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

Matthew Campagna[1], Alexander Maximov[2], John Preuß Mattsson[2]

[1] *Amazon Web Services*
[2] *Ericsson*

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. Thus GCM-SST is a mode of operation of the Advanced Encryption Standard (AES). The main differences compared to 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.

1. Introduction

Advanced Encryption Standard (AES) in Galois Counter Mode (AES-GCM) [1] is a widely used AEAD algorithm [2] 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 [3]. 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. [4] 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 [1]. 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. [5], 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 [6]. Rogaway is critical of GCM with short tags and recommends disallowing GCM with tags shorter than 96 bits. While Counter with CBC-MAC (CCM) [7] 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 [8], 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 [9], or decreased performance from using much slower constructions such as AES-CTR combined with HMAC [10–11]. 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. [4]. 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 is a mode of operation of the Advanced Encryption Standard (AES) [12]. The main differences compared to GCM [1] 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 [13] is used instead of GHASH. This enables short tags with forgery probability close to ideal and significantly decreases the probability of multiple successful forgeries.

IETF is currently discussion standardization of GCM-SST for use in media-encryption applications [14]. This document is in large parts identical with [14].

1.1. Notation

- K is the key as defined in [2]
- N is the nonce as defined in [2]
- A is the associated data as defined in [2]
- P is the plaintext as defined in [2]
- $=$ is the assignment operator
- $\neq$ is the inequality operator
- $x \,||\, y$ is concatenation of the octet strings x and y
- $\oplus$ 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 GCM-SIV [13]
- 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

2. 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. [4]. 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 is a mode of operation of the Advanced Encryption Standard (AES) [12].

GCM-SST adheres to an AEAD interface [2] 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 $Z[3], Z[4], \ldots, Z[n+2]$ are used to encrypt the plaintext. Instead of GHASH [1], GCM-SST makes use of the POLYVAL function from AES-GCM-SIV [13], which results in more efficient software implementations on little-endian architectures. GHASH and POLYVAL can be defined in terms of one another [13]. 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.

2.1. Authenticated Encryption Function

The encryption function Encrypt(K, N, A, P) encrypts a plaintext and returns the ciphertext ct along with an authentication tag tag that verifies the authenticity of the plaintext and the optional associated data. The authenticated encryption function is shown in Figure 1.

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, or 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 $ct = P \oplus \text{truncate}(Z[3{:}n+2], \text{len}(P))$
5. Let $S = \text{zeropad}(A) \,||\, \text{zeropad}(ct) \,||\, \text{LE64}(\text{len}(ct)) \,||\, \text{LE64}(\text{len}(A))$
6. Let $X = \text{POLYVAL}(H, S[0], S[1], \ldots, S[m+n-1])$
7. Let $full\_tag = \text{POLYVAL}(Q, X \oplus S[m+n]) \oplus M$
8. Let $tag = \text{truncate}(full\_tag, tag\_length)$
9. Return (ct, tag)

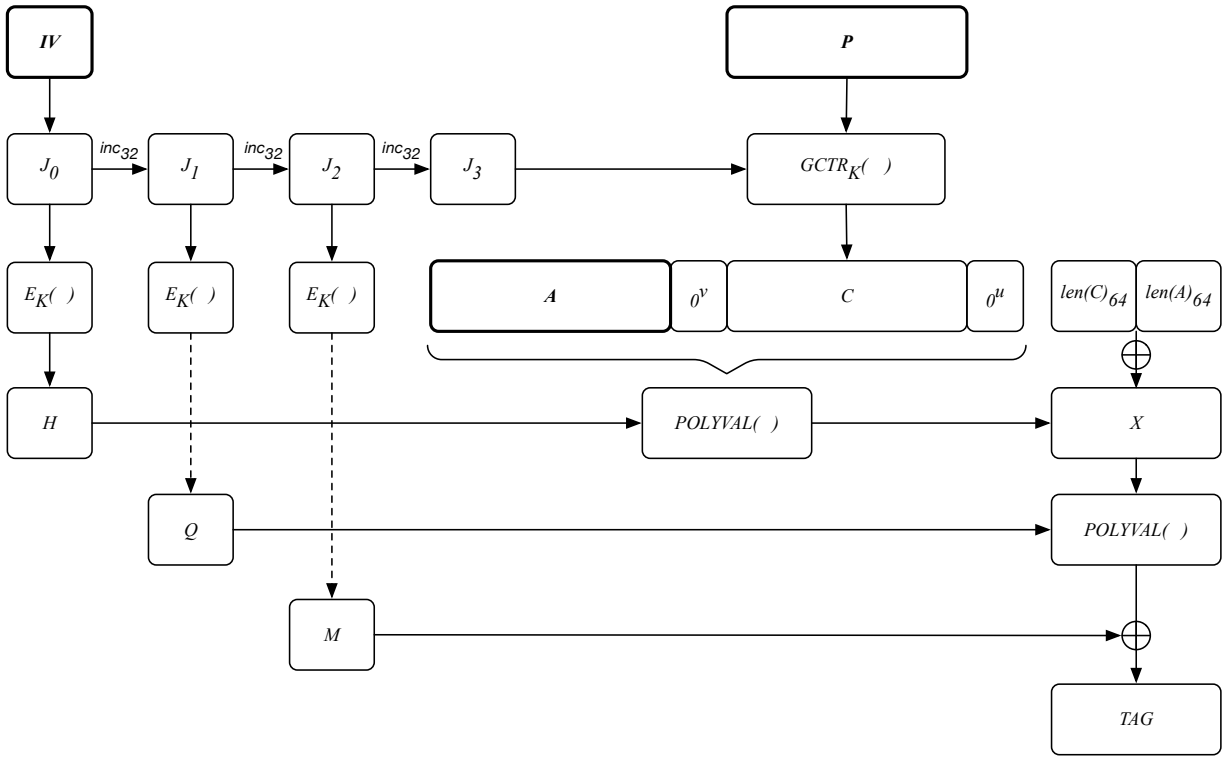

Figure 1: Authenticated encryption function

2.2. Authenticated Decryption Function

The decryption function Decrypt(K, N, A, ct, tag) decrypts a ciphertext, verifies that the authentication tag is correct, and returns the plaintext on success or an error if tag verification failed. The authenticated decryption function is shown in Figure 2.

Prerequisites and security:

- The calculation of the plaintext P (step 8) may be done in parallel with the tag verification (step 3-8). 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 $\text{len}(tag) \neq tag\_length$ 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 $S = \text{zeropad}(A) \,||\, \text{zeropad}(ct) \,||\, \text{LE64}(\text{len}(ct)) \,||\, \text{LE64}(\text{len}(A))$
5. Let $X = \text{POLYVAL}(H, S[0], S[1], \ldots, S[m + n - 1])$
6. Let $full\_tag = \text{POLYVAL}(Q, X \oplus S[m + n]) \oplus M$
7. Let $expected\_tag = \text{truncate}(full\_tag, tag\_length)$
8. If $tag \neq expected\_tag$, return error and abort
9. Let $P = ct \oplus \text{truncate}(Z[3{:}n + 2], \text{len}(ct))$
10. Return P

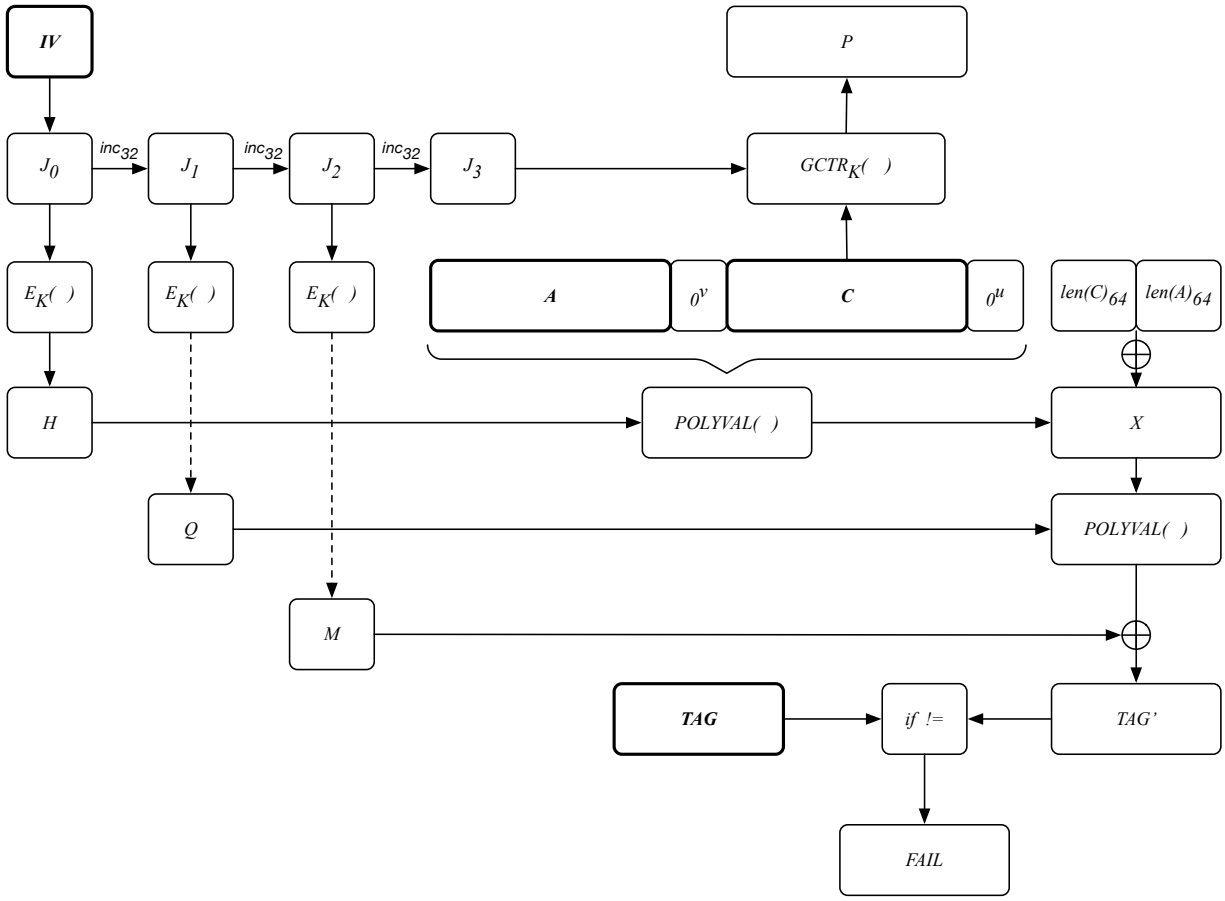

Figure 2: Authenticated decryption function

2.3. 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, tag must immediately follow the ciphertext ct

$$C = ct \,||\, tag\ .$$

3. 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

$$Z[i] = \text{AES-ENC}(K, N \,||\, \text{BE32}(i))\ ,$$

where AES-ENC is the AES encrypt function [12].

4. 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 [4]. See [4] 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 [15].

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 $\approx 2^{-t}$ [4]. This is significantly better than GCM where the security level is only $t - log2(n + m + 1)$ bits [1]. As one can note, for 128-bit tags and long messages, the forgery probability is not close to ideal and similar to GCM [1]. 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 [1] and given by the birthday bound.

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 the 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.

Just like the IETF specification of GCM [1] but unlike the NIST specification of GCM [1], AES-GCM-SST only allows 96-bit nonces.

4.1. Constraints

Following the terminology for AEAD algorithms defined in [2], we recommend the following constraints:

- 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)

The maximum size of the plaintext (P_MAX) has been adjusted from the IETF specification of GCM [2] as there is now three subkeys instead of two. The maximum size of the associated data (A_MAX) has been lowered from the IETF specification of GCM [2] to enable forgery probability close to ideal for larger tags even with maximum size plaintexts and associated data.

With the recommended constraints, $n + m + 1 < 2^{33}$ 128-bit blocks, and tags of length up to 95 bits therefore have an almost perfect security level for all allowed plaintext and associated data lengths, i.e., the worst-case forgery probability is bounded by $\approx 2^{-t}$ where t is the tag length in bits [4].

References

[1] NIST, SP 800-38D “Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC”
https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38d.pdf

[2] IETF, RFC 5116 “An Interface and Algorithms for Authenticated Encryption”
https://www.rfc-editor.org/rfc/rfc5116

[3] Ferguson, “Authentication weaknesses in GCM”
https://csrc.nist.gov/CSRC/media/Projects/Block-Cipher-Techniques/documents/BCM/Comments/CWC-GCM/Ferguson2.pdf

[4] Nyberg, Gilbert, Robshaw, “Galois MAC with forgery probability close to ideal”
https://csrc.nist.gov/CSRC/media/Projects/Block-Cipher-Techniques/documents/BCM/Comments/general-comments/papers/Nyberg_Gilbert_and_Robshaw.pdf

[5] Mattsson, Westerlund, “Authentication Key Recovery on Galois/Counter Mode (GCM)”
https://eprint.iacr.org/2015/477.pdf

[6] Rogaway, “Evaluation of Some Blockcipher Modes of Operation”
https://www.cryptrec.go.jp/exreport/cryptrec-ex-2012-2010r1.pdf

[7] NIST, SP 800-38C “Recommendation for Block Cipher Modes of Operation: the CCM Mode for Authentication and Confidentiality”
https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38c.pdf

[8] 3GPP TS 33.501, “Security architecture and procedures for 5G System”
https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3169

[9] IETF, “Media Over QUIC”
https://datatracker.ietf.org/wg/moq/about/

[10] IETF RFC 3711, “The Secure Real-time Transport Protocol (SRTP)”
https://www.rfc-editor.org/rfc/rfc3711

[11] IETF, “Secure Frame (SFrame)”
https://datatracker.ietf.org/doc/html/draft-ietf-sframe-enc

[12] NIST, FIPS 197 “Advanced Encryption Standard (AES)”
https://nvlpubs.nist.gov/nistpubs/fips/nist.fips.197.pdf

[13] IETF, RFC 8452 “AES-GCM-SIV: Nonce Misuse-Resistant Authenticated Encryption”
https://www.rfc-editor.org/rfc/rfc8452

[14] IETF “Galois Counter Mode with Secure Short Tags (GCM-SST)”
https://datatracker.ietf.org/doc/html/draft-mattsson-cfrg-aes-gcm-sst

[15] Bellare, “The Multi-User Security of Authenticated Encryption: AES-GCM in TLS 1.3”
https://eprint.iacr.org/2016/564.pdf

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 }
```