# Key Recovery Attacks on AES-GCM-SIV

## Summary of Attacks

The Crypto Forum Research Group (CFRG) is currently considering the AES-GCM-SIV authenticated encryption scheme [1] as an Internet Engineering Task Force (IETF) standard. In this scheme the master key K is 128 bits or 256 bits in length. Each message uses a 128-bit nonce N, which is used with the master key to generate a record authentication key H (128 bits) and a record encryption key L (128 or 256 bits). While nonces should be unique, AES-GCM-SIV is claimed to have some level of nonce misuse resistance. Up to $2^{32}$ messages are theoretically allowed to be encrypted under a fixed key and nonce.

In this section we summarize the resource requirements for three cryptanalytic attacks on AES-GCM-SIV under a fixed master key. Details of the attacks appear in later sections. The attacks subexhaustively recover the record encryption key and record authentication key for multiple nonces. With L and H in hand for a given N, the attacker (despite not knowing K) can encrypt/decrypt any messages that use N, and can also forge arbitrary messages under N. The first and second attacks are birthday-style attacks which search for random collisions in keys. Both attacks exploit the fact that for each nonce a different record encryption key is used. The third attack, which is only applicable to 256-bit keys, exploits both the key generation method and the fact that AES is likely to have at least one fixed point. Furthermore, the key generation method for 256-bit keys has the property that if a record encryption and authentication key pair (L,H) is compromised, then many additional key pairs can be recovered, each with one query and only 128 bits of work.

The first attack only requires a nonce to be repeated at most two times. The attack involves two parameters, s and t, where $s > t \geq 0$. Table 1 shows the resource requirements for the attack for various values of (s,t), assuming K is 128 bits in length. 256-bit keys can be exploited as well; while the work increases by a factor of $2^{128}$, significantly more (L,H) pairs can be recovered because of the property mentioned above.

| (s,t) | queries ($2^s+2^{t+1}$) | memory ($2^s$) | work ($2^{128-s+t}$) | forgery attempts ($2^t$) | recovered keys ($2^t$ L's, $2^{t+1}$ H's) |
|---|---|---|---|---|---|
| (1,0) | $2^2$ | $2^1$ | $2^{127}$ | 1 | (1,2) |
| (16,0) | $2^{16}$ | $2^{16}$ | $2^{112}$ | 1 | (1,2) |
| (32,0) | $2^{32}$ | $2^{32}$ | $2^{96}$ | 1 | (1,2) |
| (32,16) | $2^{32}$ | $2^{32}$ | $2^{112}$ | $2^{16}$ | ($2^{16}$, $2^{17}$) |
| (48,0) | $2^{48}$ | $2^{48}$ | $2^{80}$ | 1 | (1,2) |
| (48,32) | $2^{48}$ | $2^{48}$ | $2^{112}$ | $2^{32}$ | ($2^{32}$,$2^{33}$) |
| (64,0) | $2^{64}$ | $2^{64}$ | $2^{64}$ | 1 | (1,2) |
| (64,48) | $2^{64}$ | $2^{64}$ | $2^{112}$ | $2^{48}$ | ($2^{48}$,$2^{49}$) |

**Table 1: Resource requirements for first attack on AES-GCM-SIV with a 128-bit key**

The second attack does not require any nonces to be repeated, but has significantly larger data and work requirements. It involves two parameters, m and n, where $m \geq n > 0$. Table 2 shows the resource requirements for the attack for various values of (m,n), assuming K is 128 bits in length. In the table, we use the expression $Q[x,r] \approx (r!)^{1/r}\,\Gamma(1+1/r)\,x^{1-1/r}$, where $\Gamma$ is the gamma function. 256-bit keys can be exploited as well; while the work increases by a factor of $2^{128}$, significantly more (L,H) pairs can be recovered, again, because of the property mentioned previously.

| (m,n) | queries/memory $Q[2^{127},2^{m}]$ | work ($2^{128-n}$) | forgery attempts ($2^{m-n}$) | recovered keys ($2^{m-n}$ L's, $2^{m-n+1}$ H's) |
|---|---|---|---|---|
| (1,1) | $2^{63.8}$ | $2^{127}$ | 1 | (1,2) |
| (2,1) | $2^{96.3}$ | $2^{127}$ | $2^{1}$ | $(2^{1},2^{2})$ |
| (2,2) | $2^{96.3}$ | $2^{126}$ | 1 | (1,2) |
| (3,1) | $2^{113.0}$ | $2^{127}$ | $2^{2}$ | $(2^{2},2^{3})$ |
| (3,3) | $2^{113.0}$ | $2^{125}$ | 1 | (1,2) |
| (4,1) | $2^{121.8}$ | $2^{127}$ | $2^{3}$ | $(2^{3},2^{4})$ |
| (4,4) | $2^{121.8}$ | $2^{124}$ | 1 | (1,2) |
| (5,1) | $2^{126.7}$ | $2^{127}$ | $2^{4}$ | $(2^{4},2^{5})$ |
| (5,5) | $2^{126.7}$ | $2^{123}$ | 1 | (1,2) |

**Table 2: Resource requirements for second attack on AES-GCM-SIV with a 128-bit key**

The third attack does not require any nonces to be repeated, but is only applicable to 256-bit keys. It has an extremely large data requirement ($2^{128}$ queries), but unlike the first two attacks for 256-bit keys, the work is $\leq 2^{128}$ (i.e., the work is no more than the square-root of the work to exhaust).

These attacks assume that all of the required data is collected under a single master key. It should be stressed, however, that limiting the amount of data encrypted under a single master key or changing the master key more frequently does not reduce the success probability or increase the cost of these attacks. If the total amount of required data is collected across multiple master keys, the attacks will still succeed in recovering (L,H,N) triples under one or more of these master keys.

## Description of AES-GCM-SIV

We give a brief description of AES-GCM-SIV as specified in [1]. Let N denote a 128-bit nonce and K denote a 128-bit or 256-bit AES key. Let $P = P_0, P_1, \ldots$ denote a plaintext message and $A = A_0, A_1, \ldots$ denote additional authenticated data, divided into 128-bit blocks. P and A are padded with zero bits until they are each a multiple of 16 bytes. Let $E_K(X)$ denote the AES encryption of a 128-bit block X with key K. Let + denote the bitwise XOR operation, & denote the bitwise AND operation, and || denote concatenation. First, compute a 128-bit record authentication key $H = E_K(N)$. Then, if K is 128 bits in length, compute a 128-bit record encryption key $L = E_K(H)$. Otherwise compute a 256-bit record encryption key $L = E_K(H)||E_K(E_K(H))$. The 128-bit tag is computed as

$$T = E_L(\mathrm{POLYVAL}(H,A,P,\ell)\ \&\ \texttt{0x7fffffffffffffff}),$$

where $\ell = \mathrm{len}(P)||\mathrm{len}(A)$ is a 128-bit length block comprised of the 64-bit encodings of the bit lengths of A and P (before padding) and POLYVAL is a $GF(2^{128})$-based polynomial authenticator similar to the GHASH authenticator used in GCM. Let $T_{95}$ denote the 95 bits of T to the immediate right of the most significant bit, $T_{32}$ denote the low 32 bits of T, $+_{32}$ denote addition modulo $2^{32}$, and k denote a 32-bit counter starting at 0. The kth block of ciphertext is computed via counter mode as

$$C_k = P_k + E_L(1||T_{95}||(T_{32} +_{32} k))$$

for $0 \leq k < 2^{32}$. The last ciphertext block is truncated to the length of the last (unpadded) plaintext block. The ciphertext blocks and tag are transmitted to the receiver along with the nonce and the (unpadded) additional authenticated data.

**First Attack (128-bit Keys)**

In this section we describe the attack for the case of 128-bit keys K. The attack consists of a data collection phase and an offline phase. Fix both the additional authenticated data and the plaintext to be the empty string. That is, $A = \emptyset$ and $P = \emptyset$. These are valid input strings for AES-GCM-SIV, and a test vector of this type is given in [1] for each of the two key sizes. Since the lengths of these strings are already a multiple of 16 bytes, namely 0 bytes, no padding is performed. Moreover, the length block consists of 128 zero bits, that is, $\ell = 0^{128}$.

For the data collection phase of the attack we ask for the tags associated with $(A,P) = (\emptyset,\emptyset)$ under $2^s$ distinct nonces $N_i$. Since each of A and P remains the empty string after padding, and since $\ell = 0^{128}$, it follows that

$$\mathrm{POLYVAL}(H_i,A,P,\ell) = \mathrm{POLYVAL}(H_i,\emptyset,\emptyset,0^{128}) = 0^{128},$$

regardless of the value of the record authentication key $H_i = E_K(N_i)$. (A test vector for $(A,P) = (\emptyset,\emptyset)$ is given in [1] for each of the two key sizes, which confirms this.) The tags are generated by setting the most significant bit to 0 and encrypting the result with the record encryption key $L_i = E_K(H_i)$. That is, $T_i = E_{L_i}(0^{128})$. We store the $2^s$ tag/nonce pairs $(T_i,N_i)$ in a table (sorted on the tags). In the offline phase of the attack we compute $T_j = E_{L_j}(0^{128})$ for $2^{128-s+t}$ distinct values of $L_j$ (with $s > t$). We expect $2^{s+(128-s+t)-128} = 2^t$ distinct matches of the form $L_i = E_K(E_K(N_i)) = L_j$, and we can detect these events by observing a match on the corresponding tags. A (causal) match on a tag tells us that $L_j$ is the record encryption key for nonce $N_i$.

Note that we expect $2^t$ random matches as well. Together, the sets of random and causal matches could involve up to $2^{t+1}$ distinct nonces. To distinguish the causal matches from the random matches, we next ask for the encryptions of (A,P) under these nonces, where A is arbitrary and $P \neq \emptyset$. Thus, up to $2^{t+1}$ nonces are repeated a total of two times in this attack. Since $P \neq \emptyset$, we can use the counter mode equations associated with the $2^{t+1}$ queries as part of a secondary test, thereby eliminating the $2^t$ random matches. Thus far, this attack requires $2^s + 2^{t+1}$ queries, $2^s$ memory, and $2^{128-s+t}$ work to recover $2^t$ record encryption keys. (To avoid adaptive queries, we could instead query (A,P) for $P \neq \emptyset$ under the $2^s$ original nonces at the outset, along with the original $2^s$ queries. This second set of queries would be stored in another table, indexed by the nonce, so that once a match on a tag is encountered in the first table, the corresponding nonce can be used to look up the related query in the second table. In this variant $2^{s+1}$ queries (and memory) are needed, and $2^s$ nonces are repeated a total of two times in the attack.)

Once we recover a record encryption key L for some N, it is straightforward to recover H. We use the query under N above where A is arbitrary and $P \neq \emptyset$. Recall that the most significant bit of the input $\mathrm{POLYVAL}(H,A,P,\ell)$ to $E_L$ is set to 0 when computing the tag. Using the known value of L, we decrypt the tag T associated with N and (A,P). This gives us the low 127 bits of the output of a $GF(2^{128})$ polynomial in H with known coefficients. If we correctly guess the high bit of the output of this polynomial, the record authentication key is among its roots. In fact, if we choose $A = \emptyset$ and $P = 0^{128}$ in the above query, then we obtain a degree 1 equation in H and two candidates for H (one for each guess of the high bit of the output). Given L and two candidates for H, we can recover H uniquely using a single forgery attempt under the nonce N. Note that once H is recovered, we know that $H' = L$ is the record authentication key for nonce $N' = H$. Thus, we obtain an additional record authentication key. Even though we don't know the record encryption key for nonce N', we can still forge messages under nonce N' (given matched plaintext/ciphertext under this nonce). Altogether, this attack uses $2^s + 2^{t+1}$ queries, $2^s$ memory, $2^t$ forgery attempts, and $2^{128-s+t}$ work to recover the record encryption and authentication keys for $2^t$ nonces and the record authentication keys for an additional $2^t$ nonces.

**First Attack (256-bit Keys)**

The previous attack is similar for 256-bit keys. We describe the basic attack first and then improve it by exploiting the property of the 256-bit key generation method alluded to earlier. The basic attack requires $2 \times 2^s$ queries/memory, $2^t$ forgery attempts, and $2^{256-s+t}$ work to recover the record encryption and authentication keys for $2^t$ nonces and the record authentication keys for an additional $2^{t+1}$ nonces. The number of queries needed is due to the fact that $2^{128+t}$ random matches are expected. To account for this, at the outset we query two (A,P) pairs for each nonce, where the first pair is (A,P) = (ø,ø) and the second pair uses a P with at least two 128-bit blocks. Then we use the counter mode equations associated with the second pair as part of a secondary test to eliminate the $2^{128+t}$ random matches. Note also that the number of additional record authentication keys recovered by the attack has increased from $2^t$ to $2^{t+1}$. This is due to the fact that once (L,H) is recovered for some nonce N, we immediately obtain the record authentication key for two additional nonces. In particular, since $H = E_K(N)$ and $L = E_K(H)||E_K(E_K(H))$, we obtain $H' = E_K(H)$ as the record authentication key for nonce $N' = H$ and $H'' = E_K(E_K(H))$ as the record authentication key for nonce $N'' = E_K(H)$. In fact, we now use this observation to improve the attack.

Once a single (L,H) pair is recovered for some nonce N, there is an alternative and less expensive method to recover additional (L,H) pairs. Thus, we can set t = 0 in the above attack, recover one (L,H) pair, and use the alternative method to recover more (L,H) pairs. In particular, suppose we have recovered (L,H) for some nonce N. Then, as noted above, we obtain the record authentication key $H' = E_K(H)$ for nonce $N' = H$. Furthermore, we know that the record encryption key L′ for nonce N′ is given by $L' = (E_K(E_K(H)),X)$, where $E_K(E_K(H))$ is known (it is the right 128-bit half of L) and where X is an unknown 128-bit value. Therefore, we query an arbitrary (A,P) pair with nonce N′, observe the tag T and ciphertext C, exhaust over the $2^{128}$ possibilities for X, and obtain check from T and C. To summarize, given (L,H) for N, we immediately obtain H′ for nonce N′ = H, and we can recover the 256-bit L′ for nonce N′ with one query and 128 bits of work.

Clearly, this process can be repeated. Given (L′,H′) for N′, we can then recover (L′′,H′′) for N′′ = H′ with one query under nonce N′′ and 128 bits of work. Note that if we view $E_K$ as a random permutation on 128 bits, the expected length of the cycle that the nonce N is on is about $2^{127}$. Thus, we expect to be able to recover $2^k$ additional (L,H) pairs with $2^{128+k}$ work (for $k \leq 127$). Hence, the full attack requires $2^{s+1} + 2^k$ queries, $\max(2^{s+1},2^k)$ memory, $2^{256-s} + 2^{128+k}$ work, and one forgery attempt to recover $1 + 2^k$ (L,H) pairs. Note that the single forgery attempt is needed to recover the first H; no additional forgery attempts are needed to recover subsequent H values.

As an example, consider the parameters s = 64 and k = 64. In this case, using about $2^{65.6}$ queries and $2^{65}$ memory, we can recover about $2^{64}$ (L,H) pairs with $2^{193}$ work and one forgery attempt.

**Returning the Nonce**

Previous versions of AES-GCM-SIV XORed the nonce to the POLYVAL output in the tag generation process. Returning the nonce strengthens the scheme against this attack, although it does not completely thwart the attack. We can mount a variant of the attack by choosing two different nonces N and N′ that differ only in their most significant bit, say N = 0||X and N′=1||X for some 127-bit value X. Then the two tags associated with (A,P) = (ø,ø) satisfy $T = E_L(N)$ and $T' = E_{L'}(N)$. As such, in the offline phase we can set (s,t)=(1,0) and compute $T_j = E_{L_j}(N)$ for $2^{128-s+t} = 2^{127}$ distinct values of $L_j$. Consequently, we obtain an attack that uses (at most) $2^s + 2^{t+1} = 4$ queries, albeit it is marginally subexhaustive, requiring $2^{127}$ work to recover the record encryption key and record authentication key for one of the nonces, and the record authentication key for an additional nonce (in the case of 128-bit master keys).

**Second Attack (128-bit Keys)**

In this section we describe the attack for the case of 128-bit keys K. As with the first attack, this attack consists of a data collection phase and an offline phase. Fix arbitrarily a 1-block plaintext message $P_0$ and parameters $m \geq n > 0$, and ask for the encryptions of $P_0$ under distinct nonces until some 127-bit $T_{95}||T_{32}$ value is observed $2^m$ times. (We assume there is no additional authenticated data A.) For example, if m = 1 we expect a repeat in $T_{95}||T_{32}$ after observing the encryptions of $P_0$ under about $2^{64}$ nonces $N_i$.

In [2], extensions of the birthday problem are discussed which estimate Q[x,r], the average number of selections with replacement from a set of x elements until some element has been selected r times (an r-fold repeat). It is proved that

$$Q[x,r] \approx (r!)^{1/r} \mathrm{Gamma}(1+1/r)\, x^{(1-1/r)}$$

as x tends to infinity. In our case $x = 2^{127}$. If r = 4 (i.e., m = 2), we find that $Q[2^{127},4] \approx 2^{96.3}$ queries would be needed to observe some $T_{95}||T_{32}$ value four times. In general, $Q[2^{127},2^m]$ queries are needed to observe some $T_{95}||T_{32}$ value $2^m$ times. In a $Q[2^{127},2^m]$-long table we store the tags, along with the corresponding values of N and $C_0$. Thus, the memory is equal to the number of queries, namely $Q[2^{127},2^m]$. (Algorithms exist which can reduce the memory, but would require the attacker to repeat nonces.) The table is then sorted on $T_{95}||T_{32}$ in order to find the $2^m$-fold repeat. Subsequently, we consider only these $2^m$ entries of the table, but sorted on the associated values of $C_0$.

In the offline phase we compute $(C_0)_j = P_0 + E_{L_j}(1||T_{95}||T_{32})$ for $2^{128-n}$ distinct values of $L_j$, where $T_{95}||T_{32}$ is the value of the $2^m$-long repeat found above. We expect $2^{m+(128-n)-128} = 2^{m-n}$ $(N_i,L_j)$ pairs for which $L_i = E_K(E_K(N_i)) = L_j$, and we can detect these events by observing a match on the corresponding ciphertexts $C_0$. We also expect $2^{m-n}$ random matches. (One way to eliminate random matches is to use 2-block messages $(P_0,P_1)$ in the attack, gaining additional check from the $(P_1,C_1)$ pairs.) Once we recover a record encryption key L for some N, it is straightforward to recover H, as described earlier. Altogether, this attack requires $Q[2^{127},2^m]$ queries/memory, $2^{128-n}$ work, and $2^{m-n}$ forgery attempts to recover the record encryption and authentication keys for $2^{m-n}$ nonces and the record authentication keys for an additional $2^{m-n}$ nonces.

**Second Attack (256-bit Keys)**

The previous attack is similar for 256-bit keys, so we omit most of the details. Furthermore, if we set m = n and only recover one (L,H) pair, we can then use the alternative method (for the first attack with 256-bit keys) to recover additional (L,H) pairs. As such, the full attack requires $Q[2^{127},2^m] + 2^k$ queries, $\max(Q[2^{127},2^m],2^k)$ memory, $2^{256-m} + 2^{128+k}$ work, and one forgery attempt to recover $1 + 2^k$ (L,H) pairs.

**Third Attack**

Recall that when the master key K is 256 bits long, $H = E_K(N)$ and $L = E_K(H)||E_K(E_K(H))$. Suppose N is a fixed point of $E_K$, i.e., $E_K(N) = N$. Then H = N and L = N||N. If we view $E_K$ as a random permutation on 128 bits, then the probability of at least one fixed point for $E_K$ is approximately $1 - 1/e \approx 0.63$, and the expected number of fixed points is one. This leads to the following attack. Let A be arbitrary and let P denote a one-block plaintext. We query (A,P) with all $2^{128}$ nonces N. For each query we guess that H = N and L = N||N and check if (A,P) generates the observed tag T and if P encrypts to the observed ciphertext C. If N is a fixed point of $E_K$, then H = N and L = N||N; thus, we obtain a match on T and C with probability 1. If N is not a fixed point of $E_K$, then $H \neq N$ and $L \neq N||N$. Since the probability of a random match on T and C is $2^{-256}$, we do not expect any matches for nonces that are not fixed points of $E_K$. Thus, we recover at least one 256-bit record encryption key (of the form L = N||N), along with the corresponding record authentication key H = N. The attack requires $2^{128}$ queries and $2^{128}$ work, and succeeds with probability about 0.63.

## References

[1] S. Gueron, A. Langley, and Y. Lindell, "AES-GCM-SIV: Nonce Misuse-Resistant Authenticated Encryption", posted at https://tools.ietf.org/html/draft-irtf-cfrg-gcmsiv-02, 29 August 2016.

[2] M. S. Klamkin and D. J. Newman, “Extensions of the Birthday Surprise”, Journal of Combinatorial Theory, Volume 3 Issue 3, pp. 279-282, October 1967.