Key-committing AEAD

Julia Len

Cornell Tech
Authenticated Encryption

Plaintext $M$

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

For simplicity, we ignore nonces and associated data in this presentation.
Authenticated Encryption

Plaintext $M$

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

$C$

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

For simplicity, we ignore nonces and associated data in this presentation.
Authenticated Encryption

Plaintext $M$

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

$C$

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

Popular

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

Easy to use

- Efficient
- Standardized
- Widely supported

Secure

- Proven CCA-secure
- Confidentiality
- Integrity

For simplicity, we ignore nonces and associated data in this presentation.
Authenticated Encryption

Plaintext $M$

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

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

Popular

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

Secure

• Proven CCA-secure
• Confidentiality
• Integrity

Efficient

• Standardized
• Widely supported

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

For simplicity, we ignore nonces and associated data in this presentation
(Non-) Committing AEAD

Ciphertext $C'$

$$M_1 \leftarrow \text{AEAD.Dec}(\text{key}, C')$$

$$M_2 \leftarrow \text{AEAD.Dec}(\text{key}, C')$$
Partitioning Oracle Attacks [LGR USENIX’21]

Ciphertext $C'$

\[ M_1 \leftarrow \text{AEAD.Dec}(\text{ }, C') \]

\[ M_2 \leftarrow \text{AEAD.Dec}(\text{ }, C') \]

\[ M_3 \leftarrow \text{AEAD.Dec}(\text{ }, C') \]

\[ M_4 \leftarrow \text{AEAD.Dec}(\text{ }, C') \]
Partitioning Oracle Attacks [LGR USENIX’21]

Ciphertext $C'$

- $M_1 \leftarrow \text{AEAD.Dec}(\cdot, C')$
- $M_2 \leftarrow \text{AEAD.Dec}(\cdot, C')$
- $M_3 \leftarrow \text{AEAD.Dec}(\cdot, C')$
- $M_4 \leftarrow \text{AEAD.Dec}(\cdot, C')$

$M_1 \leftarrow \text{AEAD.Dec}(\cdot, C')$
Partitioning Oracle Attacks [LGR USENIX’21]

Ciphertext $C'$

$M_1 \leftarrow \text{AEAD.Dec}(\ldots, C')$

$M_2 \leftarrow \text{AEAD.Dec}(\ldots, C')$

$M_3 \leftarrow \text{AEAD.Dec}(\ldots, C')$

$M_4 \leftarrow \text{AEAD.Dec}(\ldots, C')$

$C' \quad \rightarrow \quad S\text{uccess!} \quad \leftarrow M_1 \leftarrow \text{AEAD.Dec}(\ldots, C')$
Partitioning Oracle Attacks

The attacker learns 1-bit of information about the key!
Vulnerabilities from non-committing AEAD (so far)

Content moderation

[GLR CRYPTO'17]
[DGRW CRYPTO'18]

- Facebook Messenger
- Message franking

- 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

Services by Google & Amazon

[ADGKLS ‘20]

- Key rotation in key management services
- Envelope encryption in the AWS encryption SDK
- Subscribe with Google

Partitioning oracle attacks
What do we use for key-committing AEAD?
What do we use for key-committing AEAD?

- None currently standardized!
What do we use for key-committing AEAD?

- None currently standardized!

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Description</th>
<th>Adopted by…</th>
<th>Extra overhead over base scheme</th>
<th>Potential issues?</th>
</tr>
</thead>
</table>
| Zeros Block Check       | Modifies AEAD scheme to check that a block of recovered plaintext is all-zeros string | Libsodium    | Adds **64 bytes** to each ciphertext | • Side-channel if implemented incorrectly  
• Need to implement and analyze separately for each AEAD scheme |
| Hash Key Check          | Modifies AEAD scheme to check SHA256 hash of the key during decryption       | AWS Encryption SDK | Adds at least **32 bytes** to each ciphertext | Side-channel if implemented incorrectly |
| Single-key Encrypt-then-HMAC | Plain Encrypt-then-HMAC using single key                              | -            | None!                           | Less efficient |

14
## What do we use for key-committing AEAD?

- None currently standardized!
- As we begin the process of making an internet-draft, we would love to hear your thoughts about needs and requirements

| Scheme                  | Description                                                                 | Adopted by… | Extra overhead over base scheme | Potential issues?                                                                 |
|-------------------------|-----------------------------------------------------------------------------|--------------|---------------------------------|---------------------------------------------------------------------------------
| Zeros Block Check       | Modifies AEAD scheme to check that a block of recovered plaintext is all-zeros string | Libsodium    | Adds 64 bytes to each ciphertext | • Side-channel if implemented incorrectly  
                          |                                                                               |                           |                                 | • Need to implement and analyze separately for each AEAD scheme |
| Hash Key Check          | Modifies AEAD scheme to check SHA256 hash of the key during decryption        | AWS Encryption SDK | Adds at least 32 bytes to each ciphertext | Side-channel if implemented incorrectly                                           |
| Single-key Encrypt-then-HMAC | Plain Encrypt-then-HMAC using single key                                           | -             | None!                           | Less efficient                                                                 |

As we begin the process of making an internet-draft, we would love to hear your thoughts about needs and requirements.
References


