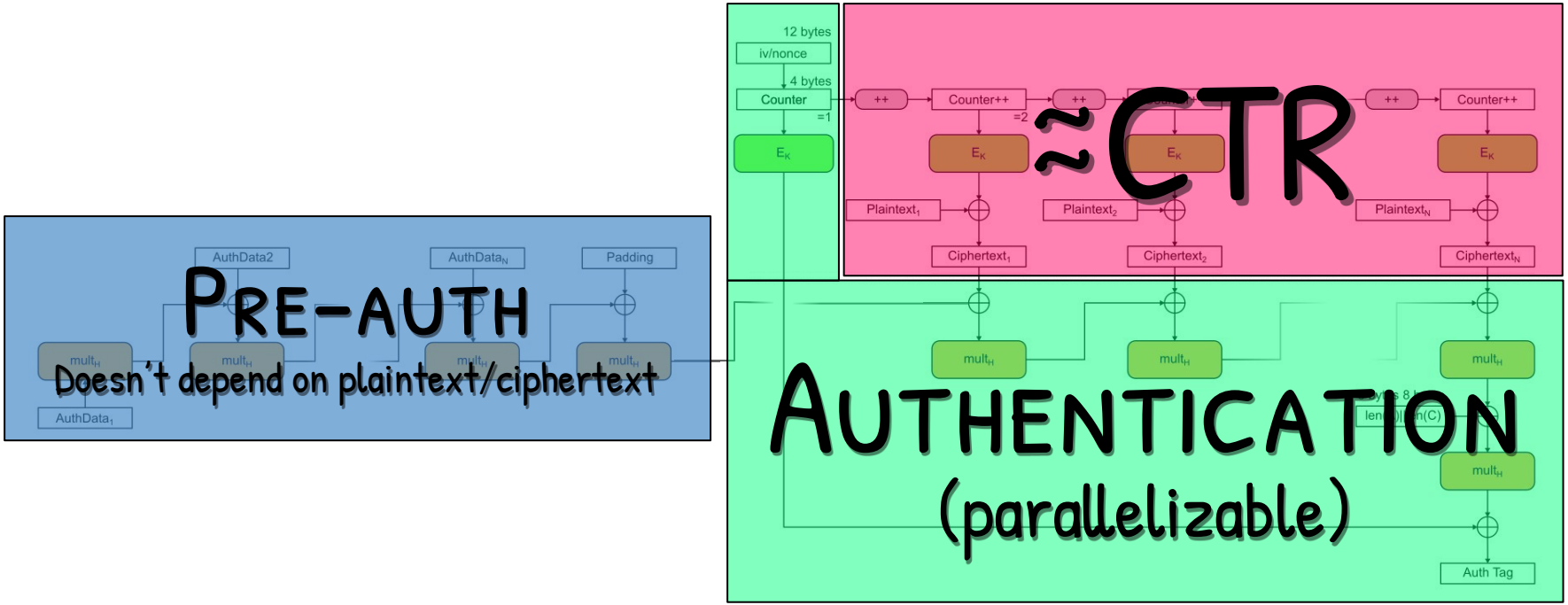


Galois/Counter Mode

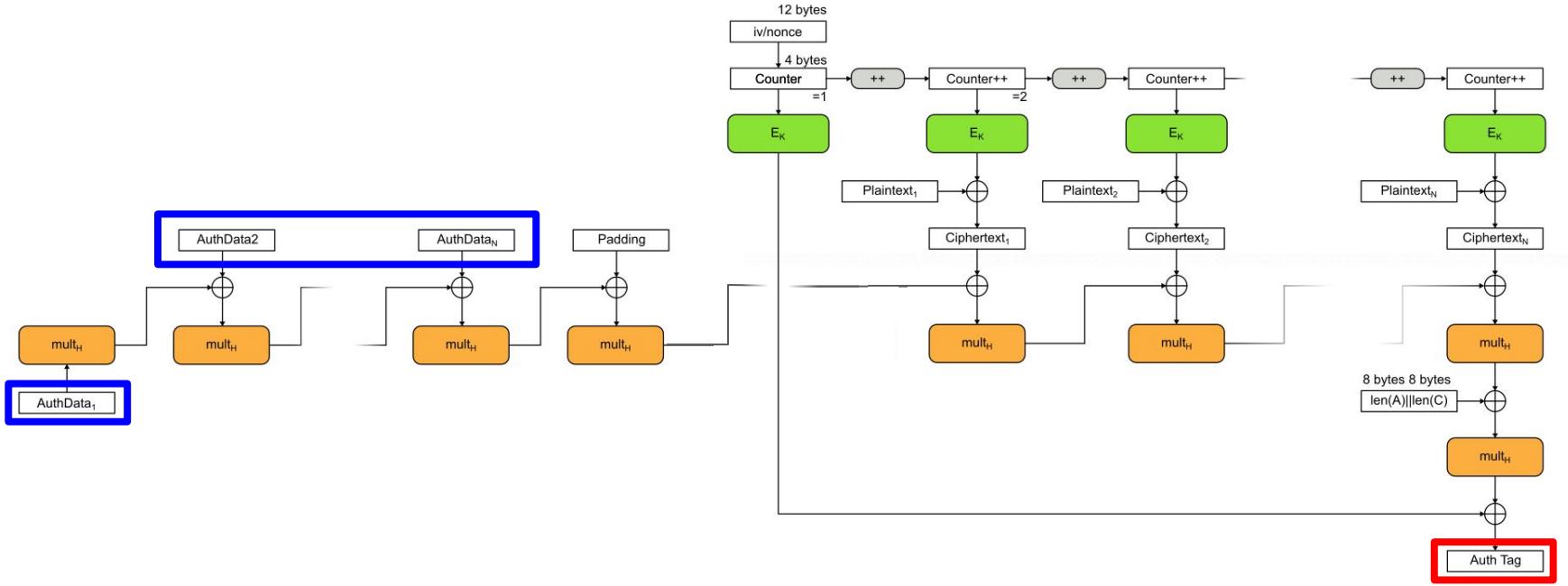
FTR

Modes of operation:

Electronic CodeBook	Cipher Block Chaining
CounTeR	Cipher FeedBack
Galois/Counter Mode	Output FeedBack



It's just CTR with authentication

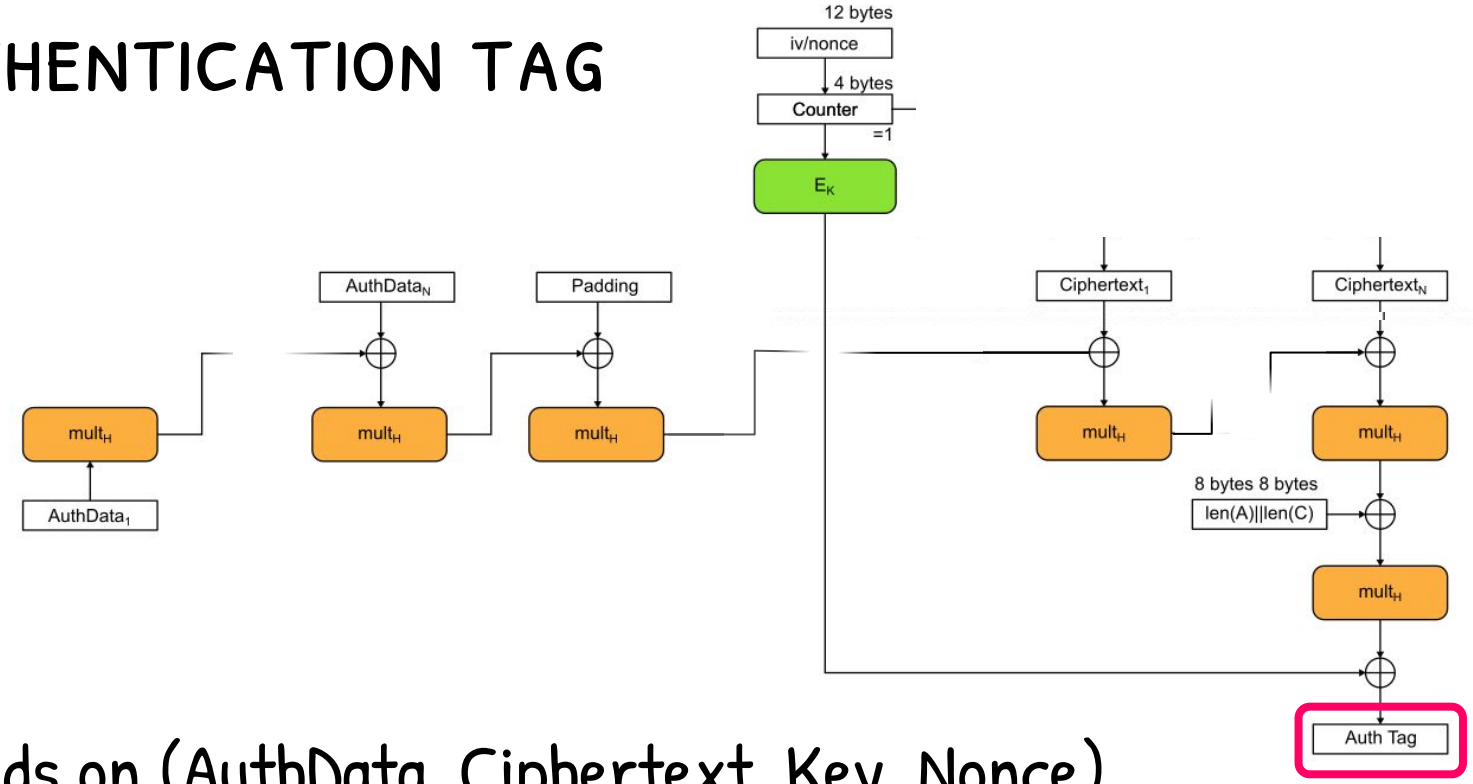


New inputs/outputs

CTR: (plaintext) -> ciphertext

GCM: (Plaintext, AuthData) -> Ciphertext, Tag

Authentication tag



Depends on (AuthData, Ciphertext, Key, Nonce)

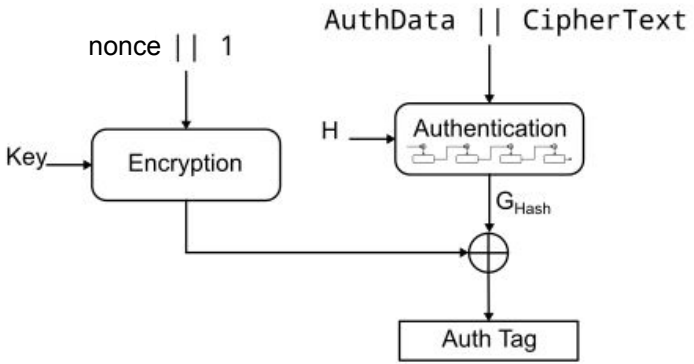
Re-computed after decryption, compared to the stored value

GCM mode

CTR for encryption + G_{hash} for Authentication on Ciphertext

The authentication tag depends on (AuthData, ciphertext, Key, Nonce)

Any modification to any of these will fail the authentication



So we can't use a different key?

WRONG!

We can make it so two decryptions w/ different keys both pass verification

We just need to sacrifice a block and do some computation
(it was known from the beginning - in 2004 - but not seen as a risk)

GCM FAILURES IN THE WILD

Facebook, Google, Amazon...

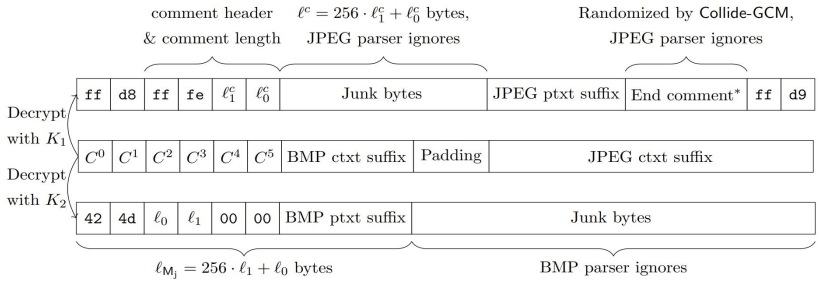
FaceBook messenger

Recipient and abuse team see different images

A BMP <-> JPEG "collision"
(with 6b of overlap)

3 Invisible Salamanders: Breaking Facebook's Franking

In this section we demonstrate an attack against Facebook's message franking. Facebook uses AES-GCM to encrypt attachments sent via Secret Conversations [16], the end-to-end encryption feature in Messenger. The attack creates a "colliding" GCM ciphertext which decrypts to an abusive attachment via one key and an innocuous attachment via another. This combined with the behavior of Facebook's server-side abuse report generation code prevents abusive messages from being reported to Facebook. Since messages in Secret Conversations are called "salamanders" by Facebook (perhaps inspired by the Axolotl ratchet used in Signal, named for an endangered salamander), ensuring Facebook does not see a message essentially makes it an *invisible salamander*. We responsibly disclosed the vulnerability to Facebook. They have remediated it and have given us a bug bounty for reporting the issue.



Fast Message Franking: From Invisible Salamanders to Encryptment https://eprint.iacr.org/2019/016.pdf
Hunting Invisible Salamanders BlackHat 2020

Subscribe with Google (internal)

Caught during the design phase. 😅

No key commitment -> different contents for different keys

Malicious content could be served depending on the user

Solution: store and check a hash of the key

Amazon Encryption SDK

CVE-2020-8897

Fixed in version 2.0.0 (breaking compatibility)

Is GCM broken?

The property we exploit does not violate
any security goals of authenticated encryption

Many other encryption modes do not bind the key to the ciphertext

Moar impact?

Some systems with key rotations
just try all the keys of the key ring
until one decryption is authenticated
(newest one first)

Result

TIMECRYPTION

If you can abuse the key generation algorithm,
You can craft a tag that will be valid now for one key
  and also in the future with a different key
And the new key will just be silently used - by design

What you want now. What you want later. You control both.

CONTROL THE PRESENT.
CONTROL THE FUTURE.
IT'S NOT A BUG.

TimeCryption

How to

Use Mitra to generate binary polyglots

Mitra/utils/GCM:

- takes a mitra polyglot, encrypts and slices
- corrects authentication (crafts collision)

Step 1: run tool

```
mitra\utils\gcm>meringue.py S(24)-GZ-RAR.a7bccab6.rar.gz gzip-rar4.gcm
key 1: Now?
key 2: L4t3r!!!
ad   : MyVoiceIsMyPass!
blocks: 7

Computing 9 coefficients.
Coef to be inverted: 4494a66081751930eed248b6fd11c74e (already computed)

mitra\utils\gcm>_
```

Default values

Collide-GCM(K_1, K_2, N_a, C):

$H_1 \leftarrow E_{K_1}(0^{128})$; $H_2 \leftarrow E_{K_2}(0^{128})$
$P_1 \leftarrow E_{K_1}(N_a + 1)$; $P_2 \leftarrow E_{K_2}(N_a + 1)$
mlen $\leftarrow |C|/128 + 1$
lens $\leftarrow$ encode$_{64}(0) \,\|\, $encode$_{64}(|C| + 128)$
acc $\leftarrow$ lens $\times_{GF} (H_1 \oplus H_2) \oplus P_1 \oplus P_2$
For $i = 1$ to mlen $- 1$:
  $H_i \leftarrow H_1^{\text{mlen}+2-i} \oplus H_2^{\text{mlen}+2-i}$
  acc $\leftarrow$ acc $\oplus\, C[i] \times_{GF} H_i$
inv $\leftarrow (H_1^2 \oplus H_2^2)^{-1}$
$C_{\text{mlen}} \leftarrow$ acc $\times_{GF}$ inv
$C_a \leftarrow C \,\|\, C_{\text{mlen}}$
$T \leftarrow$ GHASH$(H_1, C_a) \oplus P_1$
Return $N_a \,\|\, C_a \,\|\, T$

Craft ciphertext:
launch Meringue, and wait 1 min if computation is needed

Authentically decrypt

```
mitra\utils\gcm>decrypt.py gzip-rar4.gcm
key1: b'Now?'
key1: b'L4t3r!!!'
ad: b'MyVoiceIsMyPass!'
nonce: 0
tag: b'6c4fd7abc224ac5323bca8b6a5f52f28'
Success!

plaintext1: b'1f8b080827b29d5e0400677a69702e74' ...
plaintext2: b'5f508c90ee9ba2b1bcb68fedb65e5ef2' ...

mitra\utils\gcm>_
```

Verify files

```
mitra\utils\gcm>file output1.gz
output1.gz: gzip compressed data, was "gzip.txt", last modified: Mon Apr 20
14:31:03 2020, max speed, from FAT filesystem (MS-DOS, OS/2, NT)

mitra\utils\gcm>unrar t output2.rar

UNRAR 5.40 beta 2 x64 freeware      Copyright (c) 1993-2016 Alexander Roshal


Testing archive output2.rar

Testing     rar4.txt                                              OK
All OK

mitra\utils\gcm>_
```

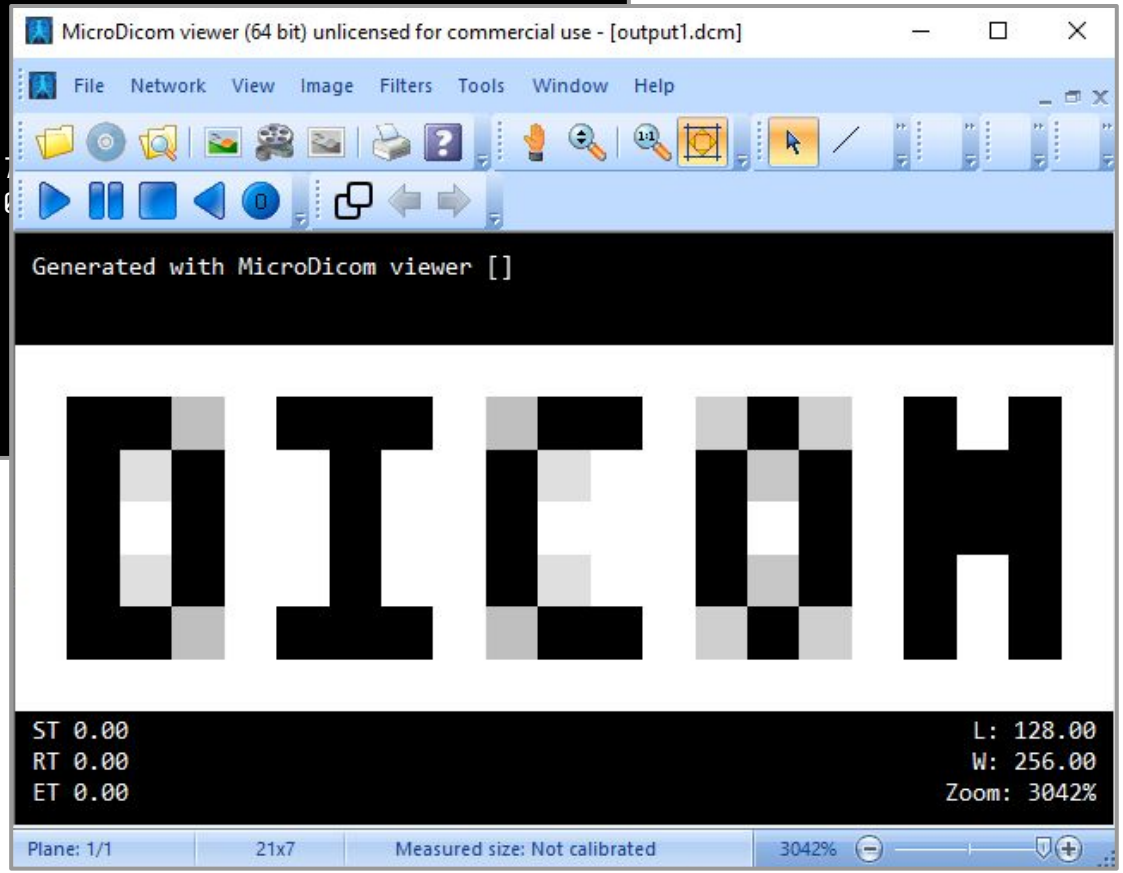
Step 2: Verify

Other examples (1/2)

```
/utils/gcm$ ./decrypt.py examples/gif-dicom.gcm
key1:  Now?
key1:  L4t3r!!!
ad:    MyVoiceIsMyPass!
nonce: 0
tag:   819fa575fa548b4ece54d573902e8f4f
Success!

plaintext1: 4749463837611900070080010000000 ...
plaintext2: 0792c2a0fe4826efbfb66896df2e7086 ...

/utils/gcm$ file output*
output1.gif: GIF image data, version 87a, 25 x 7
output2.dcm: DICOM medical imaging data
/utils/gcm$ _
```

```
/utils/gcm$ ./decrypt.py examples/pdf-exe.gcm
key1:  Now?
key1:  L4t3r!!!
ad:    AssociatedDataForAmbiguousCipher
nonce: 59334
tag:   43cf7debd82f644c6ec3f7873c91b3c4
Success!

plaintext1: 255044462d312e330a25c2b5c2b60a0a ...
plaintext2: 4d5a1526c3461a3f30486ccfd93e4a95 ...

/utils/gcm$ pdftotext output1.pdf -
http://www.evil.com


/utils/gcm$ wine ./output2.exe
32bit PE
/utils/gcm$ _
```

```
mitra\utils\gcm>..\decrypt.py dicom-pe.gcm
key1: b'Now?'
key1: b'L4t3r!!!'
ad: b'AssociatedDataForAmbiguousCipher'
nonce: 0
tag: b'c1c2de2194703324afe5e78347ae1d3d'
Success!

plaintext1: b'0d818498c9293fefb8b6e897df2e7
plaintext2: b'4d5a000000000000000000000000

mitra\utils\gcm>dicom-pe-1.28b89b4c.dcm

mitra\utils\gcm>dicom-pe-2.28b89b4c.exe
32-bit PE
mitra\utils\gcm>_
```



Other examples (2/2)

Overlapping bytes?

So far, each ciphertext byte belongs to strictly one payload

Control two outputs at once?

We know that $C = P_1 \text{ ^ } Ks_1$ and $C = P_2 \text{ ^ } Ks_2$

If we want to control some bytes of both P_1 and P_2,

we need that $P_1 \text{ ^ } P_2 = Ks_1 \text{ ^ } Ks_2$

We know that Keystreams depend on Keys, Nonce and Counter

-> bruteforce a nonce that gets the right xor value for both keys

Crypto-polyglots (w/ overlapping bytes)

- 2 formats starting at offset zero can coexist
  Ex:PDF/PE, JPG/PNG, ...
- both formats can be the same but w/ different contents
  Ex: JPG/JPG

	Minimal start offset																										Variable offset		Unsupported parasite				
	1	2	4	8		9	12	16	20	23	26	28	32	34	36	40	64	68	94	112	132	226	12	16	28							
	P	P	J	F	M	T	F	W	G	P	R	I	R	B	C	I	P	C	J	P	E	A	P	I	I	J		W	B	O		B	E	G	L	N
	S	E	P	l	P	I	L	A	Z	N	I	D	T	M	P	L	S	A	P	C	L	R	C	C	C	a		A	P	G		Z	B	I	N	E
			G	a	4	F	V	D		G	F	3	F	P	I	D	D	B	2	A	F		A	O	C	v		S	G	G		2	M	F	K	S
				c		F					F	v			O	A				P			P			a		M					L			
												2								N																
																				G																
1* PS	.	M	A	?	?	?	?	?	?	A	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?		?	?	?		?	?	?	?	?
2^ PE	M	.	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	!	!	!	!	!	!		M	M	M		!	!	!	!	!
4+ JPG	A	A	.	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M		M	M	M		M	M	M	M	M

(the table could go on but would take too long to bruteforce)

X: automated ?: likely possible
M: manual !: unknown

How to (1/2)

`nonce.py` just does what you expect

`mitra/utils/gcm`

```
mitra\utils\gcm>nonce.py
key1: b'4e6f773f' (b'Now?')
key2: b'4c34743372212121' (b'L4t3r!!!')
hdr1: b'2550' (b'%P')
hdr2: b'4d5a' (b'MZ')
start: 2 (GCM)
Results:
- nonce: 00059334 0x0000e7c6
- nonce: 00094306 0x00017062
- nonce: 00173940 0x0002a774
- nonce: 00407943 0x00063987
- nonce: 00405901 0x0006318d
- nonce: 00535752 0x00082cc8
- nonce: 00668403 0x000a32f3
- nonce: 01314505 0x00140ec9

mitra\utils\gcm>_
```

PDF and PE both have short magic,
so it's very fast to bruteforce

How to (2/2)

Just use the computed nonce

```
mitra\utils\gcm>decrypt.py examples\pdf-exe.gcm
key1: b'Now?'
key1: b'L4t3r!!!'
nonce: 59334
ad: b'AssociatedDataForAmbiguousCipher'
tag: b'43cf7debd82f644c6ec3f7873c91b3c4'
Success!

plaintext1: b'25504462d312e330a25c2b5c2b60a0a' ...
plaintext2: b'4d5a1526c3461a3f30486ccfd93e4a95' ...

mitra\utils\gcm>_
```

```
mitra\utils\gcm>file output1.pdf output2.exe
output1.pdf: PDF document, version 1.3
output2.exe: PE32 executable (console) Intel 80386, for MS Windows

mitra\utils\gcm>pdftotext output1.pdf -
http://www.corkami.com

mitra\utils\gcm>output2.exe
32bit PE

mitra\utils\gcm>_
```

It's fully generic, and works with most standard files

```
mitra\utils\gcm>decrypt.py examples\pdf-viewer.gcm
key1: b'Now?'
key1: b'L4t3r!!!'
ad: b'MyVoiceIsMyPass!'
nonce: 59334
tag: b'deadbeefcafebabe0000000000000000'
Success!

plaintext1: b'4d5a1526c3461a3f30486ccfd93e4a95' ...
plaintext2: b'25504462d312e330a25c2b5c2b60a0a' ...

mitra\utils\gcm>file output1.exe output2.pdf
output1.exe: PE32 executable (GUI) Intel 80386, for MS Windows,
output2.pdf: PDF document, version 1.3

mitra\utils\gcm>output1.exe output2.pdf

mitra\utils\gcm>_
```

A PDF viewer **executable** and an academic **document** ->
(neither source is public - only the binaries are)

Recap on overlapping polyglots

- Requires `--overlap` in Mitra:
  - It's disabled by default as the polyglots don't work as-is
  - The overlapping data is saved in the filename

    `0(4-84)-JPG[ICC]{000001C0}.5ecbd8cf.jpg.icc`
- Longer overlap takes longer to bruteforce
- That's all

Not just polyglots

We can make 2 files with the same type but with different contents

Declare a content w/ its length depending on decryption:

-> different comment lengths

-> different data being parsed

Ex: dual-contents JPG pair

The minimal structure to declare a JPG file type w/ a comment is:

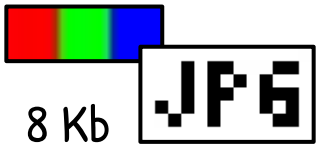
FF D8 FF FE XX XX

- JPEG magic signature
- comment declaration
- comment length

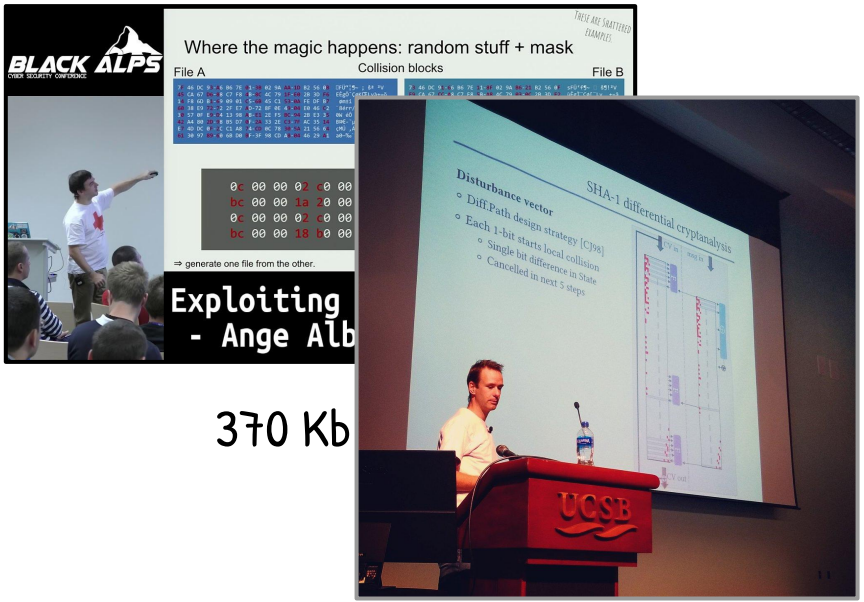
```
mitra\utils\gcm>nonce.py
key1: 4e6f773f ('Now?')
key2: 4c34743372212121 ('L4t3r!!!')
hdr1: ffd8fffe ('\xff\xd8\xff\xfe')
hdr2: ffd8fffe ('\xff\xd8\xff\xfe')
start: 2 (GCM)
Results:
- nonce: 8108314280 (0x1E34B0EA8)

mitra\utils\gcm>_
```

Reusable with any JPG files pair.

`jpg.py` (no need of recomputation)



8 Kb



370 Kb

Even further

Unlike SHA-1, the G_{Hash} function is invertible - "it's just maths".

-> you can even set a pre-defined tag

by using an extra block of the ciphertext for correction.

-> a file can tell in advance which tag will authenticate the decryption ;)

You could also correct the Authentication Data instead of the ciphertext.

How to Abuse and Fix Authenticated Encryption Without Key Commitment

Ange Albertini, Thai Duong, Shay Gueron, Stefan Kölbl, Atul Luykx, Sophie Schmieg

Cryptology ePrint Archive: Report 2020/1456 - last revised 11 Jun 2021

Abstract: Authenticated encryption (AE) is used in a wide variety of applications, potentially in settings for which it was not originally designed. Recent research tries to understand what happens when AE is not used as prescribed by its designers. A question given relatively little attention is whether an AE scheme guarantees ``key commitment'': ciphertext should only decrypt to a valid plaintext under the key used to generate the ciphertext. Generally, AE schemes do not guarantee key commitment as it is not part of AE's design goal. Nevertheless, one would not expect this seemingly obscure property to have much impact on the security of actual products. In reality, however, products do rely on key commitment. We discuss three recent applications where missing key commitment is exploitable in practice. We provide proof-of-concept attacks via a tool that constructs AES-GCM ciphertext which can be decrypted to two plaintexts valid under a wide variety of file formats, such as PDF, Windows executables, and DICOM. Finally we discuss two solutions to add key commitment to AE schemes which have not been analyzed in the literature: a generic approach that adds an explicit key commitment scheme to the AE scheme, and a simple fix which works for AE schemes like AES-GCM and ChaCha20Poly1305, but requires separate analysis for each scheme.

Category / Keywords: secret-key cryptography / authenticated encryption, robustness, commitment, key commitment, AES-GCM

Date: received 17 Nov 2020, last revised 11 Jun 2021

Contact author: aluykx at google com

Available format(s): PDF | BibTeX Citation

Note: Changes from previous version: improved overall readability, added a subsection on how to choose a fix, further clarified contributions versus prior work, fixed an oversight in the key commitment definition, and added full explanation of proof of theorem 1 (for AES-GCM padding fix).

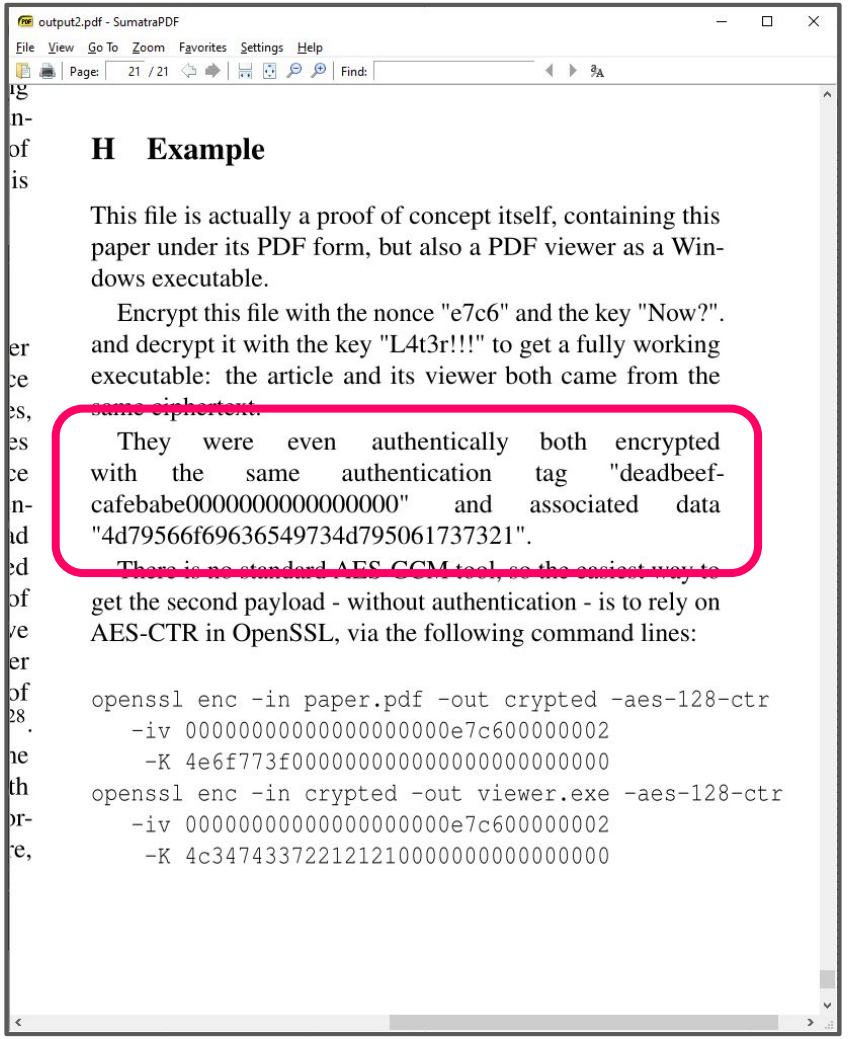
Our paper

https://eprint.iacr.org/2020/1456

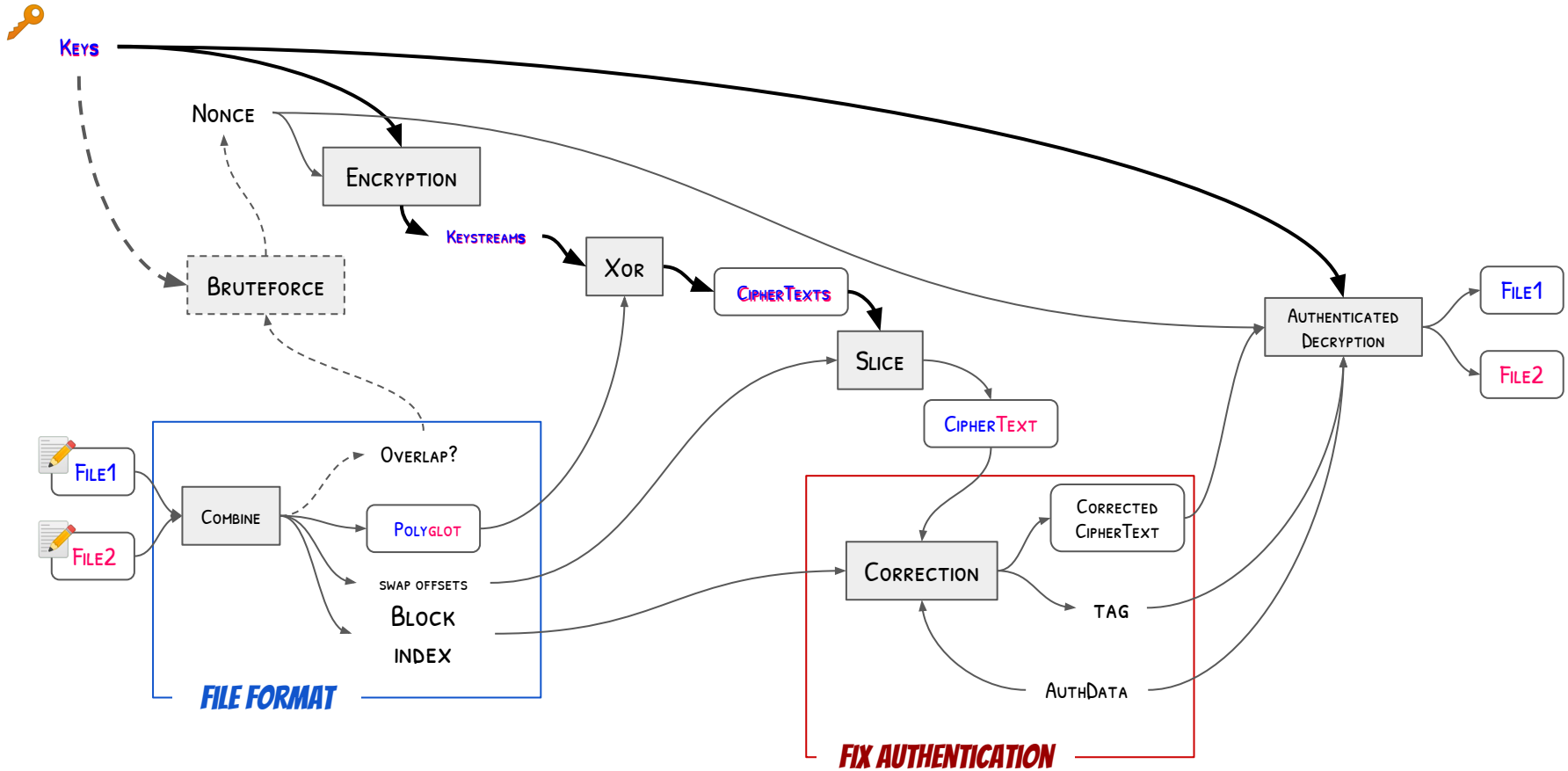
```
$ wget https://eprint.iacr.org/2020/1456.pdf
--2020-11-19 11:09:15--  https://eprint.iacr.org/2020/1456.pdf
Resolving eprint.iacr.org (eprint.iacr.org)... 216.184.8.41
Connecting to eprint.iacr.org (eprint.iacr.org)|216.184.8.41|:443...
connected.
HTTP request sent, awaiting response... 200 OK
Length: 2464928 (2.4M) [application/pdf]
Saving to: '1456.pdf'

1456.pdf            100%[===================>]   2.35M  1.65MB/s    in

2020-11-19 11:09:17 (1.65 MB/s) - '1456.pdf' saved [2464928/2464928]

$ openssl enc -in 1456.pdf -out crypted    -aes-128-ctr -iv
00000000000000000000e7c600000002 -K 4e6f773f00000000000000000000000000
$ openssl enc -in crypted  -out viewer.exe -aes-128-ctr -iv
00000000000000000000e7c600000002 -K 4c3474337221212100000000000000000
$ wine viewer.exe 1456.pdf
```

THE PDF ARTICLE FILE IS ALSO A PDF VIEWER EXECUTABLE!

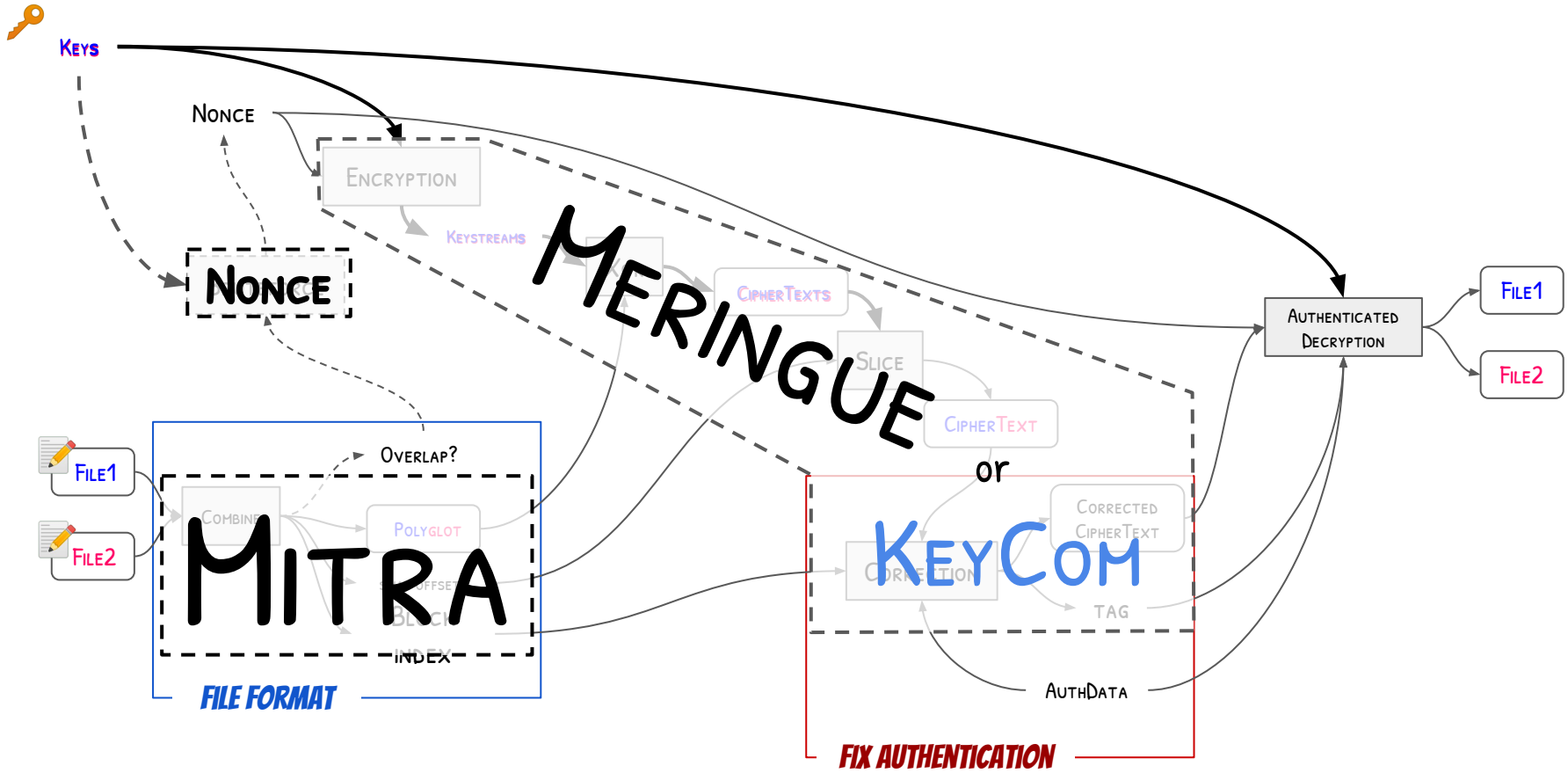
```
mitra\utils\gcm>decrypt.py examples\pdf-viewer.gcm
key1: b'Now?'
key1: b'L4t3r!!!'
ad: b'MyVoiceIsMyPass!'
nonce: 59334
tag: b'deadbeefcafebabe0000000000000000'
Success!

plaintext1: b'4d5a1526c3461a3f30486ccfd93e4a95' ...
plaintext2: b'255044462d312e330a25c2b5c2b60a0a' ...

mitra\utils\gcm>output1.exe output2.pdf

mitra\utils\gcm>_
```



Some PoCs know their secrets in advance 😉


Keys
Nonce
Encryption
Keystreams
Xor
CipherTexts
Bruteforce
Slice
CipherText
Authenticated Decryption
File1
File2
File1
File2
Combine
Overlap?
Polyglot
swap offsets
Block
index
FILE FORMAT
Correction
Corrected CipherText
TAG
AuthData
FIX AUTHENTICATION

Overview


Keys
Nonce
Encryption
Keystreams
Nonce
Meringue
CipherTexts
Slice
CipherText
Authenticated Decryption
File1
File2
File1
File2
Overlap?
Combine
Polyglot
Mitra
Index
File Format
or
KeyCom
Corrected CipherText
Correction
Tag
AuthData
Fix Authentication

Overview

Released tools

- Mitra: generates polyglots from any pair of files
  /utils/GCM: tooling to craft custom GCM exploits
- KeyCom: abuses GCM, OCB or GCM-SIV
  supports Mitra polyglots

...with many lightweight and free examples!

Conclusion

Authenticated Encryption

A.K.A. "Robustness".

A.E. WITHOUT KEY COMMITMENT

IS A SECURITY RISK

Pretty cool stuff

Craft. Upload. Download later. Pwned!

This is not the file you're looking for

Working as intended

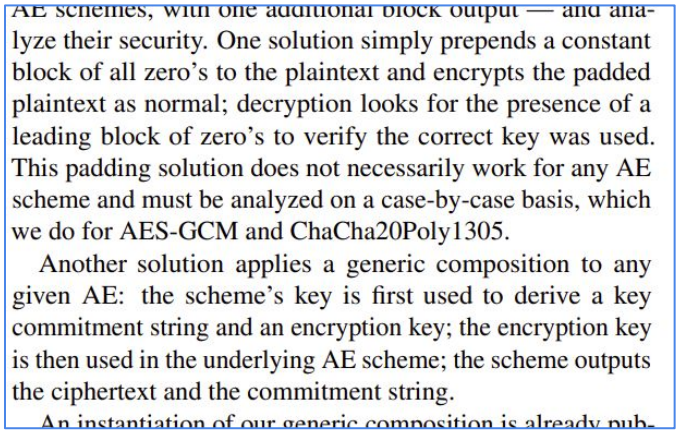
CTR & GCM DESIGNS HAD THESE WEAKNESSES FROM THE BEGINNING

Generic binary polyglots make them easy to exploit

Some easy solutions already exist

- Prepend zeros and check their presence after decryption.
- Store a hash of the key

Be careful with formats with cavities (Iso, Dicom…)!

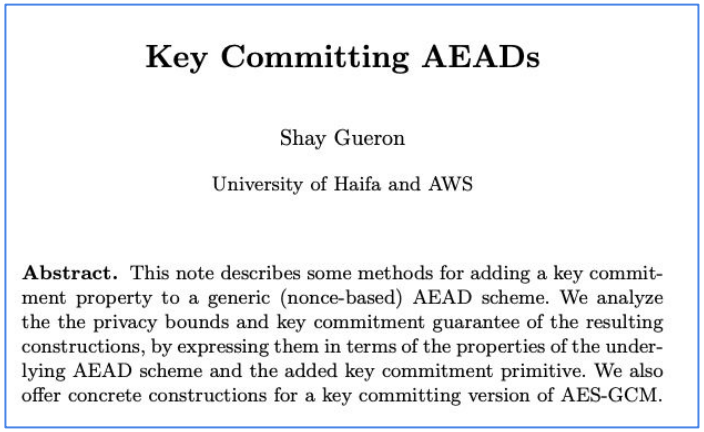


AE schemes, with one additional block output — and analyze their security. One solution simply prepends a constant block of all zero's to the plaintext and encrypts the padded plaintext as normal; decryption looks for the presence of a leading block of zero's to verify the correct key was used. This padding solution does not necessarily work for any AE scheme and must be analyzed on a case-by-case basis, which we do for AES-GCM and ChaCha20Poly1305.

Another solution applies a generic composition to any given AE: the scheme's key is first used to derive a key commitment string and an encryption key; the encryption key is then used in the underlying AE scheme; the scheme outputs the ciphertext and the commitment string.

An instantiation of our generic composition is already pub-

Our paper https://eprint.iacr.org/2020/1456



Key Committing AEADs

Shay Gueron

University of Haifa and AWS

Abstract. This note describes some methods for adding a key commitment property to a generic (nonce-based) AEAD scheme. We analyze the the privacy bounds and key commitment guarantee of the resulting constructions, by expressing them in terms of the properties of the underlying AEAD scheme and the added key commitment primitive. We also offer concrete constructions for a key committing version of AES-GCM.

Key Committing AEADs

Useless binary polyglots?

Generic and instant

crypto-polyglots are even cooler

Not a standard file™

One more thing...

Not just CTR/GCM!

GCM-SIV AND OCB3 ARE EXPLOITABLE TOO

Some extra constraints, but similar ideas

Extra constraints

Block alignment (OCB3, SIV): just need to pad to align payloads

More blocks: OCB3 requires 270 blocks, but that's only 4kb

Computing time:

- GCM-SIV require more time,
- relative to payload size: 5h30min for 300 kb

How?

Just run our scripts on Mitra polyglots

That's... all!

Python with extra math stuff to keep things readable

https://www.sagemath.org/

```
$ sage mitra_gcm.sage examples/input/S\(24\).gz.rar -p
Key1: b'01010101010101010101010101010101'
Key2: b'02020202020202020202020202020202'
Nonce: b'030303030303030303030303'
AdditionalData: b'aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
Ciphertext: b'26ce232179cfdf1af01e887f3f5a6e840a7
Tag: b'04040404040404040404040404040404'
$ _
```

```
$ unrar vt gcm2.bin

UNRAR 5.61 beta 1 freeware      Copyright (c) 1993-2018 Alexander Roshal

Archive: gcm2.bin
Details: RAR 4, SFX

       Name: rar4.txt
       Type: File
       Size: 4
 Packed size: 4
      Ratio: 100%
      mtime: 2020-01-18 19:08:40,946092200
 Attributes: ..A....
      CRC32: 982134A1
```

```
$ gzip -lv gcm1.bin
method  crc     date  time           compressed        uncompressed  ratio uncompressed_name
defla 28056779 Dec 14 07:55                176                 144  -3.5% gcm1.bin
$ _
```

https://github.com/kste/keycommitment

```
$ sage mitra_ocb.sage examples/input/Yes-No\(30-250-4d0\).pdf.pdf  -p
Key1: b'01010101010101010101010101010101'
Key2: b'02020202020202020202020202020202'
Nonce: b'030303030303030303030303'
Ciphertext: b'2c92143d3c1ecf1c76c24c3a1b4e041fa2bd14a0469ebad98e607753b28c4643'
Tag: b'04040404040404040404040404040404'
$
```

```
$ pdftotext ocb1.bin -
NO!

$ pdftotext ocb2.bin -
YES
```

```
$ sage mitra_ocb.sage examples/input/PE-PDF\(200-3D0-5E0\).exe.pdf  -p
Key1: b'01010101010101010101010101010101'
Key2: b'02020202020202020202020202020202'
Nonce: b'030303030303030303030303'
Ciphertext: b'854ab0ee6047c9da5e3c49e6f244ae9eda276584b786c43f6b03d7a8e2a78af4'
Tag: b'04040404040404040404040404040404'
$ _
```

```
$ wine ocb1.bin
Hello World!
(Windows executable)
$ pdftotext ocb2.bin -
PDF

$ _
```

Using OCB3 and GCM-SIV scripts

```
$ sage mitra_siv.sage examples/input/Yes-No\(30-250-4d0\).pdf.pdf  -p
Key1: b'01010101010101010101010101010101'
Key2: b'02020202020202020202020202020202'
Nonce: b'030303030303030303030303'
Ciphertext: b'923d5ed7812b036b4d38f8fc884c9cf387cc22fc65d6d2cce71e5c30c0c01516'
Tag: b'04040404040404040404040404040404'
$ _
```

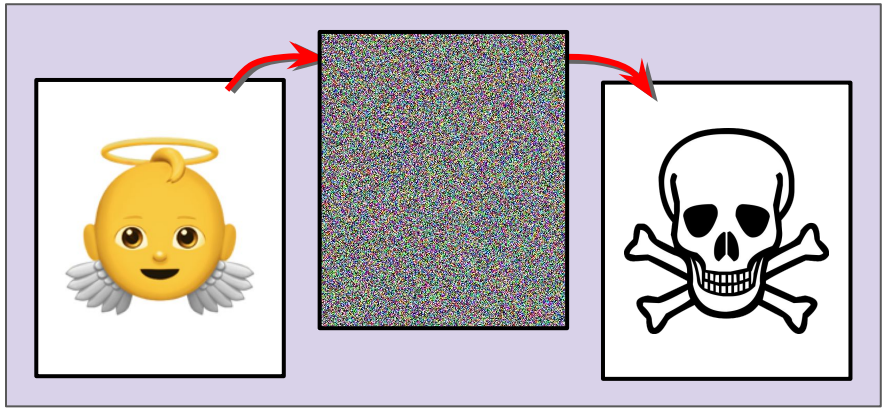
```
$ pdftotext siv1.bin -
YES

$ pdftotext siv2.bin -
NO

$
```

```
$ sage mitra_siv.sage examples/input/PE-PDF\(200-3D0-5E0\).exe.pdf  -p
Key1: b'01010101010101010101010101010101'
Key2: b'02020202020202020202020202020202'
Nonce: b'030303030303030303030303'
Ciphertext: b'fa371a91ac1a2d58471d3a494afa96c2a7fc029307bcd8f0db311055aea7617e'
Tag: b'04040404040404040404040404040404'
$ _
```

```
$ wine siv1.bin
Hello World!
(Windows executable)
$ pdftotext siv2.bin -
PDF

$ _
```

Thank you!

Any feedback is welcome!



Special thanks to:

Daniel Bleichenbacher, Atul Luykx,
Sophie Schmieg, Thai Duong, Shay Gueron,
Philippe Teuwen, Thaís Moreira Hamasaki

ANGE ALBERTINI

reverse engineering

VISUAL DOCUMENTATIONS

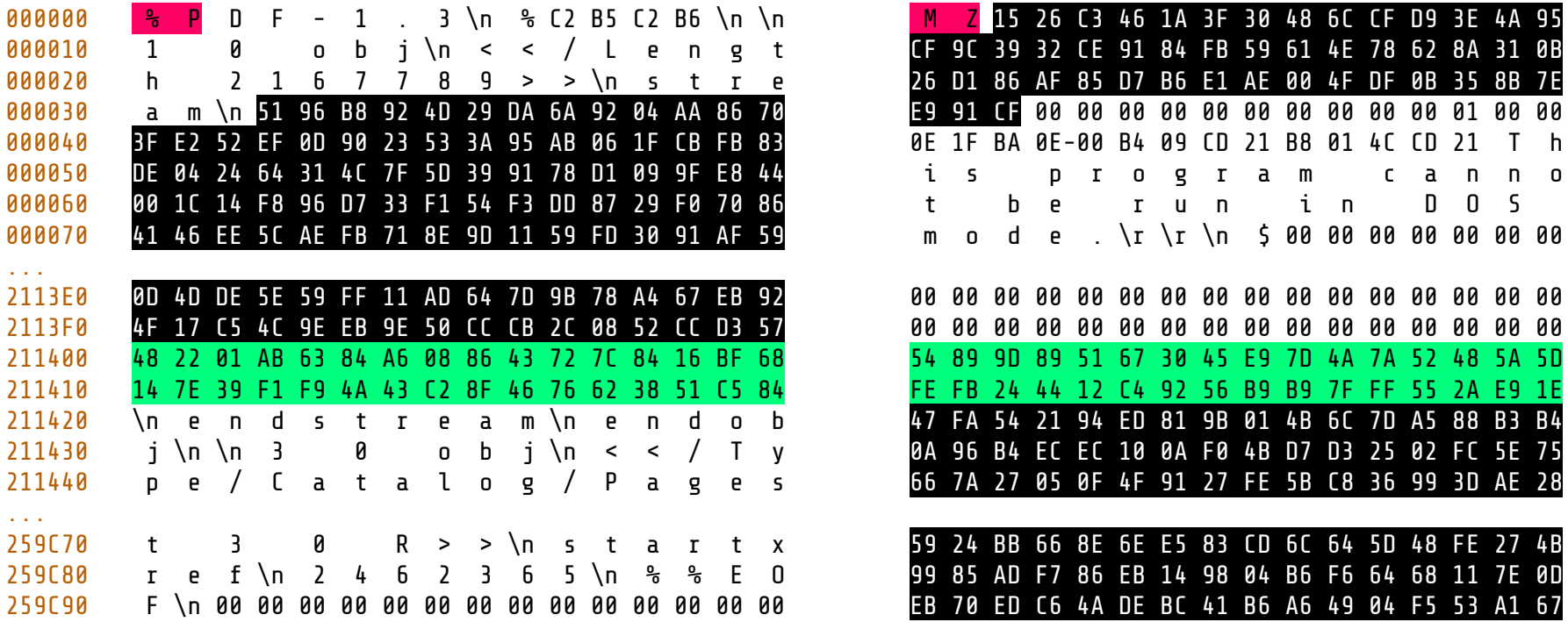
@angealbertini

ange@corkami.com

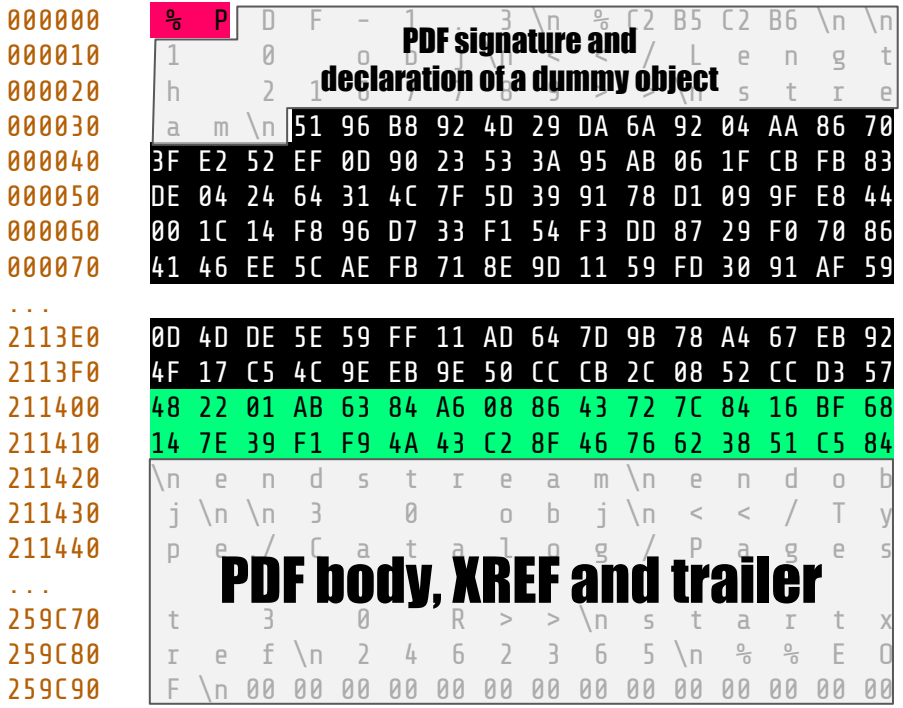
http://www.corkami.com

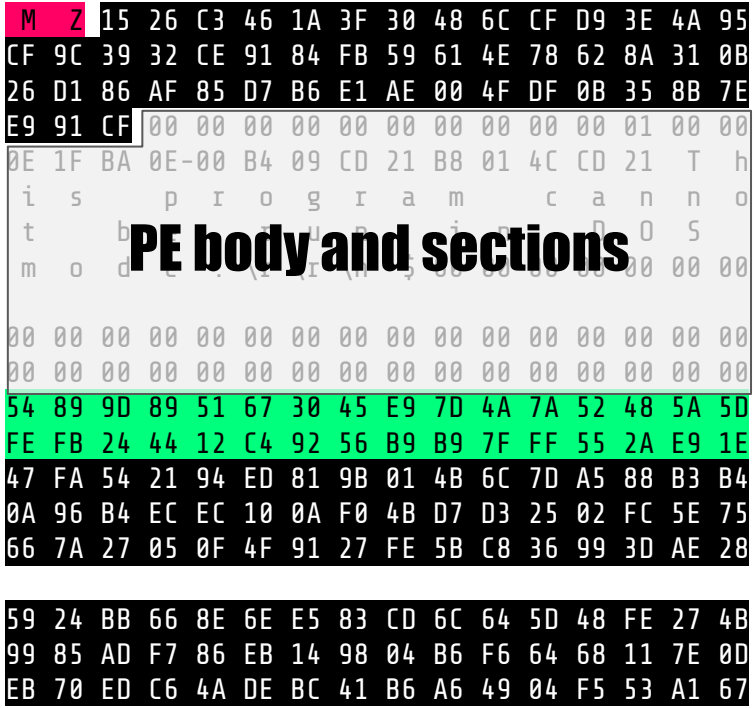
Bonus

PoC walkthrough

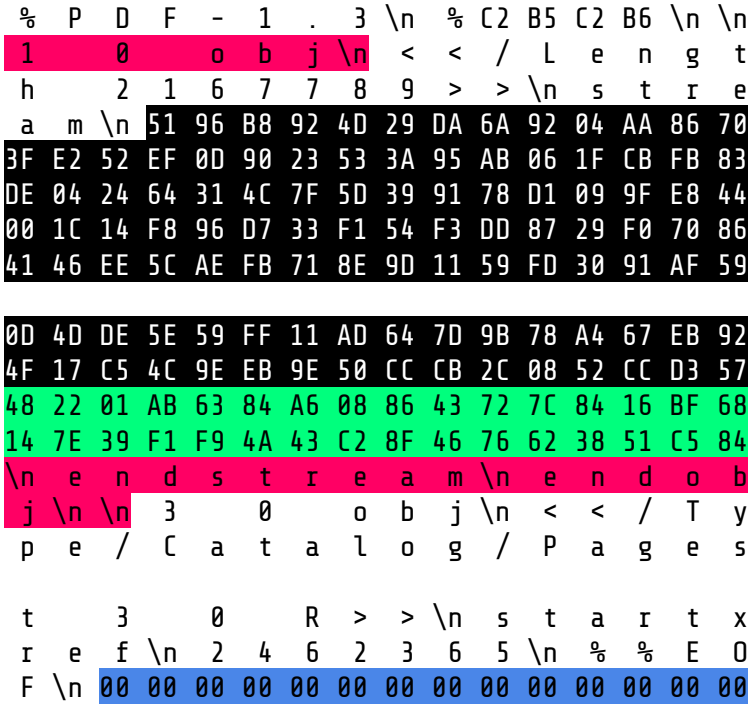


The 2 plaintexts: 2 bytes of overlapping contents, 2 correction blocks





THE PDF CONTAINS THE MOST OF THE PE PAYLOAD IN A DUMMY OBJECT



The PDF is standard

The first object is unreferenced

A tiny appended data

(that could be avoided with alignment)

The rest is fully standard

The PE is almost standard

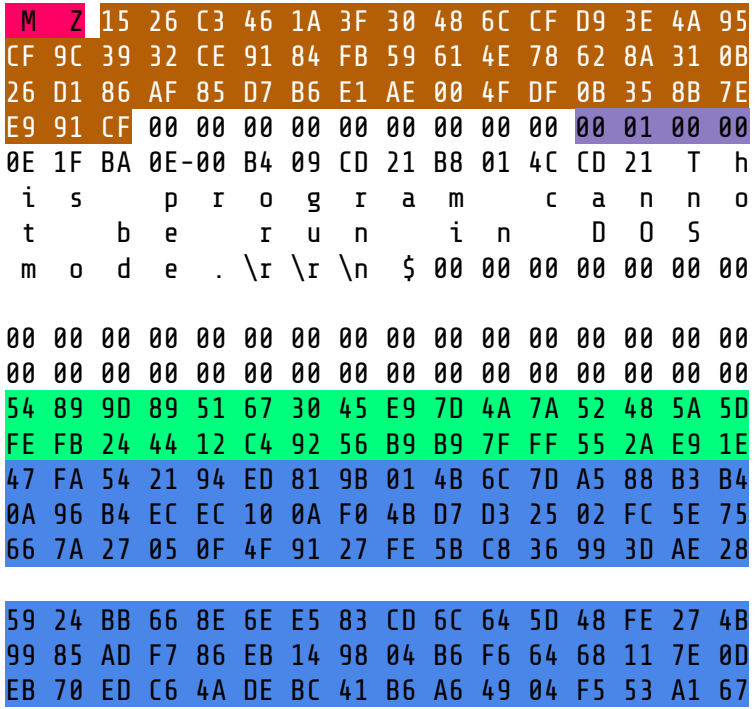
The DOS header is mostly overwritten

it's required but irrelevant nowadays anyway

However, the pointer to PE header is preserved

The rest of the PE is unmodified

The encrypted PDF is just appended data



PoC DIY

https://github.com/corkami/mitra/tree/master/utils/extra

```
mitra/utils/extra$pdfpe.py paper.pdf SumatraPDF18fixed.exe
 * normalizing, merging with a dummy page
 * removing dummy page reference
 * fixing object references
 * aligning PE header
 * finalizing main PDF
 * generating polyglot
Success!

mitra/utils/extra$../gcm/meringue.py -i 135488 -n 59334 'Z(2-33-211420).exe.pdf' test.gcm
key 1: Now?
key 2: L4t3r!!!
ad   : MyVoiceIsMyPass!
blocks: 154060

Computing 154062 coefficients.
Coef to be inverted: 78a9f0686e9b7972252c8ca3796e3100 (already computed)

mitra/utils/extra$_
```

A specially prepared executable

Run the dedicated script

Mitra itself can't do it because of the bytes overlap via nonce

```
$ python3 ./mitra.py in/jpg.jpg ./in/pe32.exe  --verbose
> Arguments parsing:
> Verbose is ON
in/jpg.jpg
File 1: JFIF / JPEG File Interchange Format
./in/pe32.exe
File 2: Portable Executable (hdr)

> Stack: JPG-PE(hdr)
> ! File type 2 (PE(hdr)) starts at offset 0 - it can't be appended.
> Parasite: JPG[PE(hdr)]
> ! File type 1 (JPG) can only host parasites at offset 0x6. File 2 should start at offset 0x0 or less.
> Zipper: JPG^PE(hdr)
> ! File type 1 (JPG) doesn't support zippers.
> Cavity: JPG_PE(hdr)
> ! File type 2 (PE(hdr)) doesn't start with any cavity.
$ _
```

BOTH FORMATS START AT OFFSET ZERO

JPG AND PE: NO STANDARD POLYGLOT IS POSSIBLE

```
$ python3 ./mitra.py in/jpg.jpg ./in/pe32.exe  --verbose --overlap
> Arguments parsing:
> Verbose is ON
> Overlap is ON
in/jpg.jpg
File 1: JFIF / JPEG File Interchange Format
./in/pe32.exe
File 2: Portable Executable (hdr)

> Stack: JPG-PE(hdr)
> ! File type 2 (PE(hdr)) starts at offset 0 - it can't be appended.
> Parasite: JPG[PE(hdr)]
> ! File type 1 (JPG) can only host parasites at offset 0x6. File 2 should start at offset 0x0 or less.
> Zipper: JPG^PE(hdr)
> ! File type 1 (JPG) doesn't support zippers.
> Cavity: JPG_PE(hdr)
> ! File type 2 (PE(hdr)) doesn't start with any cavity.
> PE Reverse overlapping parasite
> HIT JPG;PE(hdr)
> Overlapping parasite
> HIT JPG;PE(hdr)
$ _                                   -> generates OR(6-a00)-JPG[PE(hdr)]{4D5A}.e1d0b0d9.jpg.exe
```

JPG and PE: near-polyglots are possible...

Not a working polyglot without crypto!

```
$ sage mitra_gcm.sage OR(6-a00)-JPG[PE(hdr)]{4D5A}.e1d0b0d9.jpg.exe -p
Overlap file found - bruteforced nonce: 11671
Key1: b'01010101010101010101010101010101'
Key2: b'02020202020202020202020202020202'
Nonce: b'000000000000000000002d97'
AdditionalData: b'aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa'
Ciphertext: b'6a8b8127629cbc5c20d6518f6c75b31ec39c5d77b3db4314b6a5f7ff134ec266'
Tag: b'04040404040404040404040404040404'
$ _
```

ONLY 2 BYTES OF OVERLAP! (QUICK TO BRUTEFORCE)

Generates also `gcm1.bin` and `gcm2.bin` from the same ciphertext

```
$ file gcm*.bin
gcm1.bin: JPEG image data, baseline, precision 8, 6x2, components 3
gcm2.bin: PE32 executable (console) Intel 80386, for MS Windows
$ identify gcm1.bin
gcm1.bin JPEG 6x2 6x2+0+0 8-bit sRGB 2928B 0.000u 0:00.000
$ wine ./gcm2.bin
 * a 'compiled' PE
$ _
```

BOTH FORMATS START AT OFFSET ZERO

...AND QUICK TO BRUTEFORCE

A JPEG trick

Saving a few bytes of bruteforcing

Do you feel lucky?

Structure of a JPEG file

- A sequence of segments
- Each segment starts with FF then a segment marker byte
- Most segments then start with their big-endian length on 2 bytes

...the rest is complex

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	FF	D8	FF	E0	00	10	J	F	I	F	00	01	01	02	00	24
10	00	24	00	00	FF	DB	00	43	00	01	01	01	01	01	01	01
20	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01
30	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01
40	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01	01
50	01	01	01	01	01	01	01	01	01	FF	C0	00	0B	08	00	38
60	00	68	01	01	11	00	FF	C4	00	29	00	01	01	01	01	00
70	00	00	00	00	00	00	00	00	00	00	00	00	0B	04	0A	10
80	01	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
90	00	FF	DA	00	08	01	01	00	00	3F	00	EF	E0	00	00	06
A0	76	80	40	21	7F	74	02	05	FB	C1	01	01	7F	70	10	08
B0	5F	DD	00	85	FD	D0	08	5F	DD	00	85	FD	C0	04	02	17
C0	F7	40	20	5F	DC	40	20	17	F7	10	0F	5F	C1	00	85	FD
D0	D0	08	5F	DC	10	08	5F	DD	00	85	FD	C6	74	04	17	F7
E0	10	08	5F	DC	04	02	05	FD	C0	00	00	07	FF	D9		

00: FF D8 Start Of Image (size: n/a)

02: FF E0 Application 0 (size: 10)

14: FF DB Define a Quantization Table (size: 43)

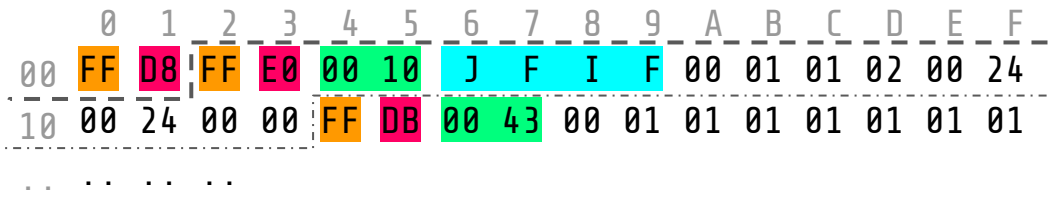
59: FF C0 Start Of Frame 0 (size: 0B)

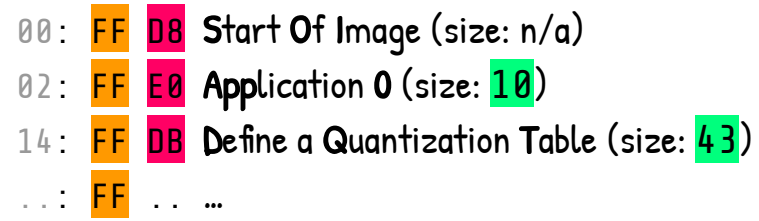
66: FF C4 Define Huffman table (size: 29)

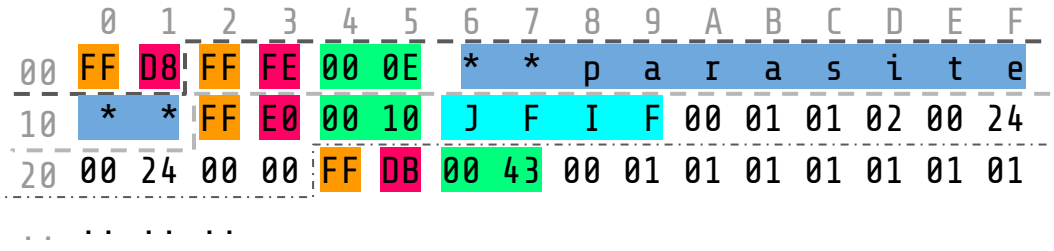
91: FF DA Start of Scan (size: n/a)

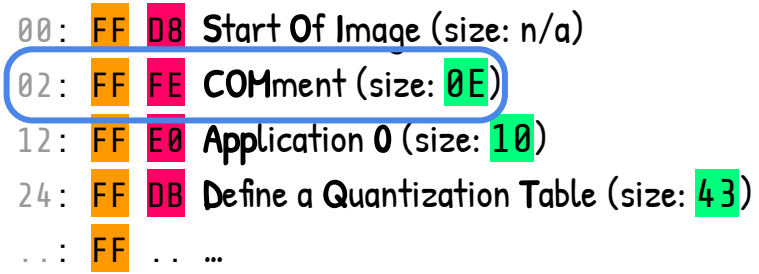
EC: FF D9 End Of Image (size: n/a)

Structure of a small JPEG image









Parasitizing: insert a comment segment FF FE

How many required bytes to control ?

- 6, to set an exact comment length

FF FE XX YY

- 5, to set a length rounded-up to `0x100`

FF FE XX[+1] ??

The length is big endian: `XX YY` => length of `0XXYY`

- 4, if our parasite fits in the random length 🤞

FF FE ?? ??

64kb more data is nothing weird for a JPG image

-> 2^{16} speed-up

Do I feel lucky?

How to...

Generate quickly a JPG overlap polyglot

```
$ mitra.py in/jpg.jpg in/icc.icc --verbose --overlap
> Arguments parsing:
> Verbose is ON
> Overlap is ON
in\jpg.jpg
File 1: JFIF / JPEG File Interchange Format
in\icc.icc
File 2: ICC / International Color Consortium profiles

> Stack: JPG-ICC
> ! File type 2 (ICC) starts at offset 0 - it can't be appended.
> Parasite: JPG[ICC]
> ! File type 1 (JPG) can only host parasites at offset 0x6. File 2 should start at offset 0x0 or less.
> Zipper: JPG^ICC
> ! File type 1 (JPG) doesn't support zippers.
> Cavity: JPG_ICC
> ! File type 2 (ICC) doesn't start with any cavity.
> Overlapping parasite
> Jpeg overlap file: reducing two bytes
>   (don't forget to postprocess after bruteforcing)
> HIT ICC;JPG
Generic overlapping polyglot file created.

$ _
```

1/4 GENERATE AN OVERLAPPING POLYGLOT

```
$ utils/gcm/nonce.py "O(4-84)-JPG[ICC]{000001C0}.5ecbd8cf.jpg.icc"
key1: b'01010101010101010101010101010101' (b'\x01\x01\x01\x01\x01\x01\x01\x01\x01\x01\x01\x01\x01\x01\x01\x01')
key2: b'02020202020202020202020202020202' (b'\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02')
hdr1: b'000001c0' (b'\x00\x00\x01\xc0')
hdr2: b'ffd8fffe' (b'\xff\xd8\xff\xfe')
start: 2 (GCM)
# - found 0x115e0000014e11ec [thread:057 time 0:05:20.259989]

$ _
```

```
$ utils/gcm/jpg4fix.py "O(4-84)-JPG[ICC]{000001C0}.5ecbd8cf.jpg.icc" 0x115e0000014e11ec
Nonce: 1251437746477470188 0x115e0000014e11ec
Keys: b'\x01\x01\x01\x01\x01\x01\x01\x01\x01\x01\x01\x01\x01\x01\x01\x01' b'\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02'
Old length: 128 0x80
New length: 25883 0x651b

$ _
```

It will generate 4-O(4-84)-JPG[ICC]{000001C0}.5ecbd8cf.jpg.icc

2/4 BRUTEFORCE NONCE, FIX POLYGLOT

Use the fixed file!

```
$ utils/gcm/meringue.py -k 01010101010101010101010101010101 02020202020202020202020202020202
    -i 9 -n 0x115e0000014e11ec
    "4-O(4-84)-JPG[ICC]{000001C0}.5ecbd8cf.jpg.icc" jpg-icc.gcm
key 1: \x01\x01\x01\x01\x01\x01\x01\x01\x01\x01\x01\x01\x01\x01\x01\x01
key 2: \x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02
ad   : MyVoiceIsMyPass!
blocks: 1633

Computing 1635 coefficients.
Coef to be inverted: 86aabaa18fba4e751da9a6de05c6159f (not present)
Inverting the coefficient (takes a few mins)...
New invert: 86aabaa18fba4e751da9a6de05c6159f 013e74df0bdab42a43e625642b293fdf

$ _
```

3/4 Generate ambiguous ciphertext from the fixed file

```
$ utils/gcm/decrypt.py jpg-icc.gcm
key1: b'\x01\x01\x01\x01\x01\x01\x01\x01\x01\x01\x01\x01\x01\x01\x01\x01'
key1: b'\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02\x02'
ad: b'MyVoiceIsMyPass!'
nonce: 1251437746477470188
tag: b'b3a546dc3faa0f3707e1ef66f6e9411d'
Success!

Key2: b'02020202020202020202020202020202'
Nonce: b'000000000000000000002d97'
AdditionalData: b'aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa'
Ciphertext: b'6a8b8127629cbc5c20d6518f6c75b31ec39c5d77b3db4314b6a5f7ff134ec266'
Tag: b'04040404040404040404040404040404'

$ _
```

```
$ file jpg-icc*
jpg-icc.b3474cf7.icc: Microsoft color profile 3.2, type lcms, GRAY/Lab-prtr device by
lcms, 448 bytes, 13-1-2009 16:10:20, no copyright tag
jpg-icc.b3474cf7.jpg: JPEG image data, baseline, precision 8, 104x56, frames 1

$ _
```

4/4 DECRYPT & TEST

Details of a Mitra polyglot file name

`O(4-84)-JPG[ICC]{000001C0}.5ecbd8cf.jpg.icc`

- layout type: **S**tack / **O**verlapping / **P**arasite / **C**avity / **Z**ipper
- (Slices): offsets where the content change side

Used for mixing contents after encryption.
(Imagine two sausages sliced in blocks and mixed)

- type layout: Tells which format is the host, which is the parasite
- {Overlapping data}: the "other" bytes of the file start

Used to bruteforce nonces.

- partial hash - to differentiate outputs
- file extensions - to ease testing