Crypto Forum
Internet-Draft
Intended status: Informational
Expires: 23 August 2025

M. Campagna
Amazon Web Services
A. Maximov
J. Preu Mattsson
Ericsson

19 February 2025

Galois Counter Mode with Strong Secure Tags (GCM-SST) draft-mattsson-cfrg-aes-gcm-sst-18

Abstract

This document defines the Galois Counter Mode with Strong Secure Tags (GCM-SST) Authenticated Encryption with Associated Data (AEAD) algorithm. GCM-SST can be used with any keystream generator, not just 128-bit block ciphers. The main differences from GCM are the use of an additional subkey H_2, the derivation of fresh subkeys H and H_2 for each nonce, and the replacement of the GHASH function with the POLYVAL function from AES-GCM-SIV. This enables truncated tags with near-ideal forgery probabilities, even against multiple forgery attacks, which are significant security improvements over GCM. GCM-SST is designed for security protocols with replay protection and addresses the strong industry demand for fast encryption with minimal overhead and high security. This document registers several instances of GCM-SST using Advanced Encryption Standard (AES) and Rijndael-256.

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

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 first weakness significantly increases the probability of successful forgery for long messages. The second weakness reveals the subkey H if an attacker succeeds in creating forgeries. Once H is known, the attacker can consistently forge subsequent messages, drastically increasing the probability of multiple successful forgeries. The two weaknesses are problematic for all tag lengths but are especially critical when short tags are used.

In 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 parameter choices or the assumed security levels. Mattsson et al. [Mattsson] later demonstrated that attackers can almost always obtain feedback on the success or failure of forgery attempts, contradicting the assumptions NIST made for short tags. Furthermore, NIST appears to have relied on non-optimal attacks when

calculating the parameters. Rogaway [Rogaway] criticizes the use of GCM with short tags and recommends prohibiting tags shorter than 96 bits. Reflecting the critique, NIST is planning to remove support for GCM with tags shorter than 96 bits [Revise]. NIST has not addressed the weaknesses for longer tags, such as the absence of reforgeability resistance [Reforge]. When AES-GCM is used in QUIC [RFC9001], the expected number of forgeries can be equivalent to that of an ideal MAC with a length of 64.4 bits. Reforgeability resistance is a key design criterion in modern AEAD algorithms such as [AEZ].

Short tags are widely used, 32-bit tags are standard in most radio link layers including 5G [Sec5G], 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. As such, they are highly sensitive to cryptographic overhead, but as each packet typically encodes only 20 ms of audio, forgery of individual packets is not a big concern and barely noticeable. Due to its weaknesses, GCM is typically not used with short tags. The result is either decreased performance from larger than needed tags [I-D.ietf-moq-transport], or decreased performance from using much slower constructions such as AES-CTR combined with HMAC [RFC3711][RFC9605]. Short tags are also useful to protect packets whose payloads are secured at higher layers, protocols where the security is given by the sum of the tag lengths, and in constrained radio networks, where the low bandwidth limits the number of forgery attempts. For all applications of short tags it is essential that the MAC behaves like an ideal MAC, i.e., the forgery probability is 1 / 2^(tag_length) even after many generated MACs, many forgery attempts, and after a successful forgery. While Counter with CBC-MAC (CCM) [RFC5116] with short tags has forgery probabilities close to ideal, its performance is lower than that of GCM. For a comprehensive discussion on the use cases and requirements of short tags, see [Comments38B].

This document defines the Galois Counter Mode with Strong Secure 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, allowing it to be used with any keystream generator, not just 128-bit block ciphers. The main differences from GCM [GCM] are the introduction of an additional subkey H_2, the derivation of fresh subkeys H and H_2 for each nonce, and the replacement of the GHASH function with the POLYVAL function from AES-GCM-SIV [RFC8452], see Section 3. These changes enable truncated tags with near-ideal forgery probabilities, even against multiple forgery attacks, see Section 5. GCM-SST is designed for use in security protocols with replay protection such as TLS [RFC8446],

QUIC [RFC9000], SRTP [RFC3711], PDCP [PDCP], etc. Security protocols with replay protection are by far the most important use case for GCM. In unicast scenarios, its authentication tag behaves like an ideal MAC, including reforgeability resistance. Users and implementers of cryptography expect algorithms to behave like ideal MACs. Compared to Poly1305 [RFC7539] and GCM [RFC5116], GCM-SST can provide better integrity with less overhead. Its performance in hardware and software is similar to GCM [GCM]. In software, 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 retains 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 GCM-SST instances using Advanced Encryption Standard (AES) [AES] and Rijndael with 256-bit keys and blocks (Rijndael-256) [Rijndael] in counter mode as keystream generators and with tag lengths of 48, 96, and 112 bits, see Section 4. The authentication tags in all registered GCM-SST instances behave like ideal MACs, which is not the case at all for GCM [GCM]. 3GPP has standardized the use of Rijndael-256 for authentication and key generation in 3GPP TS 35.23435.237 [WID23]. NIST plans to standardize Rijndael-256 [Plans], although there might be revisions to the key schedule. Rijndael-256 has very good performance on modern x86-64 platforms equipped with AES-NI and VAES instructions [Ducker]. Compared to AEGIS [I-D.irtf-cfrg-aegis-aead], AES-GCM-SST offers significantly higher performance in pure hardware implementations while retaining the advantage of being a mode of operation for AES.

The integrity part of GCM-SST was originally developed by ETSI SAGE, under the name Mac5G, following a request from 3GPP, with several years of discussion and refinement contributing to its design [SAGE23][SAGE24]. Mac5G is constructed similarly to the integrity algorithms used for SNOW 3G [UIA2] and ZUC [EIA3]. 3GPP has decided to standardize GCM-SST for use with SNOW 5G [SNOW], AES-256 [AES], and ZUC-256 [ZUC] in 3GPP TS 35.24035.248 [WID24].

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.

The following notation is used in the document:

* 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]
* Z is the keystream
* ct is the ciphertext
* tag is the authentication tag
* tag_length is the length of tag in bits
* = is the assignment operator
* ≠ is the inequality operator
* x || y is concatenation of the octet strings x and y
* is the bitwise exclusive or operator XOR
* 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 128-bit chunks x to y in the array V

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

This section defines the Galois Counter Mode with Strong Secure 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, H_2, 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 H_2 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 can be found in [SST1], [SST2], and [Inoue].

For every computational procedure that is specified in this document, a conforming implementation MAY replace the given set of steps with any mathematically equivalent set of steps. In other words, different procedures that produce the correct output for every input are permitted.

3.1. Authenticated Encryption Function

The encryption function Encrypt(K, N, A, P) 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, a nonce MUST NOT be reused under any circumstances.

* Each key MUST be restricted to a single tag_length.

* 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 (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], H_2 = Z[1], M = Z[2]

4. Let ct = P 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(H_2, X L) M

9. Let tag = truncate(full_tag, tag_length)

10. Return (ct, tag)

The encoding of L aligns with existing implementations of GCM.

3.2. 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 the 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 the tag verification fails, the plaintext P and the expected_tag MUST NOT be given as output.

* Each key MUST be restricted to a single tag_length.

* 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 (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], H_2 = 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(H_2, X L) M

8. Let expected_tag = truncate(full_tag, tag_length)

9. If tag ≠ expected_tag, return error and abort

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

11. If N passes replay protection, return P

The comparison of tag and expected_tag in step 9 MUST be performed in constant time to prevent any information leakage about the position of the first mismatched byte. For a given key, a plaintext MUST NOT be returned unless it is certain that a plaintext has not been returned for the same nonce. Replay protection can be performed either before step 1 or during step 11. Protocols with nonce-hiding mechanisms [Bellare], such as QUIC [RFC9001], implement replay protection after decryption to mitigate timing side-channel attacks.

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 and Rijndael-256 in GCM-SST

This section defines Advanced Encryption Standard (AES) and Rijndael with 256-bit keys and blocks (Rijndael-256) [Rijndael] in Galois Counter Mode with Strong Secure Tags.

4.1. AES-GCM-SST

When GCM-SSM is instantiated with AES (AES-GCM-SST), the keystream generator is AES in counter mode

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

where ENC is the AES Cipher function [AES]. Big-endian counters align with existing implementations of AES in counter mode.

4.2. Rijndael-GCM-SST

When GCM-SST is instantiated with Rijndael-256 (Rijndael-GCM-SST), the keystream generator is Rijndael-256 in counter mode

Z[2i] = ENC(K, N || BE32(i))[0]

Z[2i+1] = ENC(K, N || BE32(i))[1]

where ENC is the Rijndael-256 Cipher function [Rijndael].

4.3. AEAD Instances and Constraints

We define nine AEAD instances, in the format of [RFC5116], that use AES-GCM-SST and Rijndael-GCM-SST with tag lengths of 48, 96, and 112 bits. The key length and tag length are related to different security properties, and an application encrypting audio packets with short tags might require high confidentiality.

Name	K_LEN (bytes)	P_MAX = A_MAX (bytes)	tag_length (bits)
AEAD_AES_128_GCM_SST_6	16	2^36 - 48	48
AEAD_AES_128_GCM_SST_12	16	2^35	96
AEAD_AES_128_GCM_SST_14	16	2^19	112
AEAD_AES_256_GCM_SST_6	32	2^36 - 48	48
AEAD_AES_256_GCM_SST_12	32	2^35	96
AEAD_AES_256_GCM_SST_14	32	2^19	112
AEAD_RIJNDAEL_GCM_SST_6	32	2^36 - 48	48
AEAD_RIJNDAEL_GCM_SST_12	32	2^35	96
AEAD_RIJNDAEL_GCM_SST_14	32	2^19	112

Table 1: AEAD Algorithms

Common parameters for the six AEAD instances:

* N_MIN = N_MAX (minimum and maximum size of the nonce) is 12 octets for AES, while for Rijndael-256, it is 28 bytes.

* C_MAX (maximum size of the ciphertext and tag) is P_MAX + tag_length (in bytes)

* Q_MAX (maximum number of invocations of the encryption function) is 2^32 for AES-GCM-SST, while for Rijndael-GCM-SST, it is 2^88.

* V_MAX (maximum number of invocations of the decryption function) is 2^48 for AES-GCM-SST, while for Rijndael-GCM-SST, it is 2^88.

The maximum size of the plaintext (P_MAX) and the maximum size of the associated data (A_MAX) have been lowered from GCM [RFC5116]. To enable forgery probability close to ideal, even with maximum size plaintexts and associated data, we set P_MAX = A_MAX = min(2^(131 - tag_length), 2^36 - 48). Protocols that employ GCM-SST MAY impose stricter limits on P_MAX and A_MAX. Just like [RFC5116], AES-GCM-SST and Rijndael-GCM-SST only allow a fixed nonce length (N_MIN = N_MAX) of 96 bits and 224 bits, respectively. For the AEAD algorithms in Table 1 the worst-case forgery probability is bounded by 1 / 2^(tag_length) [Nyberg]. This is true for all allowed plaintext and associated data lengths.

The V_MAX constraint ensures that the Bernstein bound factor is δ 1 for AES-GCM-SST in protocols where P_MAX + A_MAX 2^16, such as QUIC [RFC9000], and always δ 1 for Rijndael-GCM-SST. In addition to restricting the Bernstein bound factor, the Q_MAX constraint establishes a minimum threshold for the complexity of distinguishing attacks and a maximum threshold for the fraction of a plaintext bit that an attacker can recover. Since encryption and decryption queries play an equivalent role in the Bernstein bound, it follows that Q_MAX V_MAX. Protocols that employ GCM-SST MAY impose stricter limits on Q_MAX and V_MAX.

Protocols utilizing AES-GCM-SST MUST enforce stricter limits on P_MAX, A_MAX, Q_MAX, and/or V_MAX to ensure that (P_MAX + A_MAX) (Q_MAX + V_MAX) 2^66. This ensures that δ 1.

Protocols utilizing AES-GCM-SST MUST enforce stricter limits on P_MAX and/or Q_MAX to ensure that Q_MAX P_MAX 2^63. This aligns with [ANSSI] requirements and ensures that an attacker cannot recover more than 1 / 2^10.47 0.0007 bits of the plaintexts [Entropy].

Refer to Sections 5.1, 5.2, and 5.5 for further details.

5. Security Considerations

GCM-SST introduces an additional subkey H_2, alongside the subkey H. The inclusion of H_2 enables truncated tags with forgery probabilities close to ideal. Both H and H_2 are derived for each nonce, which significantly decreases the probability of multiple successful forgeries. These changes are based on proven theoretical constructions and follows the recommendations in [Nyberg]. Inoue et al. [Inoue] prove that GCM-SST is a provably secure authenticated encryption mode, with security guaranteed for evaluations under fresh nonces, even if some earlier nonces have been reused.

GCM-SST is designed for use in security protocols with replay protection. Every key MUST be randomly chosen from a uniform distribution. GCM-SST MUST be used in a nonce-respecting setting: for a given key, a nonce MUST only be used once in the encryption function and only once in a successful decryption function call. The nonce MAY be public or predictable. It can be a counter, the output of a permutation, or a generator with a long period. The reuse of nonces in successful encryption and decryption function calls enables universal forgery, as demonstrated by [Joux], [Lindell], and [Inoue]. Therefore, GCM-SST MUST be used with replay protection. Additionally, GCM-SST MUST NOT be used with random nonces, as this significantly reduces the efficiency of replay protection. For a given tag length, GCM-SST has strictly better security properties than GCM. GCM allows universal forgery with lower complexity than GCM-SST, even when nonces are not reused. Implementations SHOULD add randomness to the nonce by XORing a unique number like a sequence number with a per-key random secret salt of the same length as the nonce. This significantly improves security against precomputation attacks and multi-key attacks [Bellare] and is for example implemented in TLS 1.3 [RFC8446], OSCORE [RFC8613], and [Ascon]. By increasing the nonce length from 96 bits to 224 bits, Rijndael-GCM-SST can offer significantly greater security against precomputation and multi-key attacks compared to AES-256-GCM-SST.

Refer to Section 5.6 for considerations on using GCM-SST in multicast or broadcast scenarios. Unless otherwise specified, formulas for expected number of forgeries apply to unicast scenarios.

5.1. Integrity

The GCM-SST tag_length SHOULD NOT be smaller than 4 bytes and cannot be larger than 16 bytes. Let = (P_MAX + A_MAX) / 16 + 1. When tag_length < 128 - log2() bits, the worst-case forgery probability is bounded by 1 / 2^(tag_length) [Nyberg]. The tags in the AEAD algorithms listed in Section 4.3 therefore have an almost perfect security level. This is significantly better than GCM where the

security level is only tag_length - log2() bits [GCM]. For a graph of the forgery probability, refer to Fig. 3 in [Inoue]. 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. In GCM-SST, the full_tag is independent of the specified tag length unless the application explicitly incorporates tag length into the keystream or the nonce.

The expected number of forgeries, when tag_length < 128 - log2() bits, depends on the keystream generator. For an ideal keystream generator, the expected number of forgeries is v / 2^(tag_length), where v is the number of decryption queries, which is ideal. For AES-GCM-SST, the expected number of forgeries is δ_128 v / 2^(tag_length), where the Bernstein bound factor δ_b 1 + (q + v)^2 ^2 / 2^(b+1), which is ideal when δ_128 1. This far outperforms AES-GCM, where the expected number of forgeries is δ_128 v^2 / 2^(tag_length+1). For Rijndael-GCM-SST, the expected number of forgeries is δ_256 v / 2^(tag_length) v / 2^(tag_length), which is ideal. For further details on the integrity advantages and expected number of forgeries for GCM and GCM-SST, see [Iwata], [Inoue], [Bernstein], and [Multiple]. BSI states that an ideal MAC with a 96-bit tag length is considered acceptable for most applications [BSI], a requirement that GCM-SST with 96-bit tags satisfies when δ 1. Achieving a comparable level of security with GCM, CCM, or Poly1305 is nearly impossible.

5.2. Confidentiality

The confidentiality offered by GCM-SST against passive attackers depends on the keystream generator. For block ciphers in counter mode it is determined by the birthday bound, with AES-based ciphers particularly constrained by their narrow 128-bit block size. For AES-GCM-SST, the confidentiality is equal to AES-GCM [GCM]. Regardless of key length, an attacker can mount a distinguishing attack with a complexity of approximately 2^129 / σ, where σ P_MAX Q_MAX / 16 is is the total plaintext length measured in 128-bit chunks. In contrast, the confidentiality offered by Rijndael-GCM-SST against passive attackers is significantly higher. The complexity of distinguishing attacks for Rijndael-GCM-SST is approximately 2^258 / σ. McGrew, Leurent, and Sibleyras [Impossible][Difference] demonstrate that for block ciphers in counter mode, an attacker with partial knowledge of the plaintext can execute plaintext-recovery attacks against counter mode with roughly the same complexity (up to logaritmic factors) as distinguishing attacks. However, Preu Mattsson [Entropy] demonstrated that an attacker cannot recover more than σ^2 / 2^b bits of the plaintext, where b is the block size. Given the constraints outlined in Section 4.3, an attacker cannot

recover more than 0.0007 bits of AES-GCM-SST plaintexts.

While Rijndael-256 in counter mode can provide 128-bit confidentiality for plaintexts much larger than 2^36 octets, GHASH and POLYVAL do not offer adequate integrity for long plaintexts. To ensure robust integrity for long plaintexts, an AEAD mode would need to replace POLYVAL with a MAC that has better security properties, such as a Carter-Wegman MAC in a larger field [Degabriele] or other alternatives such as [SMAC].

The confidentiality offered by GCM-SST against active attackers is directly linked to the forgery probability. Depending on the protocol and application, forgeries can significantly compromise privacy, in addition to affecting integrity and authenticity. It MUST be assumed that attackers always receive feedback on the success or failure of their forgery attempts. Therefore, attacks on integrity, authenticity, and confidentiality MUST all be carefully evaluated when selecting an appropriate tag length.

5.3. Weak keys

In general, there is a very small possibility in GCM-SST that either or both of the subkeys H and H_2 are zero, so called weak keys. If H is zero, the authentication tag depends only on the length of P and A and not on their content. If H_2 is zero, the authentication tag does not depend on P and A. Due to the masking with M, 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 H_2 are generated with a permutation on different input, so H and H_2 cannot both be zero.

5.4. Replay Protection

The details of the replay protection mechanism is determined by the security protocol utilizing GCM-SST. If the nonce includes a sequence number, it can be used for replay protection. Alternatively, a separate sequence number can be used, provided there is a one-to-one mapping between sequence numbers and nonces. The choice of a replay protection mechanism depends on factors such as the expected degree of packet reordering, as well as protocol and implementation details. For examples of replay protection mechanisms, see [RFC4303] and [RFC6479]. Implementing replay protection by requiring ciphertexts to arrive in order and terminating the connection if a single decryption fails is NOT RECOMMENDED as this approach reduces robustness and availability while exposing the system to denial-of-service attacks [Robust].

5.5. Comparison with ChaCha20-Poly1305 and AES-GCM

Table 2 compares the integrity of GCM-SST, ChaCha20-Poly1305 [RFC7539], and AES-GCM [RFC5116] in unicast security protocols with replay protection, where v represents the number of decryption queries. In Poly1305 and GCM, the forgery probability depends on = (P_MAX + A_MAX) / 16 + 1. In AES-based algorithms, the Bernstein bound introduces a factor δ 1 + (q + v)^2 ^2 / 2^129. GCM-SST requires δ 1, a property not mandated by GCM. Notably, GCM does not provide any reforgeability resistance, which significantly increases the expected number of forgeries. Refer to [Procter], [Iwata], and [Multiple] for further details.

Name	Forgery probability before first forgery	Forgery probability after first forgery	Expected number of forgeries
GCM_SST_14	1 / 2^112	1 / 2^112	v / 2^112
GCM_SST_12	1 / 2^96	1 / 2^96	v / 2^96
POLY1305	/ 2^103	/ 2^103	v / 2^103
GCM	/ 2^128	1	δ v^2 / 2^129

Table 2: Comparison of integrity among GCM-SST, ChaCha20-Poly1305, and AES-GCM in unicast security protocols with replay protection. v is the number of decryption queries, is the maximum length of plaintext and associated data, measured in 128-bit chunks, and δ is the Bernstein bound factor.

Table 3 compares the integrity of GCM-SST, ChaCha20-Poly1305 [RFC7539], and AES-GCM [RFC5116] in unicast QUIC [RFC9000][RFC9001], a security protocol with mandatory replay protection, and where the combined size of plaintext and associated data is less than 2^16 bytes (2^12). GCM_SST_14 and GCM_SST_12 provide better integrity than ChaCha20-Poly1305 [RFC7539] and AES-GCM [RFC5116], while also reducing overhead by 24 bytes. For GCM-SST and ChaCha20-Poly1305, the expected number of forgeries is linear in v when replay protection is employed. ChaCha20-Poly1305 achieves a security level equivalent to that of an ideal MAC with a length of 91 bits. For AES-GCM, replay protection does not mitigate reforgeries, the expected number of forgeries grows quadratically with v, and GCM provides significantly worse integrity than GCM-SST and

ChaCha20-Poly1305 unless v is kept very small. With v = 2^52 as allowed for AES-GCM in QUIC [RFC9001], the expected number of forgeries for AES-GCM is equivalent to that of an ideal MAC with a length of 64.4 bits. The effective tag length of AES-GCM in QUIC is 117 - log2(δ v).

Name	Tag length (bytes)	Forgery probability before first forgery	Forgery probability after first forgery	Expected number of forgeries
GCM_SST_14	14	1 / 2^112	1 / 2^112	v / 2^112
GCM_SST_12	12	1 / 2^96	1 / 2^96	v / 2^96
POLY1305	16	1 / 2^91	1 / 2^91	v / 2^91
GCM	16	1 / 2^116	1	δ v^2 / 2^117

Table 3: Comparison of integrity among GCM-SST, ChaCha20-Poly1305, and AES-GCM in unicast QUIC, where the maximum packet size is 65536 bytes.

Table 4 compares the confidentiality of Rijndael-GCM-SST, AES-256-GCM-SST, SNOW 5G-GCM-SST, and ChaCha20-Poly1305 [RFC7539], all of which use 256-bit keys, against passive attackers. The confidentiality of block ciphers in counter mode is determined by the birthday bound, with AES-based ciphers particularly constrained by their narrow 128-bit block size. While plaintext-recovery attacks on block ciphers in counter mode have a complexity similar to distinguishing attacks, the attacker cannot recover more than σ ^2 / 2^b bits of the plaintexts [Entropy].

Name	Key size (bits)	Complexity of distinguishing attacks
CHACHA20_POLY1305	256	2^256
SNOW_5G_GCM_SST	256	2^256
RIJNDAEL_GCM_SST	256	2^258 / σ
AES_256_GCM_SST	256	2^129 / σ

Table 4: Comparison of confidentiality against passive attackers among Rijndael-GCM-SST, SNOW 5G-GCM-SST, ChaCha20-Poly1305, and AES- 256-GCM. σ is is the total plaintext length measured in 128-bit chunks.

5.6. Multicast and Broadcast

While GCM-SST offers stronger security properties than GCM for a given tag length in multicast or broadcast contexts, it does not behave exactly like an ideal MAC. With an ideal MAC, a successful forgery against one recipient allows the attacker to reuse the same forgery against all other recipients. In contrast, with GCM, a successful forgery against one recipient enables the attacker to generate an unlimited number of new forgeries for all recipients.

With GCM-SST, a few successful forgeries against a few recipients allow the attacker to create one new forgery for all other recipients. While the total number of forgeries in GCM-SST matches that of an ideal MAC, the diversity of these forgeries is higher. To achieve one distinct forgery per recipient with an ideal MAC, the attacker would need to send on average 2^(tag_length) forgery attempts to each recipient. GCM-SST performs equally well or better than an ideal MAC with length tag_length - log2(r), where r is the number of recipients. Thus, in one-to-many scenarios with replay protection, the expected number of distinct forgeries is v r / 2^(tag_length), and the effective tag length of GCM-SST is tag_length - log2(r).

6. IANA Considerations

IANA is requested to assign the entries in the first column 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, May 2023, <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>.

[Rijndael] Joan Daemen and Vincent Rijmen, "AES Proposal: Rijndael", September 2003, <https://csrc.nist.gov/csrc/media/projects/cryptographic-standards-and-guidelines/documents/aes-development/rijndael-ammended.pdf>.

7.2. Informative References

[AEZ] Viet Tung Hoang, Ted Krovetz, and Phillip Rogaway, "Robust Authenticated-Encryption: AEZ and the Problem that it Solves", March 2017, <https://eprint.iacr.org/2014/793.pdf>.

[ANSSI] "Guide des mcanismes cryptographiques", ANSSI PG-083, January 2020, <https://cyber.gouv.fr/sites/default/files/2021/03/anssi-guide-mecanismes_crypto-2.04.pdf>.

[Ascon] Meltem Snmez Turan, Kerry A McKay, Donghoon Chang, Jinkeon Kang, and John Kelsey, "Ascon-Based Lightweight Cryptography Standards for Constrained Devices",

NIST Special Publication 800-232 Initial Public Draft, November 2024, <https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-232.ipd.pdf>.

[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>.

[Bernstein] Daniel J Bernstein, "Stronger Security Bounds for Permutations", March 2005, <https://cr.yp.to/antiforgery/permutations-20050323.pdf>.

[BSI] "Cryptographic Mechanisms Recommendations and Key Lengths", BSI Technical Guideline TR-02102-1, February 2024, <https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/TechGuidelines/TG02102/BSI-TR-02102-1.html>.

[Comments38B] NIST, "Public Comments on SP 800-38B", September 2024, <https://csrc.nist.gov/csrc/media/Projects/crypto-publication-review-project/documents/initial-comments/sp800-38b-initial-public-comments-2024.pdf>.

[Degabriele] Degabriele, J., Gilcher, J., Govinden, J., and K. Paterson, "SoK: Efficient Design and Implementation of Polynomial Hash Functions over Prime Fields", May 2024, <https://doi.org/10.3929/ethz-b-000654260>.

[Difference] Gatan Leurent and Ferdinand Sibleyras, "The Missing Difference Problem, and Its Applications to Counter Mode Encryption", March 2018, <https://link.springer.com/chapter/10.1007/978-3-319-78375-8_24>.

[Ducker] Nir Drucker and Shay Gueron, "Software Optimization of Rijndael for Modern x86-64 Platforms", May 2022, <https://rd.springer.com/chapter/10.1007/978-3-030-97652-1_18>.

[EIA3] ETSI SAGE, "128-EEA3 and 128-EIA3 Specification", January 2019, <https://www.gsma.com/solutions-and-impact/technologies/security/wp-content/uploads/2019/05/EEA3_EIA3_specification_v1_8.pdf>.

[Entropy] Preu Mattsson, J., "Collision-Based Attacks on Block Cipher Modes - Exploiting Collisions and Their Absence", February 2025, <https://eprint.iacr.org/2024/1111.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-moq-transport] Curley, L., Pugin, K., Nandakumar, S., Vasiliev, V., and I. Swett, "Media over QUIC Transport", Work in Progress, Internet-Draft, draft-ietf-moq-transport-08, 12 February 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-moq-transport-08>.

[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-16, 17 February 2025, <https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-aegis-aead-16>.

[Impossible] David McGrew, "Impossible plaintext cryptanalysis and probable-plaintext collision attacks of 64-bit block cipher modes", November 2012, <https://eprint.iacr.org/2012/623.pdf>.

[Inoue] Akiko Inoue, Ashwin Jha, Bart Mennink, and Kazuhiko Minematsu, "Generic Security of GCM-SST", November 2024, <https://eprint.iacr.org/2024/1928.pdf>.

[Iwata] Tetsu Iwata, Keisuke Ohashi, and Kazuhiko Minematsu, "Breaking and Repairing GCM Security Proofs", August 2012, <https://eprint.iacr.org/2012/438.pdf>.

[Joux] Joux, A., "Authentication Failures in NIST version of GCM", April 2006, <https://csrc.nist.gov/csrc/media/projects/block-cipher-techniques/documents/bcm/comments/800-38-series-drafts/gcm/joux_comments.pdf>.

[Lindell] Lindell, Y., "Comment on AES-GCM-SST", May 2024, <https://mailarchive.ietf.org/arch/browse/cfrg/?gbt=1&index=cWpv0QgX2ltkWhtd3R9pEW7E1CA>.

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

[Multiple] David McGrew and Scott Fluhrer, "Multiple Forgery Attacks Against Message Authentication Codes", May 2005, <https://csrc.nist.gov/csrc/media/projects/block-cipher-techniques/documents/bcm/comments/cwc-gcm/multi-forge-01.pdf>.

[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>.

[PDCP] 3GPP TS 38 323, "NR; Packet Data Convergence Protocol (PDCP) specification", December 2024, <https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3196>.

[Plans] NIST, "NIST Proposes to Standardize a Wider Variant of AES", December 2024, <https://csrc.nist.gov/pubs/sp/800/197/iprd>.

[Procter] Gordon Procter, "A Security Analysis of the Composition of ChaCha20 and Poly1305", August 2014, <https://eprint.iacr.org/2014/613.pdf>.

[Reforge] Christian Forler, Eik List, Stefan Lucks, and akob Wenzel, "Reforgeability of Authenticated Encryption Schemes", April 2017, <https://eprint.iacr.org/2017/332.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>.

[RFC4303] Kent, S., "IP Encapsulating Security Payload (ESP)", RFC 4303, DOI 10.17487/RFC4303, December 2005, <https://www.rfc-editor.org/rfc/rfc4303>.

[RFC6479] Zhang, X. and T. Tsou, "IPsec Anti-Replay Algorithm without Bit Shifting", RFC 6479, DOI 10.17487/RFC6479, January 2012, <https://www.rfc-editor.org/rfc/rfc6479>.

[RFC7539] Nir, Y. and A. Langley, "ChaCha20 and Poly1305 for IETF Protocols", RFC 7539, DOI 10.17487/RFC7539, May 2015, <https://www.rfc-editor.org/rfc/rfc7539>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/rfc/rfc8446>.

[RFC8613] Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security for Constrained RESTful Environments (OSCORE)", RFC 8613, DOI 10.17487/RFC8613, July 2019, <https://www.rfc-editor.org/rfc/rfc8613>.

[RFC9000] Iyengar, J., Ed. and M. Thomson, Ed., "QUIC: A UDP-Based Multiplexed and Secure Transport", RFC 9000, DOI 10.17487/RFC9000, May 2021, <https://www.rfc-editor.org/rfc/rfc9000>.

[RFC9001] Thomson, M., Ed. and S. Turner, Ed., "Using TLS to Secure QUIC", RFC 9001, DOI 10.17487/RFC9001, May 2021, <https://www.rfc-editor.org/rfc/rfc9001>.

[RFC9605] Omara, E., Uberti, J., Murillo, S. G., Barnes, R., Ed., and Y. Fablet, "Secure Frame (SFrame): Lightweight Authenticated Encryption for Real-Time Media", RFC 9605, DOI 10.17487/RFC9605, August 2024, <https://www.rfc-editor.org/rfc/rfc9605>.

[Robust] Fischlin, M., Gnther, F., and C. Janson, "Robust Channels: Handling Unreliable Networks in the Record Layers of QUIC and DTLS 1.3", January 2024, <https://link.springer.com/article/10.1007/s00145-023-09489-9>.

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

[SAGE23] ETSI SAGE, "Specification of the 256-bit air interface algorithms", February 2023, <https://www.3gpp.org/ftp/TSG_SA/WG3_Security/TSGS3_110_Athens/docs/S3-230642.zip>.

[SAGE24] ETSI SAGE, "Version 2.0 of 256-bit Confidentiality and Integrity Algorithms for the Air Interface", August 2024, <https://www.3gpp.org/ftp/tsg_sa/WG3_Security/TSGS3_117_Maastricht/docs/S3-243394.zip>.

[Sec5G] 3GPP TS 33 501, "Security architecture and procedures for 5G System", September 2024, <https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3169>.

[SMAC] Wang, D., Maximov, A., Ekdahl, P., and T. Johansson, "A new stand-alone MAC construct called SMAC", June 2024, <https://eprint.iacr.org/2024/819>.

[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. Preu Mattsson, "Galois Counter Mode with Strong Secure 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. Preu Mattsson, "Galois Counter Mode with Strong Secure 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>.

[UIA2] ETSI SAGE, "UEA2 and UIA2 Specification", March 2009, <https://www.gsma.com/solutions-and-impact/technologies/security/wp-content/uploads/2019/05/uea2uia2d1v21.pdf>.

[WID23] 3GPP, "New WID on Milenage-256 algorithm", November 2023, <https://www.3gpp.org/ftp/tsg_sa/WG3_Security/TSGS3_113_Chicago/Docs/S3-235072.zip>.

[WID24] 3GPP, "New WID on Addition of 256-bit security Algorithms", March 2024, <https://www.3gpp.org/ftp/tsg_sa/TSG_SA/TSGS_103_Maastricht_2024-03/Docs/SP-240476.zip>.

[ZUC] ZUC Design Team, "An Addendum to the ZUC-256 Stream Cipher", September 2024, <https://eprint.iacr.org/2021/1439>.

Appendix A. AES-GCM-SST Test Vectors

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

```
            K = { 00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f }
            N = { 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 }
          H_2 = { 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

```
            A = { }
            P = { }
            L = { 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 42 b0 0a ec b0 bc eb 8d }
           ct = { }
```

Case #1b

```
            A = { 40 41 42 43 44 }
            P = { }
            L = { 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 d5 f3 08 a5 70 4e 2f d5 }
           ct = { }
```

Case #1c

```
            A = { }
            P = { 60 61 62 63 64 65 66 67 68 69 6a 6b }
            L = { 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 fd 1a 90 d9 81 8f cb 7b }
           ct = { 64 f0 5b ae 1e d2 40 3a 71 25 5e dd }
```

Case #1d

```
            A = { 40 41 42 43 44 45 46 47 48 49 4a 4b 4c 4d 4e 4f }
            P = { 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 }
            L = { 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 0b 84 48 2c d0 14 c7 40 }
           ct = { 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

```
            A = { 40 41 42 43 44 45 46 47 48 49 4a 4b 4c 4d 4e }
            P = { 60 61 62 63 64 65 66 67 68 69 6a 6b 6c 6d 6e 6f
                  70 }
            L = { 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 11 43 ab e9 31 5a d7 eb }
           ct = { 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)

```
            K = { 29 23 be 84 e1 6c d6 ae 52 90 49 f1 f1 bb e9 eb }
            N = { 9a 50 ee 40 78 36 fd 12 49 32 f6 9e }
            A = { 1f 03 5a 7d 09 38 25 1f 5d d4 cb fc 96 f5 45 3b
                  13 0d }
            P = { 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 }
          H_2 = { 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 }
            L = { 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 }
           ct = { 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)

```
            K = { 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 }
            N = { 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 }
          H_2 = { 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

```
            A = { }
            P = { }
            L = { 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 2a 33 8e ec }
           ct = { }
```

Case #3b

```
            A = { 40 41 42 43 44 }
            P = { }
            L = { 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 28 ff c3 17 }
           ct = { }
```

Case #3c

```
            A = { }
            P = { 60 61 62 63 64 65 66 67 68 69 6a 6b }
            L = { 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 48 bd 6f cc }
           ct = { fc 46 2d 34 a7 5b 22 62 4f d7 3b 27 }
```

Case #3d

```
            A = { 40 41 42 43 44 45 46 47 48 49 4a 4b 4c 4d 4e 4f }
            P = { 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 }
            L = { 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 a5 a8 a2 8e }
           ct = { 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

```
            A = { 40 41 42 43 44 45 46 47 48 49 4a 4b 4c 4d 4e }
            P = { 60 61 62 63 64 65 66 67 68 69 6a 6b 6c 6d 6e 6f
                  70 }
            L = { 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 64 ce fd 03 }
           ct = { 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)

```
            K = { 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 }
            N = { 9a 50 ee 40 78 36 fd 12 49 32 f6 9e }
            A = { 1f 03 5a 7d 09 38 25 1f 5d d4 cb fc 96 f5 45 3b
                  13 0d }
            P = { 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 }
          H_2 = { 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 }
            L = { 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 73 49 bf 3c }
           ct = { b5 c2 a4 07 f3 3e 99 88 de c1 2f 10 64 7b 3d 4f
                  eb 8f f7 cc }
```

Appendix B. Compatibility with 3GPP Algorithms

The integrity part of GCM-SST was originally developed by ETSI SAGE under the name Mac5G, following a request from 3GPP. Its design evolved over several years of discussion and refinement [SAGE23][SAGE24]. Mac5G shares structural similarities with the integrity algorithms used for SNOW 3G [UIA2] and ZUC [EIA3].

3GPP has decided to standardize GCM-SST for use with SNOW 5G [SNOW], AES-256 [AES], and ZUC-256 [ZUC] in 3GPP TS 35.24035.248 [WID24]. These AEAD algorithms are designated as NCA4, NCA5, and NCA6, respectively. GCM-SST, as specified in this document, is fully compatible with the SNOW 5G-based NCA4 and the ZUC-256-based NCA6. The AES-based NCA5 differs only in its subkey generation but is otherwise identical. SNOW 5G is functionally equivalent to SNOW-Vi [SNOW], except that its FSM adders have been changed from 32-bit to 16-bit. Additionally, the NCA algorithms introduce more detailed specifications for nonce construction based on 3GPP protocols.

The version of GCM-SST specified in this document imposes stricter security considerations and constraints than the 3GPP and ETSI SAGE specifications for the NCA algorithms. We recommend 3GPP to follow the additional security measures outlined in this document.

Change Log

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

Changes from -17 to -18:

* Improved explanation why replay protection is required and random nonces forbidden.

* New appendix discussing compatibility with 3GPP algorithms.

* Editorial changes

Changes from -16 to -17:

* Align with ANSSI requirement on the maximum number of plaintext blocks.

* Added information of how small fraction of a bit an attacker can theoretically recover.

* Editorial changes

Changes from -15 to -16:

* Added section on multicast or broadcast allowing use in one-to-many scenarios. GCM-SST provides good security in such scenarios.

* Renamed Q to H_2 and some q to σ to avoid using the same symbol for different things.

* Updated information on confidentiality against passive attackers including comparison and links to papers showing that plaintext-recovery have similar complexity as distinguishing attacks.

* Editorial changes

Changes from -14 to -15:

* Added 48-bit tags after feedback from media people.

* Added comparison to AEGIS in pure hardware based on Scott Fluhrer’s analysis

* Changed "acceptable level" to "minimum threshold" as it can be discussed if 64-bit security is acceptable.

* Added requirement that security protocols using AES-GCM-SST MUST enforce limits to ensure that δ 1.

* Added that this specification does not allow use in multicast or broadcast. This simplify security considerations.

* Editorial changes

Changes from -13 to -14:

* Explained the formatting of L as well as why replay protection after decryption may be used.

* Updated link to NIST’s plans to standardize Rijndael-256 and aligned the name with NIST

* Aligned test vector tag length and terminology with the rest of the specification

* Editorial changes

Changes from -12 to -13:

* Changed name to Strong Secure Tags to better illustrate that GCM-SST is intended to improve security for all tag lengths.

* Editorial changes

Changes from -11 to -12:

* Introduced Q_MAX and V_MAX as formal names for the invocation constraints.

* Described that masking is important to mitigate weak keys.

* Improved and expanded the section comparing GCM-SST with Poly1305 and GCM.

* Editorial changes including subsections in the security considerations.

Changes from -10 to -11:

* Added that protocols can impose stricter limits on P_MAX and A_MAX

* Added constraints on the number of decryption queries v

* More info on replay protection implementation

* More info on nonce constructions

* Introduced the Bernstein bound factor δ instead of just assuming that $\delta < 2$

* Clarified differences between GCM-SST with different keystream generators (ideal, AES, Rijndael)

* Made it clearer that Table 1 is for unicast security protocols with replay protection and that Poly1305 is keyed with ChaCha20.

* Editorial changes including RFC 2119 terminology

Changes from -09 to -10:

* Corrected some probabilities that were off by a factor 2

* Editorial changes.

Changes from -07 to -09:

* Changed replay requirements to allow replay protection after decryption to align with protocols like QUIC and DTLS 1.3.

* Added a comparison between GCM_SST_14, GCM_SST_12, GCM_16, POLY1305 in protocols like QUIC

* Added text on the importance of behaving like an ideal MAC

* Consideration on replay protection mechanisms

* Added text and alternative implementations borrowed from NIST

* Added constraints of 2^32 encryption invocations aligning with NIST

* Added text explaining that GCM-SST offer strictly better security than GCM and that "GCM allows universal forgery with lower complexity than GCM-SST, even when nonces are not reused", to avoid any misconceptions that Lindell’s attack makes GCM-SST weaker than GCM in any way.

Changes from -06 to -07:

* Replaced 80-bit tags with 96- and 112-bit tags.

* Changed P_MAX and A_MAX and made them tag_length dependent to enable 96- and 112-bit tags with near-ideal security.

* Clarified that GCM-SST tags have near-ideal forgery probabilities, even against multiple forgery attacks, which is not the case at all for GCM.

* Added formulas for expected number of forgeries for GCM-SST (q 2^(-tag_length)) and GCM (q^2 (n + m + 1) 2^(-tag_length + 1)) and stated that GCM-SST fulfils BSI recommendation of using 96-bit ideal MACs.

Changes from -04 to -06:

* Reference to Inoue et al. for security proof, forgery probability graph, and improved attack when GCM-SST is used without replay protection.

* Editorial changes.

Changes from -03 to -04:

* Added that GCM-SST is designed for unicast protocol with replay protection

* Update info on use cases for short tags

* Updated info on ETSI and 3GPP standardization of GCM-SST

* Added Rijndael-256-256

* Added that replay is required and that random nonces, multicast, and broadcast are forbidden based on attack from Yehuda Lindell

* Security considerations for active attacks on privacy as suggested by Thomas Bellebaum

* Improved text on H and Q being zero.

* Editorial changes.

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, Thomas Bellebaum, Patrik Ekdahl, Scott Fluhrer, Eric Lagergren, Yehuda Lindell, Kazuhiko Minematsu, Santeri Paavolainen, Sini Ruohomaa, Erik Thormarker, and Magnus Westerlund 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
Sweden
Email: alexander.maximov@ericsson.com

John Preu Mattsson
Ericsson
Sweden
Email: john.mattsson@ericsson.com