# AES-GCM-SIV

CFRG, IETF 97

## Dramatis Personae

Designers/cryptographers:

**Shay Gueron** (University of Haifa & Intel)
**Yehuda Lindell** (Bar Ilan University)

Tech writer:

**Adam Langley** (Google) ← That’s me.

## AES-GCM-SIV

The context for AES-GCM-SIV is that:

1. Current AEADs (AES-GCM, ChaCha20-Poly1305) work great for transport encryption where using counters for the nonce is simple and safe. Because nonce duplication is devastating, they work much less well in other cases where, perhaps, many encryptors need to work independently with the same key.
2. Hardware support for AES and GHASH is becoming increasingly common. It can be found in (at least) recent x86-64, ARMv8, S/390 and POWER chips.

We wish to be able to reuse existing hardware abilities in an AEAD that better supports non-transport encryption cases.

## Nonce-misuse resistance

The property that we want, that AES-GCM and ChaCha20-Poly1305 don’t give us, is *nonce-misuse resistance*. That means that we can publish encryptions of $\text{msg}_1$ and $\text{msg}_2$ with the same key *and nonce* for $\text{msg}_1 \neq \text{msg}_2$ and not suffer disastrous failures of confidentiality and authenticity.

With that, it’s possible to generate nonces randomly and remain safe.

In contrast: with AES-GCM’s 96-bit nonce, NIST allows no more than $2^{32}$ encryptions per key with random nonces. Böck et al showed in a recent paper that, even in TLS, implementations managed to use a fixed nonce or a random nonce with AES-GCM. Nonce mistakes sadly do happen.

## AES-GCM-SIV

AES-GCM-SIV is an AEAD that:

1. Provides nonce-misuse resistance.
2. Reuses the building blocks of AES-GCM and thus can take advantage of existing hardware support for AES-GCM. Thus decrypts at close to the speed of AES-GCM.
3. Publicly available code: reference, optimised asm, MacOS asm, C intrinsics. All at https://github.com/Shay-Gueron/AES-GCM-SIV.

The first version was published in Gueron & Lindell, CCS 2015. The current version is specified in draft-irtf-cfrg-gcmsiv. An updated version of the original paper that reflects the current design is soon to be published.

## High-level overview

GHASH (from AES-GCM) is replaced with POLYVAL.

GHASH operates over $GF(2^{128})[x] / P(x)$ where $P(x)$ is $x^{128} + x^7 + x^2 + x + 1$, but with the order of the bits in each byte reversed.

POLYVAL operates over $GF(2^{128})[x] / Q(x)$ where $Q(x)$ is $x^{128} + x^{127} + x^{126} + x^{121} + 1$, but with the more obvious bit order. (Note that $Q(x) = P(x)$ with the bits reversed.)

## High-level overview

Rather than encrypt with a counter based on the nonce (which makes duplicate nonces violate confidentiality), AES-GCM-SIV first authenticates the plaintext and derives the counter from that. Thus different messages start at different counters, even for the same key and nonce.

AES-GCM masks (i.e., encrypts) the output of GHASH by adding the encryption of the nonce, which can be canceled out if two messages are authenticated with the same key.

AES-GCM-SIV encrypts the output of POLYVAL instead.

## Key schedule

Based on feedback from the working group, the key schedule now uses a cascade:

```
record-authentication-key = AES128(key = key-generating-key,
                                   input = nonce)
record-encryption-key = AES128(key = key-generating-key,
                               input = record-authentication-key)
```

(For AES-128-GCM-SIV. The 256-bit version is similar.)

We have a version that gets better performance on small messages and better bounds by having a 12**6**-bit nonce and encrypting nonce‖00, nonce‖01, … However, since we expect larger messages to be the typical use-case for this, and because 126 bits is awkward, we are not currently planning on moving forward with this change.

## Speed

All numbers are in cycles/byte for an Intel Skylake core.

| **Mode** | **1KB** | **2KB** | **4KB** | **8KB** | **16KB** |
|---|---|---|---|---|---|
| 128-bit Enc | 1.32 | 1.12 | 1.02 | 0.98 | 0.95 |
| 128-bit Dec | 1.09 | 0.85 | 0.74 | 0.68 | 0.66 |
| 256-bit Enc | 1.75 | 1.46 | 1.32 | 1.25 | 1.22 |
| 256-bit Dec | 1.36 | 1.10 | 1.00 | 0.94 | 0.91 |

Plain AES-128-GCM runs at 0.65 cycles/byte on that chip.

## Next steps

1. Publish updated security analysis.
2. Implement in BoringSSL and get real-world experience.
3. Publish RFC.