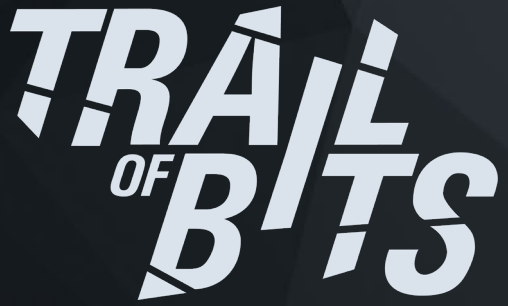
TRAIL
OF
BITS

Scott Arciszewski @ Trail of Bits

AES-GEM

Galois Extended Mode

Why propose a cipher mode?

- AES-GCM can only be safely used to encrypt 2^{32} messages per key
  - Assuming nonces are generated randomly, from a 96-bit space
  - Probability of collision >= 2^{-32} when # messages >= 2^{32}
- Cloud providers can reach this safety limit several times per minute
- Reusing a nonce leaks the authentication key
- Weaknesses with truncated tags
  - Successful forgeries leak the authentication key
- AES-GCM is not key-committing
- etc.

Why *not* invent a new cipher mode?

- AES-GCM is widely supported
- AES-GCM's security is well-understood
- New designs are risky

Is there middle ground between the two?

Extend AES-GCM's design and security assumptions to a tweaked mode that is secure in the following respects (assuming 256-bit keys):

1. Can safely encrypt up to 2^{112} messages before exceeding 2^{-32} risk
2. Supports individual messages up to 2^{61} bytes long, rather than $2^{36}-256$
3. Fixes the security risk with short tags
4. Adds key-commitment to the AEAD interface
5. Does all this with only AES block encryptions and XOR

Galois Extended Mode (GEM)

(First draft)

What is Galois Extended Mode?

- Start with the design of AES-GCM
- Extend the nonce, use most of it for subkey derivation
- Encrypt the data with the subkey, rather than the input key
- After GHASH, but before the final XOR, add an AES block encryption
- Introduce key commitment
- Use 64-bit internal counters, rather than 32-bit
- GEM comes in two variants: 128-bit and 256-bit keys

Subkey Derivation

- Use AES-CBC-MAC with the first {128, 192} bits of the {192, 256} nonce, with domain separation constants, to derive {128, 256} bits
- XOR the input key with the output of the CBC-MACs
  - See also: the Salsa20 design and security proof
- There should be 64 bits of nonce left over in either case
- Performance?
  - AES-128-GEM: Equivalent to 2 calls to AES_ecb_encrypt() plus a few XORs
  - AES-256-GEM: Equivalent to 4 calls to AES_ecb_encrypt() plus a few XORs
  - Plus setting up a second key schedule for the derived key in each mode

Actual encryption of bulk data

AES-128-GEM

- Encrypted with AES-128-CTR

AES-256-GEM

- Encrypted with AES-256-CTR

Starting internal counter is 0, rather than 2.

Actual encryption of bulk data (2)

Final block encoding is represented in bits, not bytes.

Due to the maximum length of $2^{61} - 1$ bytes, there are counter values that can never be reached.

Therefore, we reserve these internal counter values.

Actual encryption of bulk data (3)

Authentication key, H, is the encryption of -1 rather than 0:

- 0xFFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF

Final encryption counter block J0 is the leading 64 bits of nonce followed by a counter value of -2:

- [nonce bytes] 0xFFFFFFFF FFFFFFFE

Authentication Tag Improvement

We introduce a suggestion from Niels Ferguson in 2005:

Encrypt the GHASH output, using the input key (not the subkey)

In ECB mode, to introduce a nonlinear transformation of the GHASH bits

- XOR some random bits is addition (mod 2) and therefore linear

Key commitment?

Reserve two more plaintext counter values (-3, -4).

Encrypt the two blocks with AES -ECB.

Append these encrypted blocks to the authentication tag in the AEAD interface.

On decrypt: Recompute and compare (in constant -time!)

What is Galois Extended Mode? (Review)

- Start with the design of AES-GCM
- Extend the nonce, use most of it for subkey derivation
- Encrypt the data with the subkey, rather than the input key
- After GHASH, but before the final XOR, add an AES block encryption
- Introduce key commitment
- Use 64-bit internal counters, rather than 32-bit
- GEM comes in two variants: 128-bit and 256-bit keys

What is Galois Extended Mode? (Review)

- Start with the design of AES-GCM
- Extend the nonce, use most of it for subkey derivation
- Encrypt the data with the subkey, rather than the input key
- After GHASH, but before the final XOR, add an AES block encryption
- Introduce key commitment
- Use 64-bit internal counters, rather than 32 -bit ???
- GEM comes in two variants: 128-bit and 256-bit keys

Polishing AES-GEM

Food for thought

Is a 64-bit internal counter safe?

AES is a PRP, not a PRF.

There are no collisions for different inputs under the same key.

Therefore, an attacker gains an advantage in distinguishing long messages from random bytes based on the absence of block collisions or repetition.

All is not lost!

There are many ways to adjust the design to mitigate this risk.

We would like to hear your thoughts before we commit to one tactic.

But first, let me share my approach.

Hierarchical Subkeys

Add one more layer of indirection for bulk data encryption.

The top 29 bits of the internal counter is used to select a new subkey, which is used for the next 2^{32} blocks (or 2^{36} bytes) of data.

This adds a few more AES block encryptions, plus an additional AES key schedule every 2^{32} blocks.

Hierarchical Subkeys (2)

Subsubkey derivation:

- We reserve some more internal counter space
- AES-128-GEM: Encrypt one AES block to get the 128-bit subsubkey for the next 2^{32} blocks
- AES-256-GEM: Encrypt two AES blocks to get the 256-bit subsubkey for the next 2^{32} blocks

Now, every 2^{36} bytes, you're encrypting under a different AES key

Other approaches to consider

- Wide PRPs (e.g., XTS mode)
- 256-bit block ciphers
- Even-Mansour
- Possibly others?

Recap

GEM extends GCM without discarding or changing its foundations

Emphasis on subkey derivation that amortizes well for long messages

Mitigate known weaknesses with AES - GCM with short tags

Emphasis on conservative security decisions

- Security assumptions: AES, GCM, and XOR

Technical details are in the paper, and on the Trail of Bits blog

Thank you!

https://blog.trailofbits.com

scott.arciszewski@trailofbits.com