Workgroup: Crypto Forum
Internet-Draft:
draft-mattsson-cfrg-aes-gcm-sst-03
Published: 16 March 2024
Intended Status: Informational
Expires: 17 September 2024
Authors: M. Campagna
Amazon Web Services
A. Maximov
Ericsson
J. Preuß Mattsson
Ericsson

Galois Counter Mode with Secure Short Tags (GCM-SST)

Abstract

This document defines the Galois Counter Mode with Secure Short Tags (GCM-SST) Authenticated Encryption with Associated Data (AEAD) algorithm. GCM-SST can be used with any keystream generator, not just a block cipher. The main differences compared to GCM [GCM] is that GCM-SST uses an additional subkey Q, that fresh subkeys H and Q are derived for each nonce, and that the POLYVAL function from AES-GCM-SIV is used instead of GHASH. This enables short tags with forgery probabilities close to ideal. This document also registers several instances of Advanced Encryption Standard (AES) with Galois Counter Mode with Secure Short Tags (AES-GCM-SST).

This document is the product of the Crypto Forum Research Group.

About This Document

This note is to be removed before publishing as an RFC.

The latest revision of this draft can be found at https://emanjon.github.io/draft-mattsson-cfrg-aes-gcm-sst/draft-mattsson-cfrg-aes-gcm-sst.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-mattsson-cfrg-aes-gcm-sst/.

Discussion of this document takes place on the Crypto Forum Research Group mailing list (mailto:cfrg@ietf.org), which is archived at https://mailarchive.ietf.org/arch/search/?email_list=cfrg. Subscribe at https://www.ietf.org/mailman/listinfo/cfrg/.

Source for this draft and an issue tracker can be found at https://github.com/emanjon/draft-mattsson-cfrg-aes-gcm-sst.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 17 September 2024.

Copyright Notice

Table of Contents

1. Introduction

Advanced Encryption Standard (AES) in Galois Counter Mode (AES-GCM) [GCM] is a widely used AEAD algorithm [RFC5116] due to its attractive performance in both software and hardware as well as its provable security. During the NIST standardization, Ferguson pointed out two weaknesses in the GCM authentication function [Ferguson]. The weaknesses are especially concerning when GCM is used with short tags. The first weakness significantly increases the probability of successful forgery. The second weakness reveals the subkey H if the attacker manages to create successful forgeries. With knowledge of the subkey H, the attacker always succeeds with subsequent forgeries. The probability of multiple successful forgeries is therefore significantly increased.

As a comment to NIST, Nyberg et al. [Nyberg] explained how small changes based on proven theoretical constructions mitigate these weaknesses. Unfortunately, NIST did not follow the advice from Nyberg et al. and instead specified additional requirements for use with short tags in Appendix C of [GCM]. NIST did not give any motivations for the specific choice of parameters, or for that matter the security levels they were assumed to give. As shown by Mattsson et al. [Mattsson], an attacker can almost always gain feedback on success or failure of forgery attempts, contradicting NIST's assumptions for short tags. NIST also appears to have used non-optimal attacks to calculate the parameters. A detailed evaluation of GCM and other block cipher modes of operation is given by [Rogaway]. Rogaway is critical of GCM with short tags and recommends disallowing GCM with tags shorter than 96-bits. NIST is planning to remove support for GCM with tags shorter than 96-bits [Revise]. While Counter with CBC-MAC (CCM) [RFC5116] with short tags has forgery probabilities close to ideal, CCM has lower performance than GCM.

32-bit tags are standard in most radio link layers including 5G, 64-bit tags are very common in transport and application layers of the Internet of Things, and 32-, 64-, and 80-bit tags are common in media-encryption applications. Audio packets are small, numerous, and ephemeral, so on the one hand, they are very sensitive in percentage terms to crypto overhead, and on the other hand, forgery

of individual packets is not a big concern. Due to its weaknesses, GCM is typically not used with short tags. The result is either decreased performance from larger than needed tags [MoQ], or decreased performance from using much slower constructions such as AES-CTR combined with HMAC [RFC3711][I-D.ietf-sframe-enc]. Short tags are also useful to protect packets transporting a signed payload such as a firmware update.

This document defines the Galois Counter Mode with Secure Short Tags (GCM-SST) Authenticated Encryption with Associated Data (AEAD) algorithm following the recommendations from Nyberg et al. [Nyberg]. GCM-SST is defined with a general interface so that it can be used with any keystream generator, not just a 128-bit block cipher. The main differences compared to GCM [GCM] is that GCM-SST uses an additional subkey Q, that fresh subkeys H and Q are derived for each nonce, and that the POLYVAL function from AES-GCM-SIV [RFC8452] is used instead of GHASH, see Section 3. This enables short tags with forgery probability close to ideal and significantly decreases the probability of multiple successful forgeries, see Section 5. The performance of GCM-SST is very similar to GCM [GCM]. The two additional AES invocations are compensated by the use of POLYVAL, the ”little-endian version” of GHASH, which is faster on little-endian architectures. GCM-SST maintains the additive encryption characteristic of GCM, which enables efficient implementations on modern processor architectures, see [Gueron] and Section 2.4 of [GCM-Update]. This document also registers several instances of Advanced Encryption Standard (AES) with Galois Counter Mode with Secure Short Tags (AES-GCM-SST) where AES [AES] in counter mode is used as the keystream generator. See Section 4. GCM-SST has been standardized for use with AES-256 and SNOW 5G [SNOW] in 3GPP 5G Advance.

2. Conventions and Definitions

The key words "**MUST**", "**MUST NOT**", "**REQUIRED**", "**SHALL**", "**SHALL NOT**", "**SHOULD**", "**SHOULD NOT**", "**RECOMMENDED**", "**NOT RECOMMENDED**", "**MAY**", and "**OPTIONAL**" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

Primitives:

*K is the key as defined in [RFC5116]

*N is the nonce as defined in [RFC5116]

*A is the associated data as defined in [RFC5116]

*P is the plaintext as defined in [RFC5116]

*= is the assignment operator

*!= is the inequality operator

*x || y is concatenation of the octet strings x and y

*XOR is the bitwise exclusive OR operator

*len(x) is the length of x in bits.

*zeropad(x) right pads an octet string x with zeroes to a multiple of 128 bits

*truncate(x, t) is the truncation operation. The first t bits of x are kept

*n is the number of 128-bit chunks in zeropad(P)

*m is the number of 128-bit chunks in zeropad(A)

*POLYVAL is defined in [RFC8452]

*BE32(x) is the big-endian encoding of 32-bit integer x

*LE64(x) is the little-endian encoding of 64-bit integer x

*V[y] is the 128-bit chunk with index y in the array V; the first chunk has index 0.

*V[x:y] are the range of chunks x to y in the array V

3. Galois Counter Mode with Secure Short Tags (GCM-SST)

This section defines the Galois Counter Mode with Secure Short Tags (GCM-SST) AEAD algorithm following the recommendations from Nyberg et al. [Nyberg]. GCM-SST is defined with a general interface so that it can be used with any keystream generator, not just a 128-bit block cipher.

GCM-SST adheres to an AEAD interface [RFC5116] and the encryption function takes four variable-length octet string parameters. A secret key K, a nonce N, the associated data A, and a plaintext P. The keystream generator is instantiated with K and N. The keystream **MAY** depend on P and A. The minimum and maximum lengths of all parameters depend on the keystream generator. The keystream generator produces a keystream Z consisting of 128-bit chunks where the first three chunks Z[0], Z[1], and Z[2] are used as the three subkeys H, Q, and M. The following keystream chunks Z[3], Z[4], ..., Z[n + 2] are used to encrypt the plaintext. Instead of GHASH [GCM], GCM-SST makes use of the POLYVAL function from AES-GCM-SIV

[RFC8452], which results in more efficient software implementations on little-endian architectures. GHASH and POLYVAL can be defined in terms of one another [RFC8452]. The subkeys H and Q are field elements used in POLYVAL while the subkey M is used for the final masking of the tag. Both encryption and decryption are only defined on inputs that are a whole number of octets.

Figures illustrating the GCM-SST encryption and decryption functions are shown in [SST1][SST2].

3.1. Authenticated Encryption Function

Encrypt(K, N, A, P)

The encryption function encrypts a plaintext and returns the ciphertext along with an authentication tag that verifies the authenticity of the plaintext and associated data, if provided.

Prerequisites and security:

- *The key **MUST** be randomly chosen from a uniform distribution.
- *For a given key, the nonce **MUST NOT** be reused under any circumstances.
- *Supported tag_length associated with the key.
- *Definitions of supported input-output lengths.

Inputs:

- *Key K (variable-length octet string)
- *Nonce N (variable-length octet string)
- *Associated data A (variable-length octet string)
- *Plaintext P (variable-length octet string)

Outputs:

- *Ciphertext ct (variable-length octet string)
- *Tag tag (octet string with length tag_length)

Steps:

1. If the lengths of K, N, A, P are not supported return error and abort
2. Initiate keystream generator with K and N

3. Let H = Z[0], Q = Z[1], M = Z[2]

4. Let ct = P XOR truncate(Z[3:n + 2], len(P))

5. Let S = zeropad(A) || zeropad(ct)

6. Let L = LE64(len(ct)) || LE64(len(A))

7. Let X = POLYVAL(H, S[0], S[1], ...)

8. Let full_tag = POLYVAL(Q, X XOR L) XOR M

9. Let tag = truncate(full_tag, tag_length)

10. Return (ct, tag)

3.2. Authenticated Decryption Function

Decrypt(K, N, A, ct, tag)

The decryption function decrypts a ciphertext, verifies that the authentication tag is correct, and returns the plaintext on success or an error if tag verification failed.

Prerequisites and security:

- The calculation of the plaintext P (step 10) **MAY** be done in parallel with the tag verification (step 3-9). If tag verification fails, the plaintext P and the expected_tag **MUST NOT** be given as output.

- The comparison of the input tag with the expected_tag **MUST** be done in constant time.

- Supported tag_length associated with the key.

- Definitions of supported input-output lengths.

Inputs:

- Key K (variable-length octet string)

- Nonce N (variable-length octet string)

- Associated data A (variable-length octet string)

- Ciphertext ct (variable-length octet string)

- Tag tag (octet string with length tag_length)

Outputs:

* Plaintext P (variable-length octet string) or an error indicating that the authentication tag is invalid for the given inputs.

Steps:

1. If the lengths of K, N, A, or ct are not supported, or if len(tag) != tag_length return error and abort

2. Initiate keystream generator with K and N

3. Let H = Z[0], Q = Z[1], M = Z[2]

4. Let S = zeropad(A) || zeropad(ct)

5. Let L = LE64(len(ct)) || LE64(len(A))

6. Let X = POLYVAL(H, S[0], S[1], ...)

7. Let full_tag = POLYVAL(Q, X XOR L) XOR M

8. Let expected_tag = truncate(full_tag, tag_length)

9. If tag != expected_tag, return error and abort

10. Let P = ct XOR truncate(Z[3:n + 2], len(ct))

11. Return P

3.3. Encoding (ct, tag) Tuples

Applications **MAY** keep the ciphertext and the authentication tag in distinct structures or encode both as a single octet string C. In the latter case, the tag **MUST** immediately follow the ciphertext ct:

C = ct || tag

4. AES with Galois Counter Mode with Secure Short Tags

This section defines Advanced Encryption Standard (AES) with Galois Counter Mode with Secure Short Tags (AES-GCM-SST). When GCM-SSM is instantiated with AES, the keystream generator is AES in counter mode

Z[i] = AES-ENC(K, N || BE32(i))

where AES-ENC is the AES encrypt function [AES].

4.1. AES-GCM-SST AEAD Instances

We define six AEAD instances, in the format of [RFC5116], that use AES-GCM-SST. They differ only in key length (K_LEN) and tag length. The tag lengths 32, 64, and 80 have been chosen to align with secure media frames [I-D.ietf-sframe-enc]. The key length and tag length are related to different security properties, and an application encrypting audio packets with small tags might require 256-bit confidentiality.

Numeric ID	Name	K_LEN (bytes)	tag_length (bits)
TBD1	AEAD_AES_128_GCM_SST_4	16	32
TBD2	AEAD_AES_128_GCM_SST_8	16	64
TBD3	AEAD_AES_128_GCM_SST_10	16	80
TBD4	AEAD_AES_256_GCM_SST_4	32	32
TBD5	AEAD_AES_256_GCM_SST_8	32	64
TBD6	AEAD_AES_256_GCM_SST_10	32	80

Table 1: AEAD Algorithms

Common parameters for the six AEAD instances:

- *P_MAX (maximum size of the plaintext) is 2^{36} - 48 octets.
- *A_MAX (maximum size of the associated data) is 2^{36} octets.
- *N_MIN and N_MAX (minimum and maximum size of the nonce) are both 12 octets
- *C_MAX (maximum size of the ciphertext and tag) is P_MAX + tag_length (in bytes)

5. Security Considerations

GCM-SST uses an additional subkey Q and that new subkeys H, Q are derived for each nonce. The use of an additional subkey Q enables short tags with forgery probabilities close to ideal. Deriving new subkeys H, Q for each nonce significantly decreases the probability of multiple successful forgeries. These changes are based on proven theoretical constructions and follows the recommendations in [Nyberg]. See [Nyberg] for details and references to security proofs for the construction.

GCM-SST **MUST** be used in a nonce-respecting setting: for a given key, a nonce **MUST** only be used once. The nonce **MAY** be public or predictable. It can be a counter, the output of a permutation, or a generator with a long period. Every key **MUST** be randomly chosen from a uniform distribution. Implementations **SHOULD** randomize the nonce by mixing a unique number like a sequence number with a per-key

random salt. This improves security against pre-computation attacks and multi-key attacks [Bellare].

The GCM-SST tag_length **SHOULD NOT** be smaller than 4 bytes and cannot be larger than 16 bytes. For short tags of length t < 128 - log2(n + m + 1) bits, the worst-case forgery probability is bounded by $\approx 2^{-t}$ [Nyberg]. With the constraints listed in Section 4.1, n + m + 1 < 2^{33} 128-bit blocks, and tags of length up to 95 bits therefore have an almost perfect security level. This is significantly better than GCM where the security level is only t – log2(n + m + 1) bits [GCM]. As one can note, for 128-bit tags and long messages, the forgery probability is not close to ideal and similar to GCM [GCM]. If tag verification fails, the plaintext and expected_tag **MUST NOT** be given as output. The full_tag in GCM-SST does not depend on the tag length. An application can make the tag dependent on the tag length by including tag_length in the nonces.

The confidentiality offered by AES-GCM-SST against passive attackers is equal to AES-GCM [GCM] and given by the birthday bound. The maximum size of the plaintext (P_MAX) has been adjusted from GCM [RFC5116] as there is now three subkeys instead of two.

For the AES-GCM-SST algorithms in Table 1 the worst-case forgery probability is bounded by $\approx 2^{-t}$ where t is the tag length in bits [Nyberg]. This is true for all allowed plaintext and associated data lengths. The maximum size of the associated data (A_MAX) has been lowered from GCM [RFC5116] to enable forgery probability close to ideal for 80-bit tags even with maximum size plaintexts and associated data. Just like [RFC5116] AES-GCM-SST only allows 96-bit nonces.

If r random nonces are used with the same key, the collision probability for AES-GCM-SST is $\approx r^2 / 2^{97}$. As an attacker can test r nonces for collisions with complexity r, the security of AES-GCM-SST with random nonces is only $\approx 2^{97} / r$. It is therefore **NOT RECOMMENDED** to use AES-GCM-SST with random nonces.

In general, there is a very small possibility in GCM-SST that either or both of the subkeys H and Q are zero, so called weak keys. If both keys are zero, the resulting tag will not depend on the message. There are no obvious ways to detect this condition for an attacker, and the specification admits this possibility in favor of complicating the flow with additional checks and regeneration of values. In AES-GCM-SST, H and Q are generated with the AES-ENC permutation on different input, so H and Q cannot both be zero.

6. IANA Considerations

IANA is requested to assign the entries in the first two columns of Table 1 to the "AEAD Algorithms" registry under the "Authenticated Encryption with Associated Data (AEAD) Parameters" heading with this document as reference.

7. References

7.1. Normative References

[AES] "ADVANCED ENCRYPTION STANDARD (AES)", NIST Federal Information Processing Standards Publication 197, November 2001, <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197-upd1.pdf>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/rfc/rfc2119>.

[RFC5116] McGrew, D., "An Interface and Algorithms for Authenticated Encryption", RFC 5116, DOI 10.17487/RFC5116, January 2008, <https://www.rfc-editor.org/rfc/rfc5116>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/rfc/rfc8174>.

[RFC8452] Gueron, S., Langley, A., and Y. Lindell, "AES-GCM-SIV: Nonce Misuse-Resistant Authenticated Encryption", RFC 8452, DOI 10.17487/RFC8452, April 2019, <https://www.rfc-editor.org/rfc/rfc8452>.

7.2. Informative References

[Bellare] Bellare, M. and B. Tackmann, "The Multi-User Security of Authenticated Encryption: AES-GCM in TLS 1.3", November 2017, <https://eprint.iacr.org/2016/564.pdf>.

[Ferguson] Ferguson, N., "Authentication weaknesses in GCM", May 2005, <https://csrc.nist.gov/CSRC/media/Projects/Block-Cipher-Techniques/documents/BCM/Comments/CWC-GCM/Ferguson2.pdf>.

[GCM] Dworkin, M., "Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC", NIST Special Publication 800-38D, November 2007, <https://

nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38d.pdf>.

[GCM-Update] McGrew, D. and J. Viega, "GCM Update", May 2005, <https://csrc.nist.gov/csrc/media/projects/block-cipher-techniques/documents/bcm/comments/cwc-gcm/gcm-update.pdf>.

[Gueron] Gueron, S., "Constructions based on the AES Round and Polynomial Multiplication that are Efficient on Modern Processor Architectures", October 2023, <https://csrc.nist.gov/csrc/media/Presentations/2023/constructions-based-on-the-aes-round/images-media/sess-5-gueron-bcm-workshop-2023.pdf>.

[I-D.ietf-sframe-enc] Omara, E., Uberti, J., Murillo, S. G., Barnes, R., and Y. Fablet, "Secure Frame (SFrame)", Work in Progress, Internet-Draft, draft-ietf-sframe-enc-07, 29 February 2024, <https://datatracker.ietf.org/doc/html/draft-ietf-sframe-enc-07>.

[I-D.irtf-cfrg-aegis-aead] Denis, F. and S. Lucas, "The AEGIS Family of Authenticated Encryption Algorithms", Work in Progress, Internet-Draft, draft-irtf-cfrg-aegis-aead-10, 20 January 2024, <https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-aegis-aead-10>.

[Mattsson] Mattsson, J. and M. Westerlund, "Authentication Key Recovery on Galois/Counter Mode (GCM)", May 2015, <https://eprint.iacr.org/2015/477.pdf>.

[MoQ] IETF, "Media Over QUIC", September 2022, <https://datatracker.ietf.org/wg/moq/about/>.

[Nyberg] Nyberg, K., Gilbert, H., and M. Robshaw, "Galois MAC with forgery probability close to ideal", June 2005, <https://csrc.nist.gov/CSRC/media/Projects/Block-Cipher-Techniques/documents/BCM/Comments/general-comments/papers/Nyberg_Gilbert_and_Robshaw.pdf>.

[Revise] NIST, "Announcement of Proposal to Revise SP 800-38D", August 2023, <https://csrc.nist.gov/news/2023/proposal-to-revise-sp-800-38d>.

[RFC3711] Baugher, M., McGrew, D., Naslund, M., Carrara, E., and K. Norrman, "The Secure Real-time Transport Protocol

(SRTP)", RFC 3711, DOI 10.17487/RFC3711, March 2004, <https://www.rfc-editor.org/rfc/rfc3711>.

[Rogaway] Rogaway, P., "Evaluation of Some Blockcipher Modes of Operation", February 2011, <https://www.cryptrec.go.jp/exreport/cryptrec-ex-2012-2010r1.pdf>.

[SNOW] Ekdahl, P., Johansson, T., Maximov, A., and J. Yang, "SNOW-Vi: an extreme performance variant of SNOW-V for lower grade CPUs", March 2021, <https://eprint.iacr.org/2021/236>.

[SST1] Campagna, M., Maximov, A., and J. Preuß Mattsson, "Galois Counter Mode with Secure Short Tags (GCM-SST)", October 2023, <https://csrc.nist.gov/csrc/media/Events/2023/third-workshop-on-block-cipher-modes-of-operation/documents/accepted-papers/Galois%20Counter%20Mode%20with%20Secure%20Short%20Tags.pdf>.

[SST2] Campagna, M., Maximov, A., and J. Preuß Mattsson, "Galois Counter Mode with Secure Short Tags (GCM-SST)", October 2023, <https://csrc.nist.gov/csrc/media/Presentations/2023/galois-counter-mode-with-secure-short-tags/images-media/sess-5-mattsson-bcm-workshop-2023.pdf>.

Appendix A. AES-GCM-SST Test Vectors

A.1. AES-GCM-SST Test #1 (128-bit key)

```
       KEY = { 00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f }
     NONCE = { 30 31 32 33 34 35 36 37 38 39 3a 3b }
         H = { 22 ce 92 da cb 50 77 4b ab 0d 18 29 3d 6e ae 7f }
         Q = { 03 13 63 96 74 be fa 86 4d fa fb 80 36 b7 a0 3c }
         M = { 9b 1d 49 ea 42 b0 0a ec b0 bc eb 8d d0 ef c2 b9 }
```

Case #1a

```
       AAD = { }
 PLAINTEXT = { }
encode-LEN = { 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 }
  full-TAG = { 9b 1d 49 ea 42 b0 0a ec b0 bc eb 8d d0 ef c2 b9 }
       TAG = { 9b 1d 49 ea }
CIPHERTEXT = { }
```

Case #1b

```
       AAD = { 40 41 42 43 44 }
 PLAINTEXT = { }
encode-LEN = { 00 00 00 00 00 00 00 00 28 00 00 00 00 00 00 00 }
  full-TAG = { 7f f3 cb a4 d5 f3 08 a5 70 4e 2f d5 f2 3a e8 f9 }
       TAG = { 7f f3 cb a4 }
CIPHERTEXT = { }
```

Case #1c

```
       AAD = { }
 PLAINTEXT = { 60 61 62 63 64 65 66 67 68 69 6a 6b }
encode-LEN = { 60 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 }
  full-TAG = { f8 de 17 85 fd 1a 90 d9 81 8f cb 7b 44 69 8a 8b }
       TAG = { f8 de 17 85 }
CIPHERTEXT = { 64 f0 5b ae 1e d2 40 3a 71 25 5e dd }
```

Case #1d

```
       AAD = { 40 41 42 43 44 45 46 47 48 49 4a 4b 4c 4d 4e 4f }
 PLAINTEXT = { 60 61 62 63 64 65 66 67 68 69 6a 6b 6c 6d 6e 6f
               70 71 72 73 74 75 76 77 78 79 7a 7b 7c 7d 7e }
encode-LEN = { f8 00 00 00 00 00 00 00 80 00 00 00 00 00 00 00 }
  full-TAG = { 93 43 56 14 0b 84 48 2c d0 14 c7 40 7e e9 cc b6 }
       TAG = { 93 43 56 14 }
CIPHERTEXT = { 64 f0 5b ae 1e d2 40 3a 71 25 5e dd 53 49 5c e1
               7d c0 cb c7 85 a7 a9 20 db 42 28 ff 63 32 10 }
```

Case #1e

```
       AAD = { 40 41 42 43 44 45 46 47 48 49 4a 4b 4c 4d 4e }
 PLAINTEXT = { 60 61 62 63 64 65 66 67 68 69 6a 6b 6c 6d 6e 6f
               70 }
encode-LEN = { 88 00 00 00 00 00 00 00 78 00 00 00 00 00 00 00 }
  full-TAG = { f8 50 b7 97 11 43 ab e9 31 5a d7 eb 3b 0a 16 81 }
       TAG = { f8 50 b7 97 }
CIPHERTEXT = { 64 f0 5b ae 1e d2 40 3a 71 25 5e dd 53 49 5c e1
               7d }
```

A.2. AES-GCM-SST Test #2 (128-bit key)

```
       KEY = { 29 23 be 84 e1 6c d6 ae 52 90 49 f1 f1 bb e9 eb }
     NONCE = { 9a 50 ee 40 78 36 fd 12 49 32 f6 9e }
       AAD = { 1f 03 5a 7d 09 38 25 1f 5d d4 cb fc 96 f5 45 3b
               13 0d }
 PLAINTEXT = { ad 4f 14 f2 44 40 66 d0 6b c4 30 b7 32 3b a1 22
               f6 22 91 9d }
         H = { 2d 6d 7f 1c 52 a7 a0 6b f2 bc bd 23 75 47 03 88 }
         Q = { 3b fd 00 96 25 84 2a 86 65 71 a4 66 e5 62 05 92 }
         M = { 9e 6c 98 3e e0 6c 1a ab c8 99 b7 8d 57 32 0a f5 }
encode-LEN = { a0 00 00 00 00 00 00 00 90 00 00 00 00 00 00 00 }
  full-TAG = { 45 03 bf b0 96 82 39 b3 67 e9 70 c3 83 c5 10 6f }
       TAG = { 45 03 bf b0 96 82 39 b3 }
CIPHERTEXT = { b8 65 d5 16 07 83 11 73 21 f5 6c b0 75 45 16 b3
               da 9d b8 09 }
```

A.3. AES-GCM-SST Test #3 (256-bit key)

```
       KEY = { 00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f
               10 11 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f }
     NONCE = { 30 31 32 33 34 35 36 37 38 39 3a 3b }
         H = { 3b d9 9f 8d 38 f0 2e a1 80 96 a4 b0 b1 d9 3b 1b }
         Q = { af 7f 54 00 16 aa b8 bc 91 56 d9 d1 83 59 cc e5 }
         M = { b3 35 31 c0 e9 6f 4a 03 2a 33 8e ec 12 99 3e 68 }
```

Case #3a

```
       AAD = { }
 PLAINTEXT = { }
encode-LEN = { 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 }
  full-TAG = { b3 35 31 c0 e9 6f 4a 03 2a 33 8e ec 12 99 3e 68 }
       TAG = { b3 35 31 c0 e9 6f 4a 03 }
CIPHERTEXT = { }
```

Case #3b

```
       AAD = { 40 41 42 43 44 }
 PLAINTEXT = { }
encode-LEN = { 00 00 00 00 00 00 00 00 28 00 00 00 00 00 00 00 }
  full-TAG = { 63 ac ca 4d 20 9f b3 90 28 ff c3 17 04 01 67 61 }
       TAG = { 63 ac ca 4d 20 9f b3 90 }
CIPHERTEXT = { }
```

Case #3c

```
       AAD = { }
 PLAINTEXT = { 60 61 62 63 64 65 66 67 68 69 6a 6b }
encode-LEN = { 60 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 }
  full-TAG = { e1 de bf fd 5f 3a 85 e3 48 bd 6f cc 6e 62 10 90 }
       TAG = { e1 de bf fd 5f 3a 85 e3 }
CIPHERTEXT = { fc 46 2d 34 a7 5b 22 62 4f d7 3b 27 }
```

Case #3d

```
       AAD = { 40 41 42 43 44 45 46 47 48 49 4a 4b 4c 4d 4e 4f }
 PLAINTEXT = { 60 61 62 63 64 65 66 67 68 69 6a 6b 6c 6d 6e 6f
               70 71 72 73 74 75 76 77 78 79 7a 7b 7c 7d 7e }
encode-LEN = { f8 00 00 00 00 00 00 00 80 00 00 00 00 00 00 00 }
  full-TAG = { c3 5e d7 83 9f 21 f7 bb a5 a8 a2 8e 1f 49 ed 04 }
       TAG = { c3 5e d7 83 9f 21 f7 bb }
CIPHERTEXT = { fc 46 2d 34 a7 5b 22 62 4f d7 3b 27 84 de 10 51
               33 11 7e 17 58 b5 ed d0 d6 5d 68 32 06 bb ad }
```

Case #3e

```
       AAD = { 40 41 42 43 44 45 46 47 48 49 4a 4b 4c 4d 4e }
 PLAINTEXT = { 60 61 62 63 64 65 66 67 68 69 6a 6b 6c 6d 6e 6f
               70 }
encode-LEN = { 88 00 00 00 00 00 00 00 78 00 00 00 00 00 00 00 }
  full-TAG = { 49 7c 14 77 67 a5 3d 57 64 ce fd 03 26 fe e7 b5 }
       TAG = { 49 7c 14 77 67 a5 3d 57 }
CIPHERTEXT = { fc 46 2d 34 a7 5b 22 62 4f d7 3b 27 84 de 10 51
               33 }
```

A.4. AES-GCM-SST Test #4 (256-bit key)

```
       KEY = { 29 23 be 84 e1 6c d6 ae 52 90 49 f1 f1 bb e9 eb
               b3 a6 db 3c 87 0c 3e 99 24 5e 0d 1c 06 b7 b3 12 }
     NONCE = { 9a 50 ee 40 78 36 fd 12 49 32 f6 9e }
       AAD = { 1f 03 5a 7d 09 38 25 1f 5d d4 cb fc 96 f5 45 3b
               13 0d }
 PLAINTEXT = { ad 4f 14 f2 44 40 66 d0 6b c4 30 b7 32 3b a1 22
               f6 22 91 9d }
         H = { 13 53 4b f7 8a 91 38 fd f5 41 65 7f c2 39 55 23 }
         Q = { 32 69 75 a3 3a ff ae ac af a8 fb d1 bd 62 66 95 }
         M = { 59 48 44 80 b6 cd 59 06 69 27 5e 7d 81 4a d1 74 }
encode-LEN = { a0 00 00 00 00 00 00 00 90 00 00 00 00 00 00 00 }
  full-TAG = { c4 a1 ca 9a 38 c6 73 af bf 9c 73 49 bf 3c d5 4d }
       TAG = { c4 a1 ca 9a 38 c6 73 af bf 9c }
CIPHERTEXT = { b5 c2 a4 07 f3 3e 99 88 de c1 2f 10 64 7b 3d 4f
               eb 8f f7 cc }
```

Change Log

This section is to be removed before publishing as an RFC.

Changes from -02 to -03:

* Added performance information and considerations.

* Editorial changes.

Changes from -01 to -02:

* The length encoding chunk is now called L

* Use of the notation POLYVAL(H, X_1, X_2, ...) from RFC 8452

* Removed duplicated text in security considerations.

Changes from -00 to -01:

* Link to NIST decision to remove support for GCM with tags shorter than 96-bits based on Mattsson et al.

* Mention that 3GPP 5G Advance will use GCM-SST with AES-256 and SNOW 5G.

* Corrected reference to step numbers during decryption

* Changed T to full_tag to align with tag and expected_tag

* Link to images from the NIST encryption workshop illustrating the GCM-SST encryption and decryption functions.

* Updated definitions

* Editorial changes.

Acknowledgments

The authors thank Richard Barnes, Scott Fluhrer, and Eric Lagergren for their valuable comments and feedback. Some of the formatting and text were inspired by and borrowed from [I-D.irtf-cfrg-aegis-aead].

Authors' Addresses

Matthew Campagna
Amazon Web Services
Canada

Email: campagna@amazon.com

Alexander Maximov
Ericsson AB
Sweden

Email: alexander.maximov@ericsson.com

John Preuß Mattsson
Ericsson AB
Sweden

Email: john.mattsson@ericsson.com