Encrypted Nonce Modes on Farfalle

Seth Hoffert

Abstract. Nonces are a fact of life for achieving semantic security. Generating a uniformly random nonce can be costly and may not always be feasible. Using anything other than uniformly random bits can result in information leakage; e.g., a timestamp can deanonymize a communication and a counter can leak the quantity of transmitted messages. Ideally, we would like to be able to efficiently encrypt the nonce to 1) avoid needing uniformly random bits and 2) avoid information leakage. This paper presents two new authenticated encryption modes built on top of Farfalle [2] that perfectly achieve these goals.

Keywords: farfalle, deck functions, authenticated encryption, wide block cipher, modes of use, encrypted nonce

1 Introduction

A typical implementation of an AEAD mode tends to use some combination of a timestamp, a counter and uniform randomness when deriving a nonce. Using uniform randomness for nonce generation can be expensive on platforms that have a limited entropy pool, and using anything other than uniform randomness will inevitably leak information. For example, an observed timestamp can be enough for an adversary to deanonymize a communication if the sender's clock is known to be inexact. Using a counter can also leak information, because it allows an adversary to observe only two messages at different points in time and yet still be able to estimate the number of messages that were sent in between observations.

It would be nice if the nonce could somehow be contained within the plaintext. At first, this seems like a chicken-and-egg problem: how does one encrypt a nonce without using another nonce? The answer lies in the Feistel construction. Remarkably, it is possible to construct an efficient inverse-free mode that is very similar to Deck-PLAIN [3] and yet encrypts the nonce and redundancy with negligible additional overhead. Additionally, we can create a RUP-resistant variant as well, suitable for use in onion encryption protocols.

1.1 Contributions

Our main contribution is constructing two efficient authenticated encryption modes on top of Farfalle [2] that feature nonce and redundancy encryption. The first mode can be viewed as the encrypted nonce analog of Deck-PLAIN [3], and the second mode can be viewed as a constrained version of Deck-JAMBOREE [3] that requires a nonce but still supports RUP, similar to GCM-RUP [1]. Because

both modes are built entirely on top of Farfalle [2], no block ciphers are involved and the inverse direction of the underlying permutation remains unused. This provides a hardware space advantage in ASIC and FPGA designs.

1.2 Conventions

The length in bits of the string X is denoted $|X|$. The concatenation of two strings X, Y is denoted as $X\|Y$ and their bitwise addition as $X \oplus Y$. Bit string values are noted with a typewriter font, such as `01101`. The repetition of a bit is noted in exponent, e.g., $\mathtt{0}^3 = \mathtt{000}$. Finally, $\varnothing$ is the empty set and $\perp$ denotes an error code.

2 Nonce- and redundancy-encrypting AEAD mode

The algorithm listing for the nonce- and redundancy-encrypting AEAD mode is now presented in its most general form. In practice, it is expected that an implementation will fix P_L to a short constant length (e.g., a single block). This allows early rejection to work, and it allows a bulk of the ciphertext bits to be produced immediately after compressing the single-block nonce. The *valid* function called by the decryption oracle is expected to validate the redundancy that is present in P_R so that the plaintext can be safely released.

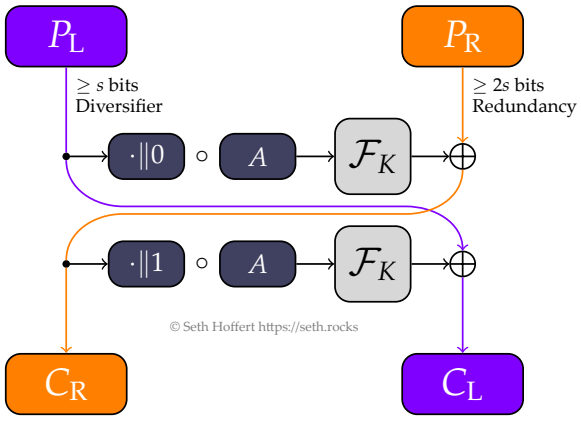

Fig. 1: Nonce- and redundancy-encrypting AEAD mode

2.1 Features

– **Encrypted nonce and redundancy**: both the nonce and redundancy are encrypted, allowing non-random nonces to be used

Algorithm 1.1: Nonce- and redundancy-encrypting AEAD mode

Definition : Farfalle instance $\mathcal{F}$

1 **function** encrypt(key K, metadata A, plaintext (P_L,P_R)): ciphertext
2 **assert** $|P_L| \geq 128$ and P_L contains nonce
3 **assert** $|P_R| \geq 256$ and P_R contains verifiable redundancy
4 $C_R \leftarrow P_R \oplus \mathcal{F}_K(P_L \| \mathtt{0} \circ A)$
5 $C_L \leftarrow P_L \oplus \mathcal{F}_K(C_R \| \mathtt{1} \circ A)$
6 **return** (C_L, C_R)

7 **function** decrypt(key K, metadata A, ciphertext (C_L,C_R)): plaintext or $\perp$
8 **if** $|C_L| < 128$ **or** $|C_R| < 256$ **then return** $\perp$
9 $P_L \leftarrow C_L \oplus \mathcal{F}_K(C_R \| \mathtt{1} \circ A)$
10 $P_R \leftarrow C_R \oplus \mathcal{F}_K(P_L \| \mathtt{0} \circ A)$
11 **if not** valid(P_R) **then return** $\perp$
12 **return** (P_L, P_R)

- **Performance**: only a single amortized read- and write-pass is performed over the plaintext
- **Online encryption**: the encryption oracle produces the majority of the ciphertext bits immediately after compressing the nonce
- **Early rejection**: by placing the verifiable redundancy at the beginning of P_R, early rejection can be realized
- **Timing-insensitive authentication**: because the redundancy is non-malleable, a timing-leaky equality function can be used in function *valid*
- **Optional/static metadata**: the metadata string compression can be skipped if empty; additionally, static metadata can be factored out and reused across invocations

2.2 Security proof

We defer the security proof to a future revision.

2.3 Variations

The redundancy is non-malleable, which allows for the use of a timing-leaky equality function while validating redundancy. If this timing-insensitive property is not needed, then it is possible to forego compression of the redundancy on line 5 of algorithm 1.1. Notably, this brings the mode even closer in parity to DeckPLAIN [3]. The modified algorithm is deferred to a future paper.

3 Nonce- and redundancy-encrypting AEAD mode with RUP resistance

To achieve RUP resistance, we simply need an additional round in order to achieve non-malleability over the entirety of the decrypted plaintext. It enjoys the

same properties as its simpler counterpart, requiring only one additional single-block compression and expansion. Similar to the simpler mode, it is expected that implementations will keep $|P_L|$ to one block maximum. This configuration maximizes the performance in that it allows for an amortized single read- and write-pass over the plaintext.

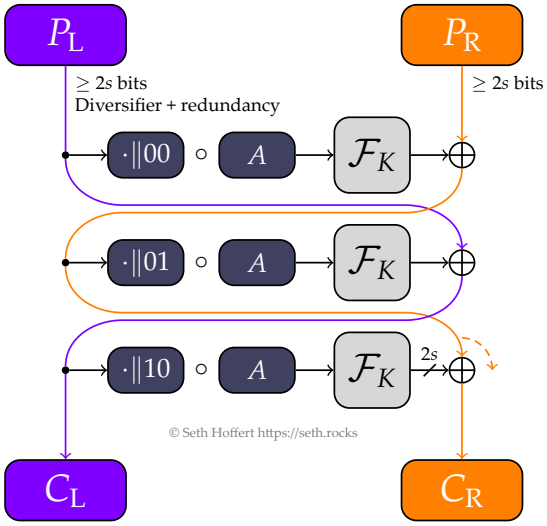

Fig. 2: Nonce- and redundancy-encrypting AEAD mode with RUP resistance

3.1 Features

- **Encrypted nonce and redundancy**: both the nonce and redundancy are encrypted, allowing non-random nonces to be used
- **Performance**: only a single amortized read- and write-pass is performed over the plaintext
- **Online encryption**: the encryption oracle produces the majority of the ciphertext bits immediately after compressing the nonce
- **Early rejection**: because the verifiable redundancy is in P_L, early rejection can be realized
- **Timing-insensitive authentication**: because the redundancy is non-malleable, a timing-leaky equality function can be used in function *valid*
- **Optional/static metadata**: the metadata string compression can be skipped if empty; additionally, static metadata can be factored out and reused across invocations
- **RUP resistance**: this mode can be used in onion protocols thanks to its RUP resistance

Algorithm 1.2: Nonce- and redundancy-encrypting AEAD mode with RUP resistance

Definition : Farfalle instance $\mathcal{F}$

function encrypt(key K, metadata A, plaintext (P_L,P_R)): ciphertext
  assert $|P_L| \geq 256$ and P_L contains nonce and verifiable redundancy
  assert $|P_R| \geq 256$
  $C_R \leftarrow P_R \oplus \mathcal{F}_K(P_L \| \texttt{00} \circ A)$
  $C_L \leftarrow P_L \oplus \mathcal{F}_K(C_R \| \texttt{01} \circ A)$
  $C_{R0} \leftarrow C_{R0} \oplus \mathcal{F}_K(C_L \| \texttt{10} \circ A)$
  return (C_L, C_R)

function decrypt(key K, metadata A, ciphertext (C_L,C_R)): plaintext or $\perp$
  if $|C_L| < 256$ **or** $|C_R| < 256$ **then return** $\perp$
  $C_{R0} \leftarrow C_{R0} \oplus \mathcal{F}_K(C_L \| \texttt{10} \circ A)$
  $P_L \leftarrow C_L \oplus \mathcal{F}_K(C_R \| \texttt{01} \circ A)$
  if not valid(P_L) **then return** $\perp$
  $P_R \leftarrow C_R \oplus \mathcal{F}_K(P_L \| \texttt{00} \circ A)$
  return (P_L,P_R)

3.2 Security proof

We defer the security proof to a future revision.

4 Session support

In an effort to keep the algorithm listings minimal, neither of the presented modes has native session support. One could implement session-based versions of both of the presented algorithms simply by keeping an incremental history. This feature is deferred to a future paper.

5 Conclusions

The presented modes are a testament to Farfalle's flexibility and power. Algorithm 1.1 can be viewed as an enhanced Deck-PLAIN [3], and algorithm 1.2 is ideal for situations when RUP is needed (e.g., onion routing). Both modes achieve the encrypted nonce goal and are efficient, requiring only a single amortized read- and write-pass over the plaintext.

References

1. Ashur, T., Dunkelman, O., Luykx, A.: Boosting authenticated encryption robustness with minimal modifications. Cryptology ePrint Archive, Paper 2017/239 (2017), https://eprint.iacr.org/2017/239, https://eprint.iacr.org/2017/239

2. Bertoni, G., Daemen, J., Hoffert, S., Peeters, M., Van Assche, G., Van Keer, R.: Farfalle: parallel permutation-based cryptography. IACR ToSC (2017)
3. Băcuieți, N., Daemen, J., Hoffert, S., Assche, G.V., Keer, R.V.: Jammin' on the deck. Cryptology ePrint Archive, Paper 2022/531 (2022), https://eprint.iacr.org/2022/531, https://eprint.iacr.org/2022/531