How to Abuse and Fix Authenticated Encryption Without Key Commitment

Ange Albertini and Thai Duong, *Google Research;* Shay Gueron, *University of Haifa and Amazon;* Stefan Kölbl, Atul Luykx, and Sophie Schmieg, *Google Research*

https://www.usenix.org/conference/usenixsecurity22/presentation/albertini

This artifact appendix is included in the Artifact Appendices to the Proceedings of the 31st USENIX Security Symposium and appends to the paper of the same name that appears in the Proceedings of the 31st USENIX Security Symposium.

August 10–12, 2022 • Boston, MA, USA

978-1-939133-31-1

Open access to the Artifact Appendices to the Proceedings of the 31st USENIX Security Symposium is sponsored by USENIX.

A Artifact Appendix

A.1 Abstract

Our artifacts provide a program to identify files of different formats and pair them together if these formats makes it possible, as polyglot or optionally as near polyglots. Other programs we provide can then turn these files into ambiguous ciphertexts, and then decrypt these ciphertexts to verify the validity of the payloads.

Many examples of input files and ambiguous ciphertexts are also provided.

A.2 Artifact check-list (meta-information)

- **Program: Mitra, Key Commitment**
- **Compilation: Python 3 (Mitra), SageMath (Key Commitment)**
- **Output: [near] polyglots and ambiguous ciphertexts**
- **How much disk space required (approximately)?: 6 Mb**
- **How much time is needed to prepare workflow (approximately)?: 5 min**
- **How much time is needed to complete experiments (approximately)?: 5 mins**
- **Publicly available?: Y**
- **Code licenses (if publicly available)?: MIT**

A.3 Description

A.3.1 How to access

Complete source and examples are available at :

- the Mitra repository (`https://github.com/corkami/mitra/` at tag `Usenix22`) – for file manipulation and ciphertext generation with GCM.
- the Key Commitment repository (`https://github.com/kste/keycommitment/` at tag `Usenix22`) – for ciphertext generation with GCM, GCM-SIV and OCB3.

A.3.2 Software dependencies

Python 3 and SageMath with the Cryptography, PyCryptodome, PyMuPDF and BitVector packages.

A.4 Installation

Clone or download the repositories and install the extra packages if needed. Note that, depending on your SageMath installation, you might have to install the Python packages *within* SageMath, e.g.: `sage -python3 -m pip install pycryptodome`.

A.5 Evaluation and expected results

A.5.1 Ambiguous PDF/PE file

The PDF/PE combination was used because it can be included in the release of the IACR archive (`https://ia.cr/2020/1456`) – the article file itself is a proof of concept.

1. Generate a near polyglot

   From any PDF file, in `mitra/utils/extra` run `pdfpe.py <file.pdf> SumatraPDF18fixed.exe` (it works with other Windows executables – this one is provided as an example).

   You get a near polyglot file named like `Z(2-33-211420).exe.pdf`.

   Keep the file name intact, as it uses a special syntax to store extra information: in this case 2, 33 and 221420 are the offsets where the polyglot content switches to the other file type.

2. Generate an ambiguous ciphertext

   From the `Key Commitments` repository, run the following command :

   ```
   sage mitra_gcm.sage
     -k 4e6f773f00000000000000000000000000
        4c3474337221212100000000000000000
     -n 000000000000000000e7c6
     -a 4d79566f69636549734d795061737321
     "Z(2-33-211420).exe.pdf" -p > poly.gcm
   ```

3. Decrypt and validate the different payloads from the ambiguous ciphertext

   From the directory `mitra/utils/gcm`, run the `decrypt.py poly.gcm` command line.

   You obtain from the same ciphertext the original PDF document and a windows executable – that is a PDF viewer in this case.

   The files have changed of course, but they behave like the original input files.

A.5.2 Ambiguous HTML file

The HTML/HTML combination was used because it's tiny and demonstrates the vulnerability which can occur in a service like *Subscribe with Google* if an AEAD without key-commitment is used.

1. Use the following example files

   - `normal.htm` :

     ```
     <html>Hello World!</html>
     ```

- `evil.htm`:
  `<html><a href="http://www.evil.com">Click here!</a></html>`

2. Generate an ambiguous HTML file

   From `mitra/utils/extra`, run `htmhtm.py normal.htm evil.htm`, and you'll get a file called `(4-26)7.d3f286cd.htm.htm`.

3. Generate an ambiguous ciphertext

   in `mitra/utils/gcm`, run the following command:

   ```
   meringue.py
     -i 7  "(4-26)7.d3f286cd.htm.htm"
     attack.gcm
   ```

4. Validate the ambiguous ciphertext by extracting the different plaintexts

   From the `mitra/utils/gcm` directory, run `decrypt.py attack.gcm`.

A.5.3 Ambiguous JPEG/? file

This combination is mentioned because the brute-forcing was reduced to 4 bytes (as opposed to 6 bytes in prior works) and requires a special workflow with post-processing of the near polyglot.

1. Generate a (non-working) near polyglot

   Use with any file that is supported with JPEG near polyglots, for example ICC files.

   ```
   mitra.py <file.jpg> <file2.bin> --verbose
   --overlap
   ```

   If ran on a JPEG, it will output the following warning :

   ```
   > Jpeg overlap file: reducing two bytes
   > (don't forget to post-process
       after bruteforcing)
   ```

   and generate a near polyglot file.

2. Find a valid nonce for the near polyglot

   From `mitra/utils/gcm/`, run the command `nonce.py <near_polyglot>`.

   This bruteforcing operation will use the 2-bytes shortcut but requires extra post-processing. This script will output a nonce value.

3. Post-process the near polyglot

   Run `jpg4fix.py <near_polyglot> <nonce>`. It will generate a fixed near polyglot starting with `4-`.

4. Generate the ambiguous ciphertext

   Run the following command:

   ```
   meringue.py
    -k 01010101010101010101010101010101
       02020202020202020202020202020202
    -i <index_offset>
    -n <nonce>
    <fixed_near_poly>
    ambiguous.gcm
   ```

   This execution will generate an ambiguous ciphertext.

   The index argument is the block index of the file where the TAG will be written. The indexed block should be blank : if the near polyglot file doesn't have such a block, you might want to add appended data to the polyglot itself or to the parasite inside.

5. Generate and validate the different payloads

   In the `mitra/utils/gcm/` folder, run the following `decrypt.py ambiguous.gcm` command.

   Once again, you'll get different files that just work like the input files.

A.6 Experiment customization

These file operations will work with different cryptographic parameters (keys, nonces, index), and other block ciphers with reasonable code modifications.

Many other file formats are supported by Mitra and can be combined as polyglots or near polyglots. Make sure you use **standard input files**: weird files created by Mitra might not be supported by Mitra itself as they may not have a standard structure anymore – you may want to use the `input/*` files provided in Mitra as a start.

Use the `--verbose` flag to get more feedback (e.g. why a polyglot was not generated). Use the `--reverse` flag if you're not sure which files should come first and last in the command line.

Other block cipher modes such as **OCB and GCM-SIV are supported** and included in the Key Commitment repository.

Use the `mitra_ocb.sage` or `mitra_siv.sage` scripts to generate ambiguous ciphertexts, and the `decrypt_ocb.sage` or `decrypt_siv.sage` scripts accordingly to decrypt payloads.

Unlike other modes that have byte granularity, GCM-SIV and OCB3 require **block alignment**. You may want to add two blocks of pre-padding and post-padding to the parasite when generating the [near] polyglot files.

For GCM-SIV, the **complexity** of generating an ambiguous ciphertext requires solving a system of linear equations of size relative to the files size, while it's constant for the other modes. The other expensive operation is bruteforcing the nonce of ambiguous ciphertext from near polyglots, which depends on the size of the overlap.