CFRG
Internet-Draft
Intended status: Informational
Expires: November 22, 2017

S. Gueron
University of Haifa and Amazon Web Services
A. Langley
Google
Y. Lindell
Bar Ilan University
May 21, 2017

AES-GCM-SIV: Nonce Misuse-Resistant Authenticated Encryption
draft-irtf-cfrg-gcmsiv-05

Abstract

This memo specifies two authenticated encryption algorithms that are nonce misuse-resistant - that is that they do not fail catastrophically if a nonce is repeated.

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 http://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 November 22, 2017.

Table of Contents

1. Introduction 2
2. Requirements Language 3
3. POLYVAL 3
4. Encryption 4
5. Decryption 6
6. AEADs 6
7. Field operation examples 7
8. Worked example 7
9. Security Considerations 8
10. IANA Considerations 9
11. Acknowledgements 9
12. References 9
12.1. Normative References 9
12.2. Informative References 10
Appendix A. The relationship between POLYVAL and GHASH 10
Appendix B. Additional comparisons with AES-GCM 12
Appendix C. Test vectors 12
C.1. AEAD_AES_128_GCM_SIV 12
C.2. AEAD_AES_256_GCM_SIV 19
C.3. Counter wrap tests 27
Authors' Addresses 28

1. Introduction

The concept of "Authenticated encryption with additional data" (AEAD [RFC5116]) couples confidentiality and integrity in a single operation that is easier for practitioners to use correctly. The most popular AEAD, AES-GCM [GCM], is seeing widespread use due to its attractive performance.

However, most AEADs suffer catastrophic failures of confidentiality and/or integrity when two distinct messages are encrypted with the same nonce. While the requirements for AEADs specify that the pair of (key, nonce) shall only ever be used once, and thus prohibit this, in practice this is a worry.

Nonce misuse-resistant AEADs do not suffer from this problem. For this class of AEADs, encrypting two messages with the same nonce only discloses whether the messages were equal or not. This is the minimum amount of information that a deterministic algorithm can leak in this situation.

This memo specifies two nonce misuse-resistant AEADs: "AEAD_AES_128_GCM_SIV" and "AEAD_AES_256_GCM_SIV". These AEADs are designed to be able to take advantage of existing hardware support for AES-GCM and can decrypt within 5% of the speed of AES-GCM (for multi-kilobyte messages). Encryption is, perforce, slower than AES-GCM because two passes are required. However, measurements suggest that it can still run at 2/3rds of the speed of AES-GCM.

We suggest that these AEADs be considered in any situation where there is the slightest doubt about nonce uniqueness.

2. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

3. POLYVAL

The GCM-SIV construction is similar to GCM: the block cipher is used in counter mode to encrypt the plaintext and a polynomial authenticator is used to provide integrity. The authenticator in GCM-SIV is called POLYVAL.

POLYVAL, like GHASH, operates in a binary field of size 2^128. The field is defined by the irreducible polynomial x^128 + x^127 + x^126 + x^121 + 1. The sum of any two elements in the field is the result of XORing them. The product of any two elements is calculated using standard (binary) polynomial multiplication followed by reduction modulo the irreducible polynomial.

We define another binary operation on elements of the field: dot(a, b), where dot(a, b) = a * b * x^-128. The value of the field element x^-128 is equal to x^127 + x^124 + x^121 + x^114 + 1. The result, dot(a, b), of this multiplication is another field element.

Polynomials in this field are converted to and from 128-bit strings by taking the least-significant bit of the first byte to be the coefficient of x^0, the most-significant bit of the first byte to the coefficient of x^7 and so on, until the most-significant bit of the last byte is the coefficient of x^127.

POLYVAL takes a field element, H, and a series of field elements X_1, ..., X_s. Its result is S_s, where S is defined by the iteration S_0 = 0; S_j = dot(S_{j-1} + X_j, H), for j = 0..s

We note that POLYVAL(H, X_1, X_2, ...) is equal to ByteReverse(GHASH(ByteReverse(H) * x, ByteReverse(X_1),

ByteReverse(X_2), ...)), where ByteReverse is a function that reverses the order of 16 bytes. See Appendix A for a more detailed explanation.

4. Encryption

AES-GCM-SIV encryption takes a 16- or 32-byte key-generating key, a 96-bit nonce, and arbitrary-length plaintext & additional data byte-strings. It outputs an authenticated ciphertext that will be 16 bytes longer than the plaintext.

If the key-generating key is 16 bytes long then AES-128 is used throughout. Otherwise AES-256 is used throughout.

The first step of encryption is to generate per-nonce, record-authentication and record-encryption keys. The record-authentication key is 128-bit and the record-encryption key is either 128- (for AES-128) or 256-bit (for AES-256).

These keys are generated by encrypting a series of plaintext blocks that contain a 32-bit, little-endian counter followed by the nonce, and then discarding the second half of the resulting ciphertext. In the AES-128 case, 128 + 128 = 256 bits of key material need to be generated and, since encrypting each block yields 64 bits after discarding half, four blocks need to be encrypted. The counter values for these blocks are 0, 1, 2 and 3. For AES-256, six blocks are needed in total, with counter values 0 through 5 (inclusive).

In pseudocode form, where ++ indicates concatenation and x[:8] indicates taking only the first eight bytes from x:

```
if bytelen(key-generating-key) == 16 {
  record-authentication-key =
      AES128(key = key-generating-key,
             input = "00000000" ++ nonce)[:8] ++
      AES128(key = key-generating-key,
             input = "01000000" ++ nonce)[:8]
  record-encryption-key =
      AES128(key = key-generating-key,
             input = "02000000" ++ nonce)[:8] ++
      AES128(key = key-generating-key,
             input = "03000000" ++ nonce)[:8]
} else if bytelen(key-generating-key) == 32 {
  record-authentication-key =
      AES256(key = key-generating-key,
             input = "00000000" ++ nonce)[:8] ++
      AES256(key = key-generating-key,
             input = "01000000" ++ nonce)[:8]
  record-encryption-key =
      AES256(key = key-generating-key,
             input = "02000000" ++ nonce)[:8] ++
      AES256(key = key-generating-key,
             input = "03000000" ++ nonce)[:8] ++
      AES256(key = key-generating-key,
             input = "04000000" ++ nonce)[:8] ++
      AES256(key = key-generating-key,
             input = "05000000" ++ nonce)[:8]
}
```

Define the _length block_ as a 16-byte value that is the concatenation of the 64-bit, little-endian encodings of bytelen(additional_length) * 8 and bytelen(plaintext) * 8. Pad the plaintext and additional data with zeros until they are each a multiple of 16 bytes, the AES block size. Then X_1, X_2, ... (the series of field elements that are inputs to POLYVAL) are the concatenation of the padded additional data, the padded plaintext and the length block.

Calculate S_s = POLYVAL(record-authentication-key, X_1, X_2, ...). XOR the first twelve bytes of S_s with the nonce and clear the most-significant bit of the last byte. Encrypt the result with AES using the record-encryption key to produce the tag.

The ciphertext is produced by using AES, with the record-encryption key, in counter mode on the unpadded plaintext. The initial counter block is the tag with the most-significant bit of the last byte set to one. The counter advances by incrementing the first 32 bits interpreted as an unsigned, little-endian integer. The result of the

encryption is the resulting ciphertext (truncated to the length of the plaintext) followed by the tag.

5. Decryption

Decryption takes a 16- or 32-byte key-generating key, a 96-bit nonce, and arbitrary-length ciphertext & additional data byte-strings. It either fails, or outputs a plaintext that is 16 bytes shorter than the ciphertext.

Firstly, the record-encryption and record-authentication keys are derived in the same manner as when encrypting.

If the ciphertext is less than 16 bytes or more than 2^36 + 16 bytes, then fail. Otherwise split the input into the encrypted plaintext and a 16-byte tag. Decrypt the encrypted plaintext with the record-encryption key in counter mode, where the initial counter block is the tag with the most-significant bit of the last byte set to one. The counter advances in the same way as for encryption.

Pad the additional data and plaintext with zeros until they are each a multiple of 16 bytes, the AES block size. Calculate the length block and X_1, X_2, ... as above and compute S_s = POLYVAL(record-authentication-key, X_1, X_2, ...). Compute the expected tag by XORing S_s and the nonce, clearing the most-significant bit of the last byte and encrypting with the record-encryption key. Compare the provided and expected tag values in constant time. If they do not match, fail. Otherwise return the plaintext.

6. AEADs

We define two AEADs, in the format of RFC 5116, that use AES-GCM-SIV: AEAD_AES_128_GCM_SIV and AEAD_AES_256_GCM_SIV. They differ only in the size of the AES key used.

The key input to these AEADs becomes the key-generating key. Thus AEAD_AES_128_GCM_SIV takes a 16-byte key and AEAD_AES_256_GCM_SIV takes a 32-byte key.

The parameters for AEAD_AES_128_GCM_SIV are then: K_LEN is 16, P_MAX is 2^36, A_MAX is 2^61 - 1, N_MIN and N_MAX are 12 and C_MAX is 2^36 + 16.

The parameters for AEAD_AES_256_GCM_SIV differ only in the key size: K_LEN is 32, P_MAX is 2^36, A_MAX is 2^61 - 1, N_MIN and N_MAX are 12 and C_MAX is 2^36 + 16.

7. Field operation examples

Polynomials in this document will be written as 16-byte values. For example, the sixteen bytes 01000000000000000000000000000492 would represent the polynomial x^127 + x^124 + x^121 + x^114 + 1, which is also the value of x^-128 in this field.

If a = 66e94bd4ef8a2c3b884cfa59ca342b2e and b = ff000000000000000000000000000000 then a + b = 99e94bd4ef8a2c3b884cfa59ca342b2e, a * b = 37856175e9dc9df26ebc6d6171aa0ae9 and dot(a, b) = ebe563401e7e91ea3ad6426b8140c394.

8. Worked example

Consider the encryption of the plaintext "Hello world" with the additional data "example" under key ee8e1ed9ff2540ae8f2ba9f50bc2f27c using AEAD_AES_128_GCM_SIV. The random nonce that we'll use for this example is 752abad3e0afb5f434dc4310.

In order to generate the record-authentication and record-encryption keys, a counter is combined with the nonce to form four blocks. These blocks are encrypted with key given above:

```
Counter |       Nonce                          Ciphertext
00000000752abad3e0afb5f434dc4310 -> 310728d9911f1f38c40e952ca83d093e
01000000752abad3e0afb5f434dc4310 -> 37b24316c3fab9a046ae90952daa0450
02000000752abad3e0afb5f434dc4310 -> a4c5ae624996327947920b2d2412474b
03000000752abad3e0afb5f434dc4310 -> c100be4d7e2c6edd1efef004305ab1e7
```

The latter halves of the ciphertext blocks are discarded and the remaining bytes are concatenated to form the per-record keys. Thus the record-authentication key is 310728d9911f1f3837b24316c3fab9a0 and the record-encryption key is a4c5ae6249963279c100be4d7e2c6edd.

The length block contains the encoding of the bit-lengths of the additional data and plaintext, respectively, which are and 56 and 88. Thus the length block is 38000000000000005800000000000000.

The input to POLYVAL is the padded additional data, padded plaintext and then the length block. This is 6578616d706c650000000000000000048656c6c6f20776f726c6400000000003800000000000000580000000000000000.

Calling POLYVAL with the record-authentication key and the input above results in S_s = ad7fcf0b5169851662672f3c5f95138f.

Before encrypting, the nonce is XORed in and the most-significant bit of the last byte is cleared. This gives

d85575d8b1c630e256bb6c2c5f95130f because that bit happened to be one previously. Encrypting with the record-encryption key gives the tag, which is 4fbcdeb7e4793f4a1d7e4faa70100af1.

In order to form the initial counter block, the most-significant bit of the last byte of the tag is set to one. That doesn't result in a change in this example. Encrypting this with the record key gives the first block of the keystream: 1551f2c1787e81deac9a99f139540ab5.

The final ciphertext is the result of XORing the plaintext with the keystream and appending the tag. That gives 5d349ead175ef6b1def6fd4fbcdeb7e4793f4a1d7e4faa70100af1.

9. Security Considerations

A detailed analysis of these schemes appears in [AES-GCM-SIV] and the remainder of this section is a summary of that paper.

We recommend a limit of 2^50 plaintexts encrypted with a given key. Past this point, AES-GCM-SIV may be distinguishable from an ideal AEAD. (This is based on standard assumptions about AES.)

The AEADs defined in this document calculate fresh AES keys for each nonce. This allows a larger number of plaintexts to be encrypted under a given key. Without this step, each SIV encryption would be like a standard GCM encryption with a random nonce. Since the nonce size for GCM is only 12 bytes, NIST set a limit [GCM] of 2^32 encryptions before the probability of duplicate nonces becomes too high.

The authors felt that, while large, 2^32 wasn't so large that this limit could be safely ignored. For example, consider encrypting the contents of a hard disk where the AEAD record size is 512 bytes, to match the traditional size of a disk sector. This process would have encrypted 2^32 records after processing 2TB, yet hard drives of multiple terabytes are now common.

Deriving fresh AES keys for each nonce alleviates this problem.

If the nonce is fixed then AES-GCM-SIV acts like AES-GCM with a random nonce, with the caveat that identical plaintexts will produce identical ciphertexts. However, we feel that the 2^32 limit for AES-GCM is too risky in a multi-key setting. Thus with AES-GCM-SIV we recommend that, for a specific key, a nonce not be repeated more than 2^8 times. (And, ideally, not be repeated at all.) See theorem six and figure four from the paper for detailed bounds.

Suzuki et al [multibirthday] show that even if nonces are selected uniformly at random, the probability that one or more values would be repeated 256 or more times is negligible until the number of nonces reaches 2^102. (Specifically the probability is 1/((2^96)^(255)) * Binomial(q, 256), where q is the number of nonces.) Since 2^102 is vastly greater than the limit on the number of plaintexts per key given above, we don't feel that this limit on the number of repeated nonces will be a problem. This also means that selecting nonces at random is a safe practice with AES-GCM-SIV.

In addition to calculating fresh AES keys for each nonce, these AEADs also calculate fresh POLYVAL keys. Previous versions of GCM-SIV did not do this and, instead, used part of the AEAD's key as the POLYVAL key. Bleichenbacher pointed out that this allowed an attacker who controlled the AEAD key to force the POLYVAL key to be zero. If a user of this AEAD authenticated messages with a secret additional-data value then this would be insecure as the attacker could calculate a valid authenticator without knowing the input. This does not violate the standard properties of an AEAD as the additional data is not assumed to be confidential. However, we want these AEADs to be robust to plausible misuse and also to be drop-in replacements for AES-GCM and so derive nonce-specific POLYVAL keys to avoid this issue.

A security analysis of a similar scheme appears in [GCM-SIV].

10. IANA Considerations

IANA is requested to add two entries to the registry of AEAD algorithms: AEAD_AES_128_GCM_SIV and AEAD_AES_256_GCM_SIV, both referencing this document as their specification.

11. Acknowledgements

The authors would like to thank Uri Blumenthal, Ondrej Mosnacek, Daniel Bleichenbacher, Kenny Paterson, Bart Preneel, John Mattsson and Deb Cooley's team at NSA Information Assurance for their helpful suggestions.

12. References

12.1. Normative References

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

12.2. Informative References

[AES-GCM-SIV]
Gueron, S., Langley, A., and Y. Lindell, "AES-GCM-SIV: specification and analysis", 2017, <https://eprint.iacr.org/2017/168>.

[GCM] Dworkin, M., "Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC", NIST SP-800-38D, November 2007, <http://csrc.nist.gov/publications/nistpubs/800-38D/SP-800-38D.pdf>.

[GCM-SIV] Gueron, S. and Y. Lindell, "GCM-SIV: Full Nonce Misuse-Resistant Authenticated Encryption at Under One Cycle Per Byte", Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security , 2015, <http://doi.acm.org/10.1145/2810103.2813613>.

[multibirthday]
Kazuhiro, S., Dongvu, T., Kaoru, K., and T. Koji, "Birthday Paradox for Multi-collisions", ICISC 2006: 9th International Conference, Busan, Korea, November 30 - December 1, 2006. Proceedings , 2006, <http://dx.doi.org/10.1007/11927587_5>.

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

Appendix A. The relationship between POLYVAL and GHASH

GHASH and POLYVAL both operate in GF(2^128), although with different irreducible polynomials: POLYVAL works modulo x^128 + x^127 + x^126 + x^121 + 1 and GHASH works modulo x^128 + x^7 + x^2 + x + 1. Note that these irreducible polynomials are the "reverse" of each other.

GHASH also has a different mapping between 128-bit strings and field elements. Where as POLYVAL takes the least-significant to most-significant bits of the first byte to be the coefficients of x^0 to x^7, GHASH takes them to be the coefficients of x^7 to x^0. This continues until, for the last byte, POLYVAL takes the least-significant to most-significant bits to be the coefficients of x^120 to x^127 while GHASH takes them to be the coefficients of x^127 to x^120.

The combination of these facts means that it's possible to "convert" values between the two by reversing the order of the bytes in a

16-byte string. The differing interpretations of bit order takes care of reversing the bits within each byte and then reversing the bytes does the rest. This may have a practical benefit for implementations that wish to implement both GHASH and POLYVAL.

In order to be clear which field a given operation is performed in, let mulX_GHASH be a function that takes a 16-byte string, converts it to an element of GHASH's field using GHASH's convention, multiplies it by x and converts back to a string. Likewise, let mulX_POLYVAL be a function that converts a 16-byte string to an element of POLYVAL's field using POLYVAL's convention, multiplies it by x and converts back.

Given the 16-byte string 01000000000000000000000000000000, mulX_GHASH of that string is 00800000000000000000000000000000 and mulX_POLYVAL of that string is 02000000000000000000000000000000. As a more general example, given 9c98c04df9387ded828175a92ba652d8, mulX_GHASH of that string is 4e4c6026fc9c3ef6c140bad495d3296c and mulX_POLYVAL of it is 3931819bf271fada0503eb52574ca5f2.

Lastly, let ByteReverse be the function that takes a 16-byte string and returns a copy where the order of the bytes has been reversed.

Now GHASH and POLYVAL can be defined in terms of one another:

POLYVAL(H, X_1, ..., X_n) =
ByteReverse(GHASH(mulX_GHASH(ByteReverse(H)), ByteReverse(X_1), ..., ByteReverse(X_n)))

GHASH(H, X_1, ..., X_n) =
ByteReverse(POLYVAL(mulX_POLYVAL(ByteReverse(H)), ByteReverse(X_1), ..., ByteReverse(X_n)))

As a worked example, let H = 25629347589242761d31f826ba4b757b, X_1 = 4f4f95668c83dfb6401762bb2d01a262 and X_2 = d1a24ddd2721d006bbe45f20d3c9f362. POLYVAL(H, X_1, X_2) = f7a3b47b846119fae5b7866cf5e5b77e. If we wished to calculate this given only an implementation of GHASH then the key for GHASH would be mulX_GHASH(ByteReverse(H)) = dcbaa5dd137c188ebb21492c23c9b112. Then ByteReverse(GHASH(dcba..., ByteReverse(X_1), ByteReverse(X_2))) = f7a3b47b846119fae5b7866cf5e5b77e, as required.

In the other direction, GHASH(H, X_1, X_2) = bd9b3997046731fb96251b91f9c99d7a. If we wished to calculate this given only an implementation of POLYVAL then we would first calculate the key for POLYVAL, mulX_POLYVAL(ByteReverse(H)), which is f6ea96744df0633aec8424b18e26c54a. Then ByteReverse(POLYVAL(f6ea...,

ByteReverse(X_1), ByteReverse(X_2))) =
bd9b3997046731fb96251b91f9c99d7a.

Appendix B. Additional comparisons with AES-GCM

Some, non-security, properties also differ between AES-GCM and AES-GCM-SIV that are worth noting:

AES-GCM allows plaintexts to be encrypted in a streaming fashion, i.e. the beginning of the plaintext can be encrypted and transmitted before the entire message has been processed. AES-GCM-SIV requires two passes for encryption and so cannot do this.

AES-GCM allows a constant additional-data input to be precomputed in order to save per-record computation. AES-GCM-SIV varies the authenticator key based on the nonce and so does not permit this.

The performance for AES-GCM vs AES-GCM-SIV on small machines can be roughly characterised by the number of AES operations and the number of GF(2^128) multiplications needed to process a message. Let a = (bytelen(additional-data) + 15) / 16 and p = (bytelen(plaintext) + 15) / 16. Then AES-GCM requires p + 1 AES operations and p + a + 1 field multiplications.

Defined similarly, AES-GCM-SIV with AES-128 requires p + 5 AES operations and p + a + 1 field multiplications. With AES-256 that becomes p + 7 AES operations.

With large machines, the available parallelism becomes far more important and such simple performance analysis is no longer representative. For such machines, we find that decryption of AES-GCM-SIV is only about 5% slower then AES-GCM, as long as the message is at least a couple of kilobytes. Encryption tends to be about 2/3's the speed because of the additional pass required.

Appendix C. Test vectors

C.1. AEAD_AES_128_GCM_SIV

```
Plaintext (0 bytes) =
AAD (0 bytes) =
Key =                          01000000000000000000000000000000
Nonce =                        030000000000000000000000
Record authentication key = d9b360279694941ac5dbc6987ada7377
Record encryption key =     4004a0dcd862f2a57360219d2d44ef6c
POLYVAL input =             00000000000000000000000000000000
POLYVAL result =            00000000000000000000000000000000
POLYVAL result XOR nonce =  03000000000000000000000000000000
```

```
   ... and masked =            03000000000000000000000000000000
   Tag =                       dc20e2d83f25705bb49e439eca56de25
   Initial counter =           dc20e2d83f25705bb49e439eca56dea5
   Result (16 bytes) =         dc20e2d83f25705bb49e439eca56de25


   Plaintext (8 bytes) =       0100000000000000
   AAD (0 bytes) =
   Key =                       01000000000000000000000000000000
   Nonce =                     030000000000000000000000
   Record authentication key = d9b360279694941ac5dbc6987ada7377
   Record encryption key =     4004a0dcd862f2a57360219d2d44ef6c
   POLYVAL input =             01000000000000000000000000000000
                               00000000000000004000000000000000
   POLYVAL result =            eb93b7740962c5e49d2a90a7dc5cec74
   POLYVAL result XOR nonce =  e893b7740962c5e49d2a90a7dc5cec74
   ... and masked =            e893b7740962c5e49d2a90a7dc5cec74
   Tag =                       578782fff6013b815b287c22493a364c
   Initial counter =           578782fff6013b815b287c22493a36cc
   Result (24 bytes) =         b5d839330ac7b786578782fff6013b81
                               5b287c22493a364c


   Plaintext (12 bytes) =      010000000000000000000000
   AAD (0 bytes) =
   Key =                       01000000000000000000000000000000
   Nonce =                     030000000000000000000000
   Record authentication key = d9b360279694941ac5dbc6987ada7377
   Record encryption key =     4004a0dcd862f2a57360219d2d44ef6c
   POLYVAL input =             01000000000000000000000000000000
                               00000000000000006000000000000000
   POLYVAL result =            48eb6c6c5a2dbe4a1dde508fee06361b
   POLYVAL result XOR nonce =  4beb6c6c5a2dbe4a1dde508fee06361b
   ... and masked =            4beb6c6c5a2dbe4a1dde508fee06361b
   Tag =                       a4978db357391a0bc4fdec8b0d106639
   Initial counter =           a4978db357391a0bc4fdec8b0d1066b9
   Result (28 bytes) =         7323ea61d05932260047d942a4978db3
                               57391a0bc4fdec8b0d106639


   Plaintext (16 bytes) =      01000000000000000000000000000000
   AAD (0 bytes) =
   Key =                       01000000000000000000000000000000
   Nonce =                     030000000000000000000000
   Record authentication key = d9b360279694941ac5dbc6987ada7377
   Record encryption key =     4004a0dcd862f2a57360219d2d44ef6c
   POLYVAL input =             01000000000000000000000000000000
                               00000000000000008000000000000000
```

```
   POLYVAL result =            20806c26e3c1de019e111255708031d6
   POLYVAL result XOR nonce =  23806c26e3c1de019e111255708031d6
   ... and masked =            23806c26e3c1de019e11125570803156
   Tag =                       303aaf90f6fe21199c6068577437a0c4
   Initial counter =           303aaf90f6fe21199c6068577437a0c4
   Result (32 bytes) =         743f7c8077ab25f8624e2e948579cf77
                               303aaf90f6fe21199c6068577437a0c4


   Plaintext (32 bytes) =      01000000000000000000000000000000
                               02000000000000000000000000000000
   AAD (0 bytes) =
   Key =                       01000000000000000000000000000000
   Nonce =                     030000000000000000000000
   Record authentication key = d9b360279694941ac5dbc6987ada7377
   Record encryption key =     4004a0dcd862f2a57360219d2d44ef6c
   POLYVAL input =             01000000000000000000000000000000
                               02000000000000000000000000000000
                               00000000000000000001000000000000
   POLYVAL result =            ce6edc9a50b36d9a98986bbf6a261c3b
   POLYVAL result XOR nonce =  cd6edc9a50b36d9a98986bbf6a261c3b
   ... and masked =            cd6edc9a50b36d9a98986bbf6a261c3b
   Tag =                       1a8e45dcd4578c667cd86847bf6155ff
   Initial counter =           1a8e45dcd4578c667cd86847bf6155ff
   Result (48 bytes) =         84e07e62ba83a6585417245d7ec413a9
                               fe427d6315c09b57ce45f2e3936a9445
                               1a8e45dcd4578c667cd86847bf6155ff


   Plaintext (48 bytes) =      01000000000000000000000000000000
                               02000000000000000000000000000000
                               03000000000000000000000000000000
   AAD (0 bytes) =
   Key =                       01000000000000000000000000000000
   Nonce =                     030000000000000000000000
   Record authentication key = d9b360279694941ac5dbc6987ada7377
   Record encryption key =     4004a0dcd862f2a57360219d2d44ef6c
   POLYVAL input =             01000000000000000000000000000000
                               02000000000000000000000000000000
                               03000000000000000000000000000000
                               00000000000000008001000000000000
   POLYVAL result =            81388746bc22d26b2abc3dcb15754222
   POLYVAL result XOR nonce =  82388746bc22d26b2abc3dcb15754222
   ... and masked =            82388746bc22d26b2abc3dcb15754222
   Tag =                       5e6e311dbf395d35b0fe39c2714388f8
   Initial counter =           5e6e311dbf395d35b0fe39c2714388f8
   Result (64 bytes) =         3fd24ce1f5a67b75bf2351f181a475c7
                               b800a5b4d3dcf70106b1eea82fa1d64d
```

```
                          f42bf7226122fa92e17a40eeaac1201b
                          5e6e311dbf395d35b0fe39c2714388f8

   Plaintext (64 bytes) = 01000000000000000000000000000000
                          02000000000000000000000000000000
                          03000000000000000000000000000000
                          04000000000000000000000000000000
   AAD (0 bytes) =
   Key =                  01000000000000000000000000000000
   Nonce =                030000000000000000000000
   Record authentication key = d9b360279694941ac5dbc6987ada7377
   Record encryption key =     4004a0dcd862f2a57360219d2d44ef6c
   POLYVAL input =        01000000000000000000000000000000
                          02000000000000000000000000000000
                          03000000000000000000000000000000
                          04000000000000000000000000000000
                          00000000000000000002000000000000
   POLYVAL result =       1e39b6d3344d348f6044f89935d1cf78
   POLYVAL result XOR nonce =  1d39b6d3344d348f6044f89935d1cf78
   ... and masked =       1d39b6d3344d348f6044f89935d1cf78
   Tag =                  8a263dd317aa88d56bdf3936dba75bb8
   Initial counter =      8a263dd317aa88d56bdf3936dba75bb8
   Result (80 bytes) =    2433668f1058190f6d43e360f4f35cd8
                          e475127cfca7028ea8ab5c20f7ab2af0
                          2516a2bdcbc08d521be37ff28c152bba
                          36697f25b4cd169c6590d1dd39566d3f
                          8a263dd317aa88d56bdf3936dba75bb8

   Plaintext (8 bytes) =  0200000000000000
   AAD (1 bytes) =        01
   Key =                  01000000000000000000000000000000
   Nonce =                030000000000000000000000
   Record authentication key = d9b360279694941ac5dbc6987ada7377
   Record encryption key =     4004a0dcd862f2a57360219d2d44ef6c
   POLYVAL input =        01000000000000000000000000000000
                          02000000000000000000000000000000
                          08000000000000004000000000000000
   POLYVAL result =       b26781e7e2c1376f96bec195f3709b2a
   POLYVAL result XOR nonce =  b16781e7e2c1376f96bec195f3709b2a
   ... and masked =       b16781e7e2c1376f96bec195f3709b2a
   Tag =                  3b0a1a2560969cdf790d99759abd1508
   Initial counter =      3b0a1a2560969cdf790d99759abd1588
   Result (24 bytes) =    1e6daba35669f4273b0a1a2560969cdf
                          790d99759abd1508
```

```
   Plaintext (12 bytes) =      020000000000000000000000
   AAD (1 bytes) =             01
   Key =                       0100000000000000000000000000000000
   Nonce =                     030000000000000000000000
   Record authentication key = d9b360279694941ac5dbc6987ada7377
   Record encryption key =     4004a0dcd862f2a57360219d2d44ef6c
   POLYVAL input =             0100000000000000000000000000000000
                               0200000000000000000000000000000000
                               0800000000000000600000000000000000
   POLYVAL result =            111f5affb18e4cc1164a01bdc12a4145
   POLYVAL result XOR nonce =  121f5affb18e4cc1164a01bdc12a4145
   ... and masked =            121f5affb18e4cc1164a01bdc12a4145
   Tag =                       08299c5102745aaa3a0c469fad9e075a
   Initial counter =           08299c5102745aaa3a0c469fad9e07da
   Result (28 bytes) =         296c7889fd99f41917f4462008299c51
                               02745aaa3a0c469fad9e075a


   Plaintext (16 bytes) =      02000000000000000000000000000000
   AAD (1 bytes) =             01
   Key =                       0100000000000000000000000000000000
   Nonce =                     030000000000000000000000
   Record authentication key = d9b360279694941ac5dbc6987ada7377
   Record encryption key =     4004a0dcd862f2a57360219d2d44ef6c
   POLYVAL input =             0100000000000000000000000000000000
                               0200000000000000000000000000000000
                               0800000000000000800000000000000000
   POLYVAL result =            79745ab508622c8a958543675fac4688
   POLYVAL result XOR nonce =  7a745ab508622c8a958543675fac4688
   ... and masked =            7a745ab508622c8a958543675fac4608
   Tag =                       8f8936ec039e4e4bb97ebd8c4457441f
   Initial counter =           8f8936ec039e4e4bb97ebd8c4457449f
   Result (32 bytes) =         e2b0c5da79a901c1745f700525cb335b
                               8f8936ec039e4e4bb97ebd8c4457441f


   Plaintext (32 bytes) =      02000000000000000000000000000000
                               03000000000000000000000000000000
   AAD (1 bytes) =             01
   Key =                       0100000000000000000000000000000000
   Nonce =                     030000000000000000000000
   Record authentication key = d9b360279694941ac5dbc6987ada7377
   Record encryption key =     4004a0dcd862f2a57360219d2d44ef6c
   POLYVAL input =             0100000000000000000000000000000000
                               0200000000000000000000000000000000
                               0300000000000000000000000000000000
                               0800000000000000001000000000000000
   POLYVAL result =            2ce7daaf7c89490822051255b12eca6b
```

```
   POLYVAL result XOR nonce =  2fe7daaf7c89490822051255b12eca6b
   ... and masked =            2fe7daaf7c89490822051255b12eca6b
   Tag =                       e6af6a7f87287da059a71684ed3498e1
   Initial counter =           e6af6a7f87287da059a71684ed3498e1
   Result (48 bytes) =         620048ef3c1e73e57e02bb8562c416a3
                               19e73e4caac8e96a1ecb2933145a1d71
                               e6af6a7f87287da059a71684ed3498e1


   Plaintext (48 bytes) =      020000000000000000000000000000000
                               030000000000000000000000000000000
                               040000000000000000000000000000000
   AAD (1 bytes) =             01
   Key =                       010000000000000000000000000000000
   Nonce =                     030000000000000000000000
   Record authentication key = d9b360279694941ac5dbc6987ada7377
   Record encryption key =     4004a0dcd862f2a57360219d2d44ef6c
   POLYVAL input =             010000000000000000000000000000000
                               020000000000000000000000000000000
                               030000000000000000000000000000000
                               040000000000000000000000000000000
                               080000000000000080010000000000000
   POLYVAL result =            9ca987715d69c1786711dfcd22f830fc
   POLYVAL result XOR nonce =  9fa987715d69c1786711dfcd22f830fc
   ... and masked =            9fa987715d69c1786711dfcd22f8307c
   Tag =                       6a8cc3865f76897c2e4b245cf31c51f2
   Initial counter =           6a8cc3865f76897c2e4b245cf31c51f2
   Result (64 bytes) =         50c8303ea93925d64090d07bd109dfd9
                               515a5a33431019c17d93465999a8b005
                               3201d723120a8562b838cdff25bf9d1e
                               6a8cc3865f76897c2e4b245cf31c51f2


   Plaintext (64 bytes) =      020000000000000000000000000000000
                               030000000000000000000000000000000
                               040000000000000000000000000000000
                               050000000000000000000000000000000
   AAD (1 bytes) =             01
   Key =                       010000000000000000000000000000000
   Nonce =                     030000000000000000000000
   Record authentication key = d9b360279694941ac5dbc6987ada7377
   Record encryption key =     4004a0dcd862f2a57360219d2d44ef6c
   POLYVAL input =             010000000000000000000000000000000
                               020000000000000000000000000000000
                               030000000000000000000000000000000
                               040000000000000000000000000000000
                               050000000000000000000000000000000
                               080000000000000000002000000000000
```

```
   POLYVAL result =            ffcd05d5770f34ad9267f0a59994b15a
   POLYVAL result XOR nonce =  fccd05d5770f34ad9267f0a59994b15a
   ... and masked =            fccd05d5770f34ad9267f0a59994b15a
   Tag =                       cdc46ae475563de037001ef84ae21744
   Initial counter =           cdc46ae475563de037001ef84ae217c4
   Result (80 bytes) =         2f5c64059db55ee0fb847ed513003746
                               aca4e61c711b5de2e7a77ffd02da42fe
                               ec601910d3467bb8b36ebbaebce5fba3
                               0d36c95f48a3e7980f0e7ac299332a80
                               cdc46ae475563de037001ef84ae21744


   Plaintext (4 bytes) =       02000000
   AAD (12 bytes) =            010000000000000000000000
   Key =                       01000000000000000000000000000000
   Nonce =                     030000000000000000000000
   Record authentication key = d9b360279694941ac5dbc6987ada7377
   Record encryption key =     4004a0dcd862f2a57360219d2d44ef6c
   POLYVAL input =             01000000000000000000000000000000
                               02000000000000000000000000000000
                               60000000000000002000000000000000
   POLYVAL result =            f6ce9d3dcd68a2fd603c7ecc18fb9918
   POLYVAL result XOR nonce =  f5ce9d3dcd68a2fd603c7ecc18fb9918
   ... and masked =            f5ce9d3dcd68a2fd603c7ecc18fb9918
   Tag =                       07eb1f84fb28f8cb73de8e99e2f48a14
   Initial counter =           07eb1f84fb28f8cb73de8e99e2f48a94
   Result (20 bytes) =         a8fe3e8707eb1f84fb28f8cb73de8e99
                               e2f48a14


   Plaintext (20 bytes) =      03000000000000000000000000000000
                               04000000
   AAD (18 bytes) =            01000000000000000000000000000000
                               0200
   Key =                       01000000000000000000000000000000
   Nonce =                     030000000000000000000000
   Record authentication key = d9b360279694941ac5dbc6987ada7377
   Record encryption key =     4004a0dcd862f2a57360219d2d44ef6c
   POLYVAL input =             01000000000000000000000000000000
                               02000000000000000000000000000000
                               03000000000000000000000000000000
                               04000000000000000000000000000000
                               90000000000000a000000000000000
   POLYVAL result =            4781d492cb8f926c504caa36f61008fe
   POLYVAL result XOR nonce =  4481d492cb8f926c504caa36f61008fe
   ... and masked =            4481d492cb8f926c504caa36f610087e
   Tag =                       24afc9805e976f451e6d87f6fe106514
   Initial counter =           24afc9805e976f451e6d87f6fe106594
```

```
   Result (36 bytes) =         6bb0fecf5ded9b77f902c7d5da236a43
                               91dd029724afc9805e976f451e6d87f6
                               fe106514


   Plaintext (18 bytes) =      030000000000000000000000000000000
                               0400
   AAD (20 bytes) =            010000000000000000000000000000000
                               02000000
   Key =                       010000000000000000000000000000000
   Nonce =                     03000000000000000000000
   Record authentication key = d9b360279694941ac5dbc6987ada7377
   Record encryption key =     4004a0dcd862f2a57360219d2d44ef6c
   POLYVAL input =             010000000000000000000000000000000
                               020000000000000000000000000000000
                               030000000000000000000000000000000
                               040000000000000000000000000000000
                               a000000000000009000000000000000
   POLYVAL result =            75cbc23a1a10e348aeb8e384b5cc79fd
   POLYVAL result XOR nonce =  76cbc23a1a10e348aeb8e384b5cc79fd
   ... and masked =            76cbc23a1a10e348aeb8e384b5cc797d
   Tag =                       bff9b2ef00fb47920cc72a0c0f13b9fd
   Initial counter =           bff9b2ef00fb47920cc72a0c0f13b9fd
   Result (34 bytes) =         44d0aaf6fb2f1f34add5e8064e83e12a
                               2adabff9b2ef00fb47920cc72a0c0f13
                               b9fd
```

C.2. AEAD_AES_256_GCM_SIV

```
   Plaintext (0 bytes) =
   AAD (0 bytes) =
   Key =                       010000000000000000000000000000000
                               000000000000000000000000000000000
   Nonce =                     03000000000000000000000
   Record authentication key = b5d3c529dfafac43136d2d11be284d7f
   Record encryption key =     b914f4742be9e1d7a2f84addbf96dec3
                               456e3c6c05ecc157cdbf0700fedad222
   POLYVAL input =             000000000000000000000000000000000
   POLYVAL result =            000000000000000000000000000000000
   POLYVAL result XOR nonce =  030000000000000000000000000000000
   ... and masked =            030000000000000000000000000000000
   Tag =                       07f5f4169bbf55a8400cd47ea6fd400f
   Initial counter =           07f5f4169bbf55a8400cd47ea6fd408f
   Result (16 bytes) =         07f5f4169bbf55a8400cd47ea6fd400f


   Plaintext (8 bytes) =       0100000000000000
   AAD (0 bytes) =
```

```
   Key =                        01000000000000000000000000000000
                                00000000000000000000000000000000
   Nonce =                      030000000000000000000000
   Record authentication key = b5d3c529dfafac43136d2d11be284d7f
   Record encryption key =     b914f4742be9e1d7a2f84addbf96dec3
                               456e3c6c05ecc157cdbf0700fedad222
   POLYVAL input =             01000000000000000000000000000000
                               00000000000000004000000000000000
   POLYVAL result =            05230f62f0eac8aa14fe4d646b59cd41
   POLYVAL result XOR nonce =  06230f62f0eac8aa14fe4d646b59cd41
   ... and masked =            06230f62f0eac8aa14fe4d646b59cd41
   Tag =                       843122130f7364b761e0b97427e3df28
   Initial counter =           843122130f7364b761e0b97427e3dfa8
   Result (24 bytes) =         c2ef328e5c71c83b843122130f7364b7
                               61e0b97427e3df28


   Plaintext (12 bytes) =      010000000000000000000000
   AAD (0 bytes) =
   Key =                       01000000000000000000000000000000
                               00000000000000000000000000000000
   Nonce =                     030000000000000000000000
   Record authentication key = b5d3c529dfafac43136d2d11be284d7f
   Record encryption key =     b914f4742be9e1d7a2f84addbf96dec3
                               456e3c6c05ecc157cdbf0700fedad222
   POLYVAL input =             01000000000000000000000000000000
                               00000000000000006000000000000000
   POLYVAL result =            6d81a24732fd6d03ae5af544720a1c13
   POLYVAL result XOR nonce =  6e81a24732fd6d03ae5af544720a1c13
   ... and masked =            6e81a24732fd6d03ae5af544720a1c13
   Tag =                       8ca50da9ae6559e48fd10f6e5c9ca17e
   Initial counter =           8ca50da9ae6559e48fd10f6e5c9ca1fe
   Result (28 bytes) =         9aab2aeb3faa0a34aea8e2b18ca50da9
                               ae6559e48fd10f6e5c9ca17e


   Plaintext (16 bytes) =      01000000000000000000000000000000
   AAD (0 bytes) =
   Key =                       01000000000000000000000000000000
                               00000000000000000000000000000000
   Nonce =                     030000000000000000000000
   Record authentication key = b5d3c529dfafac43136d2d11be284d7f
   Record encryption key =     b914f4742be9e1d7a2f84addbf96dec3
                               456e3c6c05ecc157cdbf0700fedad222
   POLYVAL input =             01000000000000000000000000000000
                               00000000000000008000000000000000
   POLYVAL result =            74eee2bf7c9a165f8b25dea73db32a6d
   POLYVAL result XOR nonce =  77eee2bf7c9a165f8b25dea73db32a6d
```

```
   ... and masked =           77eee2bf7c9a165f8b25dea73db32a6d
   Tag =                      c9eac6fa700942702e90862383c6c366
   Initial counter =          c9eac6fa700942702e90862383c6c3e6
   Result (32 bytes) =        85a01b63025ba19b7fd3ddfc033b3e76
                              c9eac6fa700942702e90862383c6c366


   Plaintext (32 bytes) =     01000000000000000000000000000000
                              02000000000000000000000000000000
   AAD (0 bytes) =
   Key =                      01000000000000000000000000000000
                              00000000000000000000000000000000
   Nonce =                    030000000000000000000000
   Record authentication key = b5d3c529dfafac43136d2d11be284d7f
   Record encryption key =    b914f4742be9e1d7a2f84addbf96dec3
                              456e3c6c05ecc157cdbf0700fedad222
   POLYVAL input =            01000000000000000000000000000000
                              02000000000000000000000000000000
                              00000000000000000001000000000000
   POLYVAL result =           899b6381b3d46f0def7aa0517ba188f5
   POLYVAL result XOR nonce = 8a9b6381b3d46f0def7aa0517ba188f5
   ... and masked =           8a9b6381b3d46f0def7aa0517ba18875
   Tag =                      e819e63abcd020b006a976397632eb5d
   Initial counter =          e819e63abcd020b006a976397632ebdd
   Result (48 bytes) =        4a6a9db4c8c6549201b9edb53006cba8
                              21ec9cf850948a7c86c68ac7539d027f
                              e819e63abcd020b006a976397632eb5d


   Plaintext (48 bytes) =     01000000000000000000000000000000
                              02000000000000000000000000000000
                              03000000000000000000000000000000
   AAD (0 bytes) =
   Key =                      01000000000000000000000000000000
                              00000000000000000000000000000000
   Nonce =                    030000000000000000000000
   Record authentication key = b5d3c529dfafac43136d2d11be284d7f
   Record encryption key =    b914f4742be9e1d7a2f84addbf96dec3
                              456e3c6c05ecc157cdbf0700fedad222
   POLYVAL input =            01000000000000000000000000000000
                              02000000000000000000000000000000
                              03000000000000000000000000000000
                              00000000000000008001000000000000
   POLYVAL result =           c1f8593d8fc29b0c290cae1992f71f51
   POLYVAL result XOR nonce = c2f8593d8fc29b0c290cae1992f71f51
   ... and masked =           c2f8593d8fc29b0c290cae1992f71f51
   Tag =                      790bc96880a99ba804bd12c0e6a22cc4
   Initial counter =          790bc96880a99ba804bd12c0e6a22cc4
```

```
   Result (64 bytes) =         c00d121893a9fa603f48ccc1ca3c57ce
                               7499245ea0046db16c53c7c66fe717e3
                               9cf6c748837b61f6ee3adcee17534ed5
                               790bc96880a99ba804bd12c0e6a22cc4


   Plaintext (64 bytes) =      0100000000000000000000000000000000
                               0200000000000000000000000000000000
                               0300000000000000000000000000000000
                               0400000000000000000000000000000000
   AAD (0 bytes) =
   Key =                       0100000000000000000000000000000000
                               0000000000000000000000000000000000
   Nonce =                     030000000000000000000000
   Record authentication key = b5d3c529dfafac43136d2d11be284d7f
   Record encryption key =     b914f4742be9e1d7a2f84addbf96dec3
                               456e3c6c05ecc157cdbf0700fedad222
   POLYVAL input =             0100000000000000000000000000000000
                               0200000000000000000000000000000000
                               0300000000000000000000000000000000
                               0400000000000000000000000000000000
                               0000000000000000000020000000000000
   POLYVAL result =            6ef38b06046c7c0e225efaef8e2ec4c4
   POLYVAL result XOR nonce =  6df38b06046c7c0e225efaef8e2ec4c4
   ... and masked =            6df38b06046c7c0e225efaef8e2ec444
   Tag =                       112864c269fc0d9d88c61fa47e39aa08
   Initial counter =           112864c269fc0d9d88c61fa47e39aa88
   Result (80 bytes) =         c2d5160a1f8683834910acdafc41fbb1
                               632d4a353e8b905ec9a5499ac34f96c7
                               e1049eb080883891a4db8caaa1f99dd0
                               04d80487540735234e3744512c6f90ce
                               112864c269fc0d9d88c61fa47e39aa08


   Plaintext (8 bytes) =       0200000000000000
   AAD (1 bytes) =             01
   Key =                       0100000000000000000000000000000000
                               0000000000000000000000000000000000
   Nonce =                     030000000000000000000000
   Record authentication key = b5d3c529dfafac43136d2d11be284d7f
   Record encryption key =     b914f4742be9e1d7a2f84addbf96dec3
                               456e3c6c05ecc157cdbf0700fedad222
   POLYVAL input =             0100000000000000000000000000000000
                               0200000000000000000000000000000000
                               0800000000000000400000000000000000
   POLYVAL result =            34e57bafe011b9b36fc6821b7ffb3354
   POLYVAL result XOR nonce =  37e57bafe011b9b36fc6821b7ffb3354
   ... and masked =            37e57bafe011b9b36fc6821b7ffb3354
```

```
   Tag =                        91213f267e3b452f02d01ae33e4ec854
   Initial counter =            91213f267e3b452f02d01ae33e4ec8d4
   Result (24 bytes) =          1de22967237a813291213f267e3b452f
                                02d01ae33e4ec854


   Plaintext (12 bytes) =       020000000000000000000000
   AAD (1 bytes) =              01
   Key =                        0100000000000000000000000000000000
                                0000000000000000000000000000000000
   Nonce =                      030000000000000000000000
   Record authentication key = b5d3c529dfafac43136d2d11be284d7f
   Record encryption key =      b914f4742be9e1d7a2f84addbf96dec3
                                456e3c6c05ecc157cdbf0700fedad222
   POLYVAL input =              0100000000000000000000000000000000
                                0200000000000000000000000000000000
                                0800000000000006000000000000000000
   POLYVAL result =             5c47d68a22061c1ad5623a3b66a8e206
   POLYVAL result XOR nonce =   5f47d68a22061c1ad5623a3b66a8e206
   ... and masked =             5f47d68a22061c1ad5623a3b66a8e206
   Tag =                        c1a4a19ae800941ccdc57cc8413c277f
   Initial counter =            c1a4a19ae800941ccdc57cc8413c27ff
   Result (28 bytes) =          163d6f9cc1b346cd453a2e4cc1a4a19a
                                e800941ccdc57cc8413c277f


   Plaintext (16 bytes) =       02000000000000000000000000000000
   AAD (1 bytes) =              01
   Key =                        0100000000000000000000000000000000
                                0000000000000000000000000000000000
   Nonce =                      030000000000000000000000
   Record authentication key = b5d3c529dfafac43136d2d11be284d7f
   Record encryption key =      b914f4742be9e1d7a2f84addbf96dec3
                                456e3c6c05ecc157cdbf0700fedad222
   POLYVAL input =              0100000000000000000000000000000000
                                0200000000000000000000000000000000
                                0800000000000008000000000000000000
   POLYVAL result =             452896726c616746f01d11d82911d478
   POLYVAL result XOR nonce =   462896726c616746f01d11d82911d478
   ... and masked =             462896726c616746f01d11d82911d478
   Tag =                        b292d28ff61189e8e49f3875ef91aff7
   Initial counter =            b292d28ff61189e8e49f3875ef91aff7
   Result (32 bytes) =          c91545823cc24f17dbb0e9e807d5ec17
                                b292d28ff61189e8e49f3875ef91aff7


   Plaintext (32 bytes) =       02000000000000000000000000000000
                                03000000000000000000000000000000
```

```
   AAD (1 bytes) =             01
   Key =                       0100000000000000000000000000000000
                               0000000000000000000000000000000000
   Nonce =                     030000000000000000000000
   Record authentication key = b5d3c529dfafac43136d2d11be284d7f
   Record encryption key =     b914f4742be9e1d7a2f84addbf96dec3
                               456e3c6c05ecc157cdbf0700fedad222
   POLYVAL input =             0100000000000000000000000000000000
                               0200000000000000000000000000000000
                               0300000000000000000000000000000000
                               0800000000000000001000000000000000
   POLYVAL result =            4e58c1e341c9bb0ae34eda9509dfc90c
   POLYVAL result XOR nonce =  4d58c1e341c9bb0ae34eda9509dfc90c
   ... and masked =            4d58c1e341c9bb0ae34eda9509dfc90c
   Tag =                       aea1bad12702e1965604374aab96dbbc
   Initial counter =           aea1bad12702e1965604374aab96dbbc
   Result (48 bytes) =         07dad364bfc2b9da89116d7bef6daaaf
                               6f255510aa654f920ac81b94e8bad365
                               aea1bad12702e1965604374aab96dbbc


   Plaintext (48 bytes) =      0200000000000000000000000000000000
                               0300000000000000000000000000000000
                               0400000000000000000000000000000000
   AAD (1 bytes) =             01
   Key =                       0100000000000000000000000000000000
                               0000000000000000000000000000000000
   Nonce =                     030000000000000000000000
   Record authentication key = b5d3c529dfafac43136d2d11be284d7f
   Record encryption key =     b914f4742be9e1d7a2f84addbf96dec3
                               456e3c6c05ecc157cdbf0700fedad222
   POLYVAL input =             0100000000000000000000000000000000
                               0200000000000000000000000000000000
                               0300000000000000000000000000000000
                               0400000000000000000000000000000000
                               0800000000000000080010000000000000
   POLYVAL result =            2566a4aff9a525df9772c16d4eaf8d2a
   POLYVAL result XOR nonce =  2666a4aff9a525df9772c16d4eaf8d2a
   ... and masked =            2666a4aff9a525df9772c16d4eaf8d2a
   Tag =                       03332742b228c647173616cfd44c54eb
   Initial counter =           03332742b228c647173616cfd44c54eb
   Result (64 bytes) =         c67a1f0f567a5198aa1fcc8e3f213143
                               36f7f51ca8b1af61feac35a86416fa47
                               fbca3b5f749cdf564527f2314f42fe25
                               03332742b228c647173616cfd44c54eb


   Plaintext (64 bytes) =      0200000000000000000000000000000000
```

```
                          03000000000000000000000000000000
                          04000000000000000000000000000000
                          05000000000000000000000000000000
   AAD (1 bytes) =        01
   Key =                  01000000000000000000000000000000
                          00000000000000000000000000000000
   Nonce =                030000000000000000000000
   Record authentication key = b5d3c529dfafac43136d2d11be284d7f
   Record encryption key =    b914f4742be9e1d7a2f84addbf96dec3
                          456e3c6c05ecc157cdbf0700fedad222
   POLYVAL input =        01000000000000000000000000000000
                          02000000000000000000000000000000
                          03000000000000000000000000000000
                          04000000000000000000000000000000
                          05000000000000000000000000000000
                          08000000000000000002000000000000
   POLYVAL result =       da58d2f61b0a9d343b2f37fb0c519733
   POLYVAL result XOR nonce = d958d2f61b0a9d343b2f37fb0c519733
   ... and masked =       d958d2f61b0a9d343b2f37fb0c519733
   Tag =                  5bde0285037c5de81e5b570a049b62a0
   Initial counter =      5bde0285037c5de81e5b570a049b62a0
   Result (80 bytes) =    67fd45e126bfb9a79930c43aad2d3696
                          7d3f0e4d217c1e551f59727870beefc9
                          8cb933a8fce9de887b1e40799988db1f
                          c3f91880ed405b2dd298318858467c89
                          5bde0285037c5de81e5b570a049b62a0


   Plaintext (4 bytes) =  02000000
   AAD (12 bytes) =       010000000000000000000000
   Key =                  01000000000000000000000000000000
                          00000000000000000000000000000000
   Nonce =                030000000000000000000000
   Record authentication key = b5d3c529dfafac43136d2d11be284d7f
   Record encryption key =    b914f4742be9e1d7a2f84addbf96dec3
                          456e3c6c05ecc157cdbf0700fedad222
   POLYVAL input =        01000000000000000000000000000000
                          02000000000000000000000000000000
                          60000000000000002000000000000000
   POLYVAL result =       6dc76ae84b88916e073a303aafde05cf
   POLYVAL result XOR nonce = 6ec76ae84b88916e073a303aafde05cf
   ... and masked =       6ec76ae84b88916e073a303aafde054f
   Tag =                  1835e517741dfddccfa07fa4661b74cf
   Initial counter =      1835e517741dfddccfa07fa4661b74cf
   Result (20 bytes) =    22b3f4cd1835e517741dfddccfa07fa4
                          661b74cf
```

```
   Plaintext (20 bytes) =      03000000000000000000000000000000
                               04000000
   AAD (18 bytes) =            01000000000000000000000000000000
                               0200
   Key =                       01000000000000000000000000000000
                               00000000000000000000000000000000
   Nonce =                     030000000000000000000000
   Record authentication key = b5d3c529dfafac43136d2d11be284d7f
   Record encryption key =     b914f4742be9e1d7a2f84addbf96dec3
                               456e3c6c05ecc157cdbf0700fedad222
   POLYVAL input =             01000000000000000000000000000000
                               02000000000000000000000000000000
                               03000000000000000000000000000000
                               04000000000000000000000000000000
                               90000000000000a000000000000000
   POLYVAL result =            973ef4fd04bd31d193816ab26f8655ca
   POLYVAL result XOR nonce =  943ef4fd04bd31d193816ab26f8655ca
   ... and masked =            943ef4fd04bd31d193816ab26f86554a
   Tag =                       b879ad976d8242acc188ab59cabfe307
   Initial counter =           b879ad976d8242acc188ab59cabfe387
   Result (36 bytes) =         43dd0163cdb48f9fe3212bf61b201976
                               067f342bb879ad976d8242acc188ab59
                               cabfe307


   Plaintext (18 bytes) =      03000000000000000000000000000000
                               0400
   AAD (20 bytes) =            01000000000000000000000000000000
                               02000000
   Key =                       01000000000000000000000000000000
                               00000000000000000000000000000000
   Nonce =                     030000000000000000000000
   Record authentication key = b5d3c529dfafac43136d2d11be284d7f
   Record encryption key =     b914f4742be9e1d7a2f84addbf96dec3
                               456e3c6c05ecc157cdbf0700fedad222
   POLYVAL input =             01000000000000000000000000000000
                               02000000000000000000000000000000
                               03000000000000000000000000000000
                               04000000000000000000000000000000
                               a000000000000009000000000000000
   POLYVAL result =            2cbb6b7ab2dbffefb797f825f826870c
   POLYVAL result XOR nonce =  2fbb6b7ab2dbffefb797f825f826870c
   ... and masked =            2fbb6b7ab2dbffefb797f825f826870c
   Tag =                       cfcdf5042112aa29685c912fc2056543
   Initial counter =           cfcdf5042112aa29685c912fc20565c3
   Result (34 bytes) =         462401724b5ce6588d5a54aae5375513
                               a075cfcdf5042112aa29685c912fc205
                               6543
```

C.3. Counter wrap tests

The tests in this section use AEAD_AES_256_GCM_SIV and are crafted to test correct wrapping of the block counter.

```
Plaintext (32 bytes) =      0000000000000000000000000000000000
                            4db923dc793ee6497c76dcc03a98e108
AAD (0 bytes) =
Key =                       0000000000000000000000000000000000
                            0000000000000000000000000000000000
Nonce =                     000000000000000000000000
Record authentication key = dc95c078a24089895275f3d86b4fb868
Record encryption key =     779b38d15bffb63d39d6e9ae76a9b2f3
                            75d11b0e3a68c422845c7d4690fa594f
POLYVAL input =             0000000000000000000000000000000000
                            4db923dc793ee6497c76dcc03a98e108
                            0000000000000000001000000000000000
POLYVAL result =            7367cdb411b730128dd56e8edc0eff56
POLYVAL result XOR nonce =  7367cdb411b730128dd56e8edc0eff56
... and masked =            7367cdb411b730128dd56e8edc0eff56
Tag =                       ffffffff0000000000000000000000000
Initial counter =           ffffffff0000000000000000000000080
Result (48 bytes) =         f3f80f2cf0cb2dd9c5984fcda908456c
                            c537703b5ba70324a6793a7bf218d3ea
                            ffffffff0000000000000000000000000


Plaintext (24 bytes) =      eb3640277c7ffd1303c7a542d02d3e4c
                            0000000000000000
AAD (0 bytes) =
Key =                       0000000000000000000000000000000000
                            0000000000000000000000000000000000
Nonce =                     000000000000000000000000
Record authentication key = dc95c078a24089895275f3d86b4fb868
Record encryption key =     779b38d15bffb63d39d6e9ae76a9b2f3
                            75d11b0e3a68c422845c7d4690fa594f
POLYVAL input =             eb3640277c7ffd1303c7a542d02d3e4c
                            0000000000000000000000000000000000
                            0000000000000000c000000000000000
POLYVAL result =            7367cdb411b730128dd56e8edc0eff56
POLYVAL result XOR nonce =  7367cdb411b730128dd56e8edc0eff56
... and masked =            7367cdb411b730128dd56e8edc0eff56
Tag =                       ffffffff0000000000000000000000000
Initial counter =           ffffffff0000000000000000000000080
Result (40 bytes) =         18ce4f0b8cb4d0cac65fea8f79257b20
                            888e53e72299e56dffffffff00000000
                            0000000000000000
```

Authors' Addresses

Shay Gueron
University of Haifa and Amazon Web Services
Abba Khoushy Ave 199
Haifa 3498838
Israel

Email: shay@math.haifa.ac.il

Adam Langley
Google
345 Spear St
San Francisco, CA 94105
US

Email: agl@google.com

Yehuda Lindell
Bar Ilan University
Ramat Gan
5290002
Israel

Email: Yehuda.Lindell@biu.ac.il