Fast Message Franking: From Invisible Salamanders to Encryptment

Yevgeniy Dodis, Paul Grubbs, Thomas Ristenpart, Joanne Woodage
End-to-end encrypted messaging

End-to-end security: Provider cannot read or modify messages

[Frosch et al. 2014]
[Cohn-Gordon et al. 2016]
[Cohn-Gordon, Cremers, Garratt 2016]
[Bellare et al. 2017]
[Jaeger and Stepanovs 2018]
Providers want to help users with abuse

Authenticated Encryption

Service provider

He said !%$#!

End-to-end security: cannot verify “!%$#!” was sent

[Facebook 2016]:

• Provide cryptographic proof of message contents when reporting abuse
• Called technique message franking
Our contributions

Show vulnerability in Facebook’s scheme: *invisible salamanders*

Lower bound on efficiency of ccAE

New symmetric-key primitive: *encryptment.*
Hash-Function-Chaining (HFC):
single-pass encryption construction

Generic, fast transform:
encryptment + compression function=ccAE
Facebook’s message franking protocol

Sender \textit{cryptographically commits} to message: \[ C_B = \text{HMAC}(K_B, M) \]

\textbf{Encrypt-then-HMAC} message along with \( K_B \) (called the opening)

Provider signs \( C_B \) using HMAC to generate tag \( T_{FB} \) (fast because \( C_B \) short)

Receiver decrypts, retrieves \( K_B \), and verifies \( C_B \)
Facebook’s message franking protocol

To report abuse, send message as well as $K_B, C_B, T_{FB}$

Provider can verify $C_B, T_{FB}$, convinced that message was “!%$#!”

Attachments (images, videos) handled differently

Is Facebook’s approach secure?

[GLR17]: without attachments, yes
This work: with attachments, no!
Security goals for message franking

1) **Receiver binding**: receiver can’t open a message not sent
2) **Sender binding**: can’t send a message that can’t be reported
3) End-to-end confidentiality/authenticity for messages not reported
Facebook’s attachment franking protocol

Sender *cryptographically commits* to attachment encryption key:

\[ C_B = \text{HMAC}(K_B, K_{\text{file}}) \]

*Encrypt-then-HMAC* file encryption key \( K_{\text{file}} \) along with \( K_B \)

*AES-GCM* encrypt attachment: \( \text{AES-GCM}(K_{\text{file}}, \text{file}) \)

Receiver decrypts as before to get \( K_{\text{file}} \) and then decrypts attachment.
To report abuse, receiver opens $K_{file}$ and other recent messages. Facebook checks openings & decrypts all unique AES-GCM ciphertexts to add them to abuse report.
Our attack exploits AES-GCM

1. Craft special **AES-GCM** ciphertext:
   - Decrypts under $K_{file}$ to innocuous image
   - Decrypts under $K_{file2}$ to abuse image

2. Send ciphertext twice - $K_{file}, K_{file2}$

3. receiver sees both

4. Only the innocuous image appears in report! (Violates sender binding)
Our attack exploits AES-GCM

Craft special **AES-GCM** ciphertext:
1) Decrypts under $K_{file}$ to innocuous image
2) Decrypts under $K_{file2}$ to abuse image

But isn’t **AES-GCM** a secure authenticated encryption scheme?

Yes, but ... this type of attack is not standard attacker gets to choose $K_{file}$ and $K_{file2}$

GCM uses a universal-hash-based MAC **not collision resistant (CR)**

Our attack violates **robustness**: can find ciphertext that decrypts under two keys (First robustness attack against real system)

[Abdalla, Bellare, Neven 2010] [Farshim et al. 2013] [Farshim et al. 2017]
Abusive JPEG seen by receiver, but not in abuse report

Innocuous BMP in abuse report

Disclosed to Facebook

Thanks to Jon Millican for answering questions!

They fixed by changing report generation logic

Awarded us a bug bounty
Recall Facebook’s message franking

 Commitment + authenticated encryption (AE): [GLR] proved secure as ccAE

 Didn’t use for attachments because too slow

• Signal uses AES-CBC then HMAC for AE
• Total of 3 passes (HMAC-Encrypt-HMAC)

Can we make faster ccAE schemes?
How do we build faster ccAE?

Ideally: ~1 blockcipher call per msg block.

*Can any secure scheme achieve this?*

**No!**

**Thm. Secure ccAE => CR hashing.**

Leverage prior impossibility results for CR hashing from fixed-key blockciphers

- [Black, Cochran, Shrimpton 2005]
- [Rogaway, Steinberger 2008]

No similar ccAE scheme can be secure!

<table>
<thead>
<tr>
<th>Scheme</th>
<th>ccAE?</th>
<th># passes</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES-GCM</td>
<td>No</td>
<td>1</td>
</tr>
<tr>
<td>OCB</td>
<td>No</td>
<td>1</td>
</tr>
<tr>
<td>Encrypt-then-HMAC (distinct keys)</td>
<td>No</td>
<td>2</td>
</tr>
<tr>
<td>Encrypt-then-HMAC (one key)</td>
<td>Yes</td>
<td>2</td>
</tr>
<tr>
<td>Facebook HMAC-Encrypt-HMAC</td>
<td>Yes</td>
<td>3</td>
</tr>
</tbody>
</table>
How do we build faster ccAE?

**Step 1**
- New primitive: *encryptment* “one-time” ccAE
- Hash-Function-Chaining (HFC) scheme

**Step 2**
- Simple transforms from encryption to ccAE
- Encryption-to-ccAE transform from compression function

ccAE in one SHA-256 call
Encryptment: syntax, semantics, security

$EC(K, M) = C_1, C_B$  \hspace{1cm} encrypts and commits to $M$

$DO(K, C_1, C_B) = M/\bot$  \hspace{1cm} decrypts $(C_1, C_B)$ and opens to $M$

$EVer(M, K, C_B) = 0/1$  \hspace{1cm} verifies commitment $C_B$ of $M$

1. **Confidentiality:** can’t distinguish ciphertexts from random bits
2. **Second-ciphertext unforgeability:** can’t forge ciphertexts in particular way
3. **Receiver binding:** can’t generate $K,M$ pairs that verify for same $C_B$
4. **Sender binding:** can’t decrypt ciphertext that doesn’t verify properly
The hash-function chaining (HFC) scheme

Recall Merkle-Damgard style hash functions (e.g., SHA-256) built in two steps:

1) Specify a compression function $f: \{0,1\}^n \times \{0,1\}^d \rightarrow \{0,1\}^n$
2) Iterate $f$ to hash long message (after some suitable padding)

![Diagram of HFC scheme]

Constant bit string called initialization vector
The hash-function chaining (HFC) scheme

The HFC scheme $\text{EC}(K, M)$:
1) Prepend message with a block of zeros, XOR key into each block
2) Use chaining variables as encryption pad to compute $C_1$
3) MD output is the binding tag $C_B$
The hash-function chaining (HFC) scheme

The HFC scheme $\text{EC}(K, M)$:

1) Prepend message with a block of zeros, XOR key into each block
2) Use chaining variables as encryption pad to compute $C_1$
3) MD output is the binding tag $C_B$

```
IV → f → K ⊕ M_1 → f → K ⊕ M_2 → f → K ⊕ M_3 → f → F(M)
```
The hash-function chaining (HFC) scheme

The HFC scheme \( EC(K, M) \):
1) Prepend message with a block of zeros, XOR key into each block
2) Use chaining variables as encryption pad to compute \( C_1 \)
3) MD output is the binding tag \( C_B \)
The hash-function chaining (HFC) scheme

The HFC scheme $\mathbf{EC}(K, M)$:
1) Prepend message with a block of zeros, XOR key into each block
2) Use chaining variables as encryption pad to compute $C_1$
3) MD output is the binding tag $C_B$

$\mathbf{DO}(K, C_1, C_B)$ runs MD, recovers message blocks, checks $C_B$
$\mathbf{EVer}(K, M, C_B)$ recomputes, checks $C_B$

Similar to AE from [Cogliani et al. ‘10]
[Bertoni et al. ‘11]

EC/DO/EVer require just one pass of hash function
(Fast) Encryption => (Fast) ccAE

Construct fast ccAE from fast encryption: 2 additional compression function calls

1. Use long-term key $K_{lt}$
2. Derive encryption key via $f$
3. MAC the binding tag $C_B$
(Fast) Encryption $\Rightarrow$ (Fast) ccAE

Construct fast ccAE from fast encryption: 2 additional compression function calls.

**Thm.** If $\textbf{EC}$ is a secure encryption scheme and compression function is PRF, this construction is ccAE.

Encryption is useful elsewhere, gives single-pass:
- concealments [DH03]
- remotely-keyed AE [BFN98]
- robust AE [FOR17]

See paper for details.
Conclusion

Show vulnerability in Facebook’s scheme: *invisible salamanders*

Lower bound on efficiency of ccAE

New symmetric-key primitive: *encryptment.*
Hash-Function-Chaining (HFC):
single-pass encryption construction

Generic, fast transform:
encryptment + compression function=ccAE

Thanks for listening! Any questions?
Security of HFC

**Theorem (informal):**
HFC is a secure encryption scheme

See paper for details!

Constant bit string called initialization vector
CTR mode encryption with AES blockcipher $E$

Universal hash-based message authentication (called GMAC)

Can rewrite GMAC as:

$$Tag = C_1^*H_3 + C_2^*H_2 + \text{len}^*H + \text{Pad}$$

1) Pick key $K_{file}$, derive $H1$, Pad1
2) Pick block of plaintext
3) Let $C_1$ be ciphertext block using $K_{file}$
4) Pick key $K_{file2}$, derive $H2$, Pad2
5) Solve $Tag$ equation for $C_2$:

$$Tag = C_1^*H_1^3 + C_2^*H_1^2 + \text{len}*H1 + \text{Pad1}$$
$$= C_1^*H_2^3 + C_2^*H_2^2 + \text{len}*H2 + \text{Pad2}$$

6) Output $K_{file}$, $K_{file2}$, $C_1$, $C_2$, $Tag$
Our contributions

Show vulnerability in Facebook’s scheme: *invisible salamanders*

Introduce new symmetric-key primitive: *encryptment*

Lower bound on efficiency of encryptment

Construct optimally-efficient encryptment: gives fastest-known ccAEAD, robust encryption, remotely-keyed AE, etc.
Our contributions

Show vulnerability in Facebook’s scheme: invisible salamanders
Introduce new symmetric-key primitive: encryptment

Lower bound on efficiency of encryptment

Construct optimally-efficient encryptment: gives fastest-known ccAEAD, robust encryption, remotely-keyed AE, etc.
Our contributions

Show vulnerability in Facebook’s scheme: 
*invisible salamanders*

Introduce new symmetric-key primitive: 
*encryptment*

Lower bound on efficiency of encryptment

Construct optimally-efficient encryptment: 
gives fastest-known ccAEAD, robust encryption, remotely-keyed AE, etc.
Our contributions

Show vulnerability in Facebook’s scheme:
  invisible salamanders

Introduce new symmetric-key primitive:
  encryption

Lower bound on efficiency of encryption

Construct optimally-efficient encryption:
  gives fastest-known ccAEAD, robust encryption, remotely-keyed AE, etc.
To report abuse, receiver opens $K_{file}$ and other recent messages.

Facebook checks openings & decrypts all unique AES-GCM ciphertexts to add them to abuse report.
Our attack exploits AES-GCM

Craft special AES-GCM ciphertext:
1) Decrypts under $K_{file}$ to innocuous image
2) Decrypts under $K_{file2}$ to abuse image

Adversary can use to violate sender binding:
  i. Craft special ciphertext and keys
  ii. Send ciphertext twice as distinct encrypted attachments
  iii. Victim sees both plaintext attachments
  iv. Abuse report will omit first (chosen) attachment
How do we build faster ccAE?

Define new primitive: *encryptment* simpler than ccAE

Generic, efficient transforms from encryptment to ccAE

Introduce Hash-Function-Chaining (HFC): optimally-efficient encryptment

+ Encryption-to-ccAE transform from fixed-length AE (others too, see paper)

Fastest-possible ccAE!
Encryptment:
syntax, semantics, security

\[ EC(K, H, M) = C_1, C_B \]
\[ DO(K, H, C_1, C_B) = M/ \]
\[ EVer(H, M, K, C_B) = 0/1 \]

- **Encrypts** \( M \) and **commits** to \((H, M)\)
- **Decrypts** \((C_1, C_B)\) and **opens** to \( M \)
- **Verifies** commitment \( C_B \) of \((H,M)\)

**Confidentiality**
One-time real-or-random (otROR): cannot distinguish between EC oracle and random bits oracle

**Integrity**
Second ctxt unforgeability (SCU): cannot forge new ciphertext for fixed \( K, C_B \)

**Binding**
Strong receiver binding (srBIND): cannot verify two \((H, M, K)\) tuples with same \( C_B \)
Sender binding as in [GLR]
Construct ccAEAD from encryption with same performance profile

\[
\text{ccAEAD-Enc}(K, H, M):
\]
\[
K_{EC} \leftarrow \$ \text{ECKeyGen}()
\]
\[
C_1, B_{EC} \leftarrow \text{EC}(K_{EC}, H, M)
\]
\[
C_2 \leftarrow \$ \text{AEAD-Enc}(K, B_{EC}, K_{EC})
\]
Return \( C_1, B_{EC} || C_2 \)

Encryption is “core” primitive for other interesting applications:
- concealments [DH03]
- remotely-keyed AE [BFN98]
- robust AE [FOR17]

See paper for details
Encryptment $\implies$ ccAEAD

Construct ccAEAD from encryption with same performance profile

\[ \text{ccAEAD-Enc}(K, H, M): \]
\[ K_{\text{EC}} \leftarrow \$ \text{ECKeyGen}() \]
\[ C_1, B_{\text{EC}} \leftarrow \text{EC}(K_{\text{EC}}, H, M) \]
\[ C_2 \leftarrow \$ \text{AEAD-Enc}(K, B_{\text{EC}}, K_{\text{EC}}) \]
Return $C_1, B_{\text{EC}} \bowtie C_2$

Use a fixed-input-length AEAD scheme with header $B_{\text{EC}}$ to encrypt $K_{\text{EC}}$

**Theorem (informal):** If EC is a secure encryption scheme and AEAD is secure AE scheme, this construction is ccAE