Partitioning Oracle Attacks

Julia Len        Paul Grubbs        Thomas Ristenpart
Cornell Tech

ePrint 2020/1491
Authenticated Encryption

Nonce N
Plaintext M
C ← AEAD.Enc(K, N, M)
Authenticated Encryption

Nonce N
Plaintext M
C $\leftarrow$ AEAD.Enc($\text{key}$, N, M)

N $\parallel$ C

M $\leftarrow$ AEAD.Dec($\text{key}$, N, C)
Authenticated Encryption

Nonce N
Plaintext M
C ← AEAD.Enc(K, N, M)

N || C

M ← AEAD.Dec(K, N, C)

Popular
• AES-GCM
• XSalsa20/Poly1305
• ChaCha20/Poly1305
• AES-GCM-SIV

Easy to use
• Efficient
• Standardized
• Library implementations

Secure
• Proven CCA-secure
• Confidentiality
• Integrity
Authenticated Encryption

Nonce N
Plaintext M
C ← AEAD.Enc( , N, M)

N || C

M ← AEAD.Dec( , N, C)

But don’t target robustness, also called committing AEAD, as a security goal
[ABN TCC’10], [FLPQ PKC’13] for PKE, [FOR FSE’17] for AEAD

- AES-GCM
- XSalsa20/Poly1305
- ChaCha20/Poly1305
- AES-GCM-SIV

- Efficient
- Standardized
- Library implementations

- Proven CCA-secure
- Confidentiality
- Integrity
(Non-) Committing AEAD

Nonce $N'$
Ciphertext $C'$

\[ M \leftarrow \text{AEAD.Dec}(\text{key}, N', C') \]

\[ M^* \leftarrow \text{AEAD.Dec}(\text{key}, N', C') \]
(Non-) Committing AEAD

\[ M \leftarrow \text{AEAD.Dec}(\text{Nonce } N', \text{Ciphertext } C') \]

\[ M^* \leftarrow \text{AEAD.Dec}(\text{Nonce } N', \text{Ciphertext } C') \]
(Non-) Committing AEAD

Nonce $N'$
Ciphertext $C'$

$M \leftarrow \text{AEAD.Dec}(\ldots, N', C')$

$M^* \leftarrow \text{AEAD.Dec}(\ldots, N', C')$

$N' \parallel C'$

$M \leftarrow \text{AEAD.Dec}(\ldots, N', C')$
(Non-) Committing AEAD

Nonce $N'$
Ciphertext $C'$

$N' \parallel C'$

$M \leftarrow \text{AEAD.Dec}(\text{key}, N', C')$

$M^* \leftarrow \text{AEAD.Dec}(\text{key}, N', C')$

No guarantee the sender actually knows the exact key the recipient will use to decrypt!

Not considered an essential security goal, except in moderation settings [GLR CRYPTO’17], [DGRW CRYPTO’18]
Partitioning Oracle Attack

Password dictionary
D

password1
password2
password3
password4
password5
password6
password7
password8
Partitioning Oracle Attack

Password dictionary

D

password1
password2
password3
password4
password5
password6
password7
password8

password1
password2
password3
password4
password5
password6
password7
password8
Partitioning Oracle Attack

Password dictionary \( D \)

\[
\text{Password} \\
\text{password1} \\
\text{password2} \\
\text{password3} \\
\text{password4} \\
\text{password5} \\
\text{password6} \\
\text{password7} \\
\text{password8}
\]

Nonce \( N \)

Ciphertext \( C \)

splitting ciphertext \( k = 4 \)
Partitioning Oracle Attack

$\leftarrow$ AEAD.Dec("password5", N, C)
Partitioning Oracle Attack

\[ N \parallel C \]

\[ \leftarrow \text{AEAD.Dec("password5", N, C)} \]

Decryption error

Password dictionary

D

password1
password2
password3
password4
password5
password6
password7
password8

Nonce N

Ciphertext C

splitting ciphertext \( k = 4 \)
Partitioning Oracle Attack

Password dictionary $D$

Nonce $N$ || Ciphertext $C$

$\leftarrow$ AEAD.Dec("password5", $N$, $C$)

Decryption error

splitting ciphertext $k = 4$
Partitioning Oracle Attack

\[ k = \frac{|D|}{2}, \quad \frac{|D|}{4}, \quad \frac{|D|}{8}, \quad \frac{|D|}{16}, \quad \frac{|D|}{32} \]

Requires \( \Theta(\log |D|) \) queries to learn the password

Exponential speedup over brute-force dictionary attack!
Partitioning Oracle Attack

Requires $O(\log |D|)$ queries to learn the password

Exponential speedup over brute-force dictionary attack!

$|D|$ is large so a more realistic case is $k = 5000$

This still offers a good speedup over brute-force
Partitioning oracle attacks rely on:

1. Building splitting ciphertexts that can decrypt under $k > 1$ different keys

2. Access to a partitioning oracle
Partitioning oracle attacks rely on:

1. Building splitting ciphertexts that can decrypt under \( k > 1 \) different keys

   **Key Multi-collision Attacks**

   [GLR CRYPTO’17] first showed an attack against AES-GCM for \( k = 2 \)

2. Access to a partitioning oracle
Computing Key Multi-Collisions for AES-GCM

Encrypt then MAC

Counter mode encryption of AES

GHASH: polynomial MAC

Run time: $\mathcal{O}(k^2)$

Ciphertext that decrypts under all $k$ keys

Length: $k$ 16-byte blocks
Computing Key Multi-Collisions for AES-GCM

Encrypt → Counter mode encryption of AES

- then -

MAC

GHASH: polynomial MAC

Run time: $\mathcal{O}(k^2)$

Reduces finding ciphertext to solving set of linear equations

Ciphertext that decrypts under all $k$ keys

Length: $k$ 16-byte blocks
Computing Key Multi-Collisions for AES-GCM

Input: Let nonce N, authentication tag T, and keys $K_1$, $K_2$, $K_3$ be arbitrary.

Goal: Compute ciphertext $C$ that decrypts under all 3 keys.

Pre-compute: $H_i = AES_{K_i}(0^{128})$, $P_i = AES_{K_i}(N || 0^{311})$, $L = |C|$

\[
H_1^4 \cdot C_1 \oplus H_1^3 \cdot C_2 \oplus H_1^2 \cdot C_3 \oplus H_1 \cdot L \oplus P_1 = T
\]

\[
H_2^4 \cdot C_1 \oplus H_2^3 \cdot C_2 \oplus H_2^2 \cdot C_3 \oplus H_2 \cdot L \oplus P_2 = T
\]

\[
H_3^4 \cdot C_1 \oplus H_3^3 \cdot C_2 \oplus H_3^2 \cdot C_3 \oplus H_3 \cdot L \oplus P_3 = T
\]
Computing Key Multi-Collisions for AES-GCM

**Input:** Let nonce N, authentication tag T, and keys K₁, K₂, K₃ be arbitrary

**Goal:** Compute ciphertext C that decrypts under all 3 keys

**Pre-compute:** $H_i = \text{AES}_K(0^{128})$, $P_i = \text{AES}_K(N \parallel 0^{311})$, $L = |C|$

$$\begin{bmatrix}
H_1^2 & H_1 & 1 \\
H_2^2 & H_2 & 1 \\
H_3^2 & H_3 & 1
\end{bmatrix}
\begin{bmatrix}
C_1 \\
C_2 \\
C_3
\end{bmatrix}
= 
\begin{bmatrix}
(T \oplus H_1 \cdot L \oplus P_1) \cdot H_1^{-2} \\
(T \oplus H_2 \cdot L \oplus P_2) \cdot H_2^{-2} \\
(T \oplus H_3 \cdot L \oplus P_3) \cdot H_3^{-2}
\end{bmatrix}$$

Vandermonde matrix: we can use polynomial interpolation!
Computing Key Multi-Collisions for AES-GCM

- Implemented Multi-Collide-GCM using SageMath and Magma computational algebra system
- Timing experiments performed on desktop with Intel Core i9 processor and 128 GB RAM, running Linux x86-64

<table>
<thead>
<tr>
<th>$k$</th>
<th>Time (s)</th>
<th>Size (B)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$2$</td>
<td>0.18</td>
<td>48</td>
</tr>
<tr>
<td>$2^{10}$</td>
<td>6.6</td>
<td>16,400</td>
</tr>
<tr>
<td>$2^{12}$</td>
<td>29</td>
<td>65,552</td>
</tr>
<tr>
<td>$2^{16}$</td>
<td>1,820</td>
<td>1,048,592</td>
</tr>
</tbody>
</table>

We make a ciphertext that decrypts under > 4000 keys in < 30 seconds!
Computing Key Multi-Collisions for AES-GCM

- Implemented Multi-Collide-GCM using SageMath and Magma computational algebra system
- Timing experiments performed on desktop with Intel Core i9 processor and 128 GB RAM, running Linux x86-64

<table>
<thead>
<tr>
<th>$k$</th>
<th>Time (s)</th>
<th>Size (B)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.18</td>
<td>48</td>
</tr>
<tr>
<td>$2^{10}$</td>
<td>6.6</td>
<td>16,400</td>
</tr>
<tr>
<td>$2^{12}$</td>
<td>29</td>
<td>65,552</td>
</tr>
<tr>
<td>$2^{16}$</td>
<td>1,820</td>
<td>1,048,592</td>
</tr>
</tbody>
</table>

There exists an algorithm that does polynomial interpolation in $O(k \log^2 k)$ using FFTs, so it’s possible to create multi-collisions much faster [BM ’74]

We make a ciphertext that decrypts under > 4000 keys in < 30 seconds!
Key Multi-Collisions for Other AEAD Schemes

- XSalsa20/Poly1305
- ChaCha20/Poly1305
- AES-GCM-SIV

⇒

Also vulnerable to key multi-collision attacks!

Attacks are more complex and less scalable than those for AES-GCM
Partitioning oracle attacks rely on:

1. Building splitting ciphertexts that can decrypt under $k > 1$ different keys

   **Key Multi-collision Attacks**
   
   [GLR CRYPTO’17] first showed an attack against AES-GCM for $k = 2$

2. Access to a partitioning oracle
Partitioning oracle attacks rely on:

1. Building splitting ciphertexts that can decrypt under $k > 1$ different keys
   
   **Key Multi-collision Attacks**
   
   [GLR CRYPTO’17] first showed an attack against AES-GCM for $k = 2$

2. Access to a partitioning oracle

   Where do partitioning oracles arise?
Partitioning Oracles

Schemes we looked at in depth

- Shadowsocks proxy servers for UDP
- Early implementations of the OPAQUE asymmetric PAKE protocol

Possible partitioning oracles

- Hybrid encryption: Hybrid Public-Key Encryption (HPKE)
- Age file encryption tool
- Kerberos drafts (not adopted)
- JavaScript Object Signing and Encryption (JOSE)
- Anonymity systems: use partitioning oracles to learn which public key a recipient is using from a set of public keys
What do we do?

- Our paper is the latest in a growing body of evidence that non-committing AEAD can be dangerous*

- So which committing AEAD scheme do we use?
  - None currently standardized!

We need a committing AEAD standard, and it should be the default choice for AEAD

* After we published our results, [ADGKLS ‘20] also discussed the importance of committing AEAD
Conclusion

Read the paper: https://eprint.iacr.org/2020/1491.pdf

- Described partitioning oracle attacks, which exploit non-committing AEAD to recover secrets
- Widely-used AEAD schemes, such as AES-GCM, XSalsa20/Poly1305, ChaCha20/Poly1305, and AES-GCM-SIV, are not committing
- Partitioning oracle attacks can be used to recover passwords from Shadowsocks proxy servers and incorrect implementations of OPAQUE

**Recommendation**: Design and standardize committing AEAD for deployment

Thank you to my co-authors and Hugo Krawczyk, Mihir Bellare, Scott Fluhrer, David McGrew, Kenny Patterson, Chris Wood, Steven Bellovin, and Samuel Neves!
References


