Nonce- and Redundancy-encrypting Modes with 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 new modes built on top of Farfalle [3] that achieve nonce and redundancy encryption in the AEAD and onion AE settings.

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

1 Introduction

Typical usage of an AEAD mode tends to involve 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 accurately estimate the number of messages that were sent in between observations.

It would be preferable to encrypt the nonce; i.e., to treat the nonce as part of the plaintext. At first, this seems like a chicken-and-egg problem: how does one encrypt a nonce without using another nonce? One such solution lies in the Feistel construction. Remarkably, it is possible to construct an efficient inverse-free mode that is very similar to Deck-PLAIN [4] and yet encrypts the nonce and redundancy with negligible additional overhead. We showcase RUP-resistant and onion AE modes as well.

1.1 Contributions

Our primary contribution is constructing efficient authenticated encryption modes on top of Farfalle [3] that feature nonce and redundancy encryption. We provide a variant that is secure against RUP, similar to GCM-RUP [1]. Additionally, we address the application of onion AE. Specifically, we propose two efficient stateful modes that can be viewed as generalizations of our aforementioned stateless AEAD modes.

Because our modes are built entirely on top of Farfalle [3], no block ciphers are involved and the inverse direction of the underlying permutation remains unused. Besides the obvious benefit of elegance from needing only one type of primitive, this provides a hardware space advantage in ASIC and FPGA designs. We also benefit from existing cryptanalysis on the Farfalle instantiation.

Another important advantage of building modes on top of Farfalle is that it allows the designer to focus on mode design instead of low-level concerns such as the handling of multiple input/output blocks, parallelizability, and the number of rounds to take in the underlying permutation. Farfalle provides us with a random oracle primitive that can be used to build a wide variety of modes without the typical pitfalls of building new modes from scratch.

1.2 Related work

In an effort to leverage existing nonce-based encryption schemes, Bellare, Ng and Tackmann [2] describe a set of parameterized transformations. Each transformation has different properties but all achieve the goal of protecting the nonce. Note that while our modes are not generic transformations, we nonetheless also avoid building from scratch, preferring instead to leverage the power of Farfalle.

Providing security even when unverified plaintext is released is addressed by Ashur, Dunkelman and Luykx [1]. Applicability to onion AE is also discussed. Note that our RUP-resistant mode differs in that it does not make use of a block cipher and places no maximum length restriction on the nonce. Indeed, our modes treat the nonce, redundancy and plaintext on equal footing.

1.3 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 denoted with a typewriter font, such as `01101`. The repetition of a bit is denoted in exponent, e.g., $\texttt{0}^3 = \texttt{000}$. Substrings with exclusive upper bounds are denoted $[x..y)$. $\varnothing$ is the empty set and $\perp$ denotes an error code. Finally, pad is any injective padding function.

2 Two-round Feistel cipher

We first present a two-round Feistel cipher depicted in figure 1, parameterized by Farfalle instance $\mathcal{F}$. We then present two concrete modes built using the cipher. The first mode is a stateless nonce- and redundancy-encrypting mode, and the second is a stateful onion AE mode.

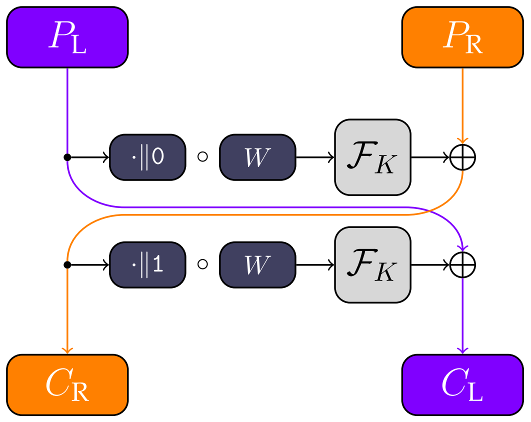

Fig. 1: Two-round Feistel cipher

2.1 Details

The Feistel cipher in figure 1 can be used as a building block in secure modes of operation. We will show that one way to build such a mode is by providing a nonce in (K, W, P_L) and validating redundancy in P_R, and then we will show how to generalize this to the onion AE setting.

2.2 Mode: Nonce- and redundancy-encrypting AEAD

Consider a virtual private network (VPN) protocol. In such a protocol, many small packets are exchanged at a high frequency. It would be very costly to generate a uniformly random nonce per packet for the following reasons:

– **Space**: to achieve 128 bits of security, we would need a 256-bit nonce to mitigate birthday bound collisions
– **Time**: because of the high-frequency nature of the application, the OS entropy pool could become exhausted and block the application until more can be gathered

Deck-PLAIN allows for a single nonce to be specified per session. However, because packets can be lost and arrive out-of-order, we cannot use a session-based mode. A solution to this problem is to build a stateless mode that treats the nonce as part of the plaintext, enabling the use of non-random nonces without leaking information. We now present such a mode in algorithm 1.1.

This mode is effectively a phase-shifted (and stateless) version of Deck-PLAIN [4]. In the encryption oracle, note that $C \sim \mathrm{PRF}(A, P_L)$. By requiring (K, A, P_L) to be a nonce, we ensure that C is indistinguishable from random. Likewise, in the decryption oracle, note that $P_R \sim \mathrm{PRF}(A, C)$. By requiring verifiable redundancy in P_R, we ensure that any tampering of (A, C) is detected. Additionally, note that P_L is allowed to contain arbitrary plaintext, as long as

(K, A, P_L) is a nonce. Once P_R reaches the minimum length requirement, an implementation may choose to fill P_L to an entire block to achieve optimal performance.

Remarkably, the early rejection feature of Deck-PLAIN is retained. By keeping P_L short and placing the entirety of the redundancy within the first block of P_R, the decryption oracle can expand just enough bits to decrypt the redundancy and validate it. If valid, then it can expand the remaining bits to decrypt the rest of C_R. Asymptotically, authentication can be performed after a single read pass, just like in Deck-PLAIN. Because of this feature, the risk of leaking unverified plaintext is eliminated. Another advantage of keeping P_L short is to allow the encryption oracle to be online: the asymptotic majority of the ciphertext bits are produced immediately after compressing P_L.

Because $P_\mathrm{R} \sim \mathrm{PRF}(A, C)$, if any of the encrypted redundancy bits of C are tampered with, then every bit of decrypted redundancy is flipped with 50% probability. This allows for the use of a non-constant-time equality function. Due to the fact that the redundancy is non-malleable, nothing is gleaned from the timing of the comparison during redundancy validation. Note that if this feature is not needed, i.e., a constant-time equality function is used, then the compression of the redundancy portion of C_R can be skipped on lines 5 and 8 of algorithm 1.1. This optimization brings the mode even closer to parity with Deck-PLAIN. For brevity, the modified algorithm is deferred to a future paper.

Another benefit of encrypted and non-malleable redundancy is the ability to use part of the plaintext as redundancy. For example, a timestamp could be validated against a fixed time window relative to the receiver's clock. Given a 64-bit timestamp with 1-second resolution and a window size of 256 seconds, $64 - \log_2 256 = 56$ bits of authenticity is already achieved. Because of the requirements of figure 1, this mode does not allow overlapping the nonce and redundancy. For treatment of this feature, refer to section 3.2.

Metadata string A is optional and can be safely omitted from compression thanks to the frame bits on the plaintext. Note, however, that metadata-only input is not supported. The algorithm can be modified to support the metadata-only case, but this circumvents the nonce- and redundancy-encrypting goals of the mode. If a message authentication code is desired, then we recommend supplying the nonce and redundancy as part of the plaintext for consistency.

Similar to [2], this mode achieves nonce encryption, but has important differences as well. The transformations described in [2] are designed to work with existing AEAD schemes, providing the benefit of reusing existing work. Nonetheless, this comes at the cost of requiring additional key material and a separate PRF invocation to encrypt the nonce. Algorithm 1.1 on the other hand is designed with nonce encryption in mind from the beginning, with the benefit of using only a single key and requiring only two invocations of the random oracle primitive, putting its performance at parity with that of Deck-PLAIN.

Features

Algorithm 1.1: Nonce- and redundancy-encrypting AEAD mode

```
   Definition : Farfalle instance $\mathcal{F}$
   Definition : Security level $s = 128$ bits
   Precondition : Nonce $(K, A, P_\mathrm{L})$
   Precondition : $P_\mathrm{R}$ contains $s$ bits of valid redundancy
1  function encrypt(key $K$, metadata $A$, plaintext $(P_\mathrm{L}, P_\mathrm{R})$) : ciphertext
2      $P'_\mathrm{L} \leftarrow \mathrm{pad}(P_\mathrm{L})$ such that $|P'_\mathrm{L}| \geq s$
3      $P'_\mathrm{R} \leftarrow \mathrm{pad}(P_\mathrm{R})$ such that $|P'_\mathrm{R}| \geq 2s$
4      $C_\mathrm{R} \leftarrow P'_\mathrm{R} \oplus \mathcal{F}_K(P'_\mathrm{L} \| \mathtt{0} \circ A)$
5      $C_\mathrm{L} \leftarrow P'_\mathrm{L} \oplus \mathcal{F}_K(C_\mathrm{R} \| \mathtt{1} \circ A)$
6      return $(C_\mathrm{L}, C_\mathrm{R})$
   Precondition : $|C_\mathrm{L}| \geq s$ and $|C_\mathrm{R}| \geq 2s$
7  function decrypt(key $K$, metadata $A$, ciphertext $(C_\mathrm{L}, C_\mathrm{R})$) : plaintext or $\perp$
8      $P_\mathrm{L} \leftarrow C_\mathrm{L} \oplus \mathcal{F}_K(C_\mathrm{R} \| \mathtt{1} \circ A)$
9      $P_\mathrm{R} \leftarrow C_\mathrm{R} \oplus \mathcal{F}_K(P_\mathrm{L} \| \mathtt{0} \circ A)$
10     if invalid redundancy in $P_\mathrm{R}$ then return $\perp$
11     return $(\mathrm{pad}^{-1}(P_\mathrm{L}), \mathrm{pad}^{-1}(P_\mathrm{R}))$
```

- **Encrypted nonce and redundancy**: both the nonce and redundancy are encrypted, allowing embedded nonces and redundancy to be used without risk of information leak
- **Performance**: asymptotically performs only a single read- and write-pass over the plaintext
- **Online encryption**: for short P_L, the encryption oracle produces the majority of the ciphertext bits immediately after compressing P_L
- **Early rejection**: by keeping P_L short and 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 non-constant-time equality function can be used to validate redundancy
- **Optional/static metadata**: the metadata string compression can be skipped if empty; additionally, static metadata can be factored out and reused across invocations

Security proof We defer the security proof to a future revision.

2.3 Mode: Onion AE without leaky pipe

Consider the application of onion AE. In this application, we must recursively encrypt a message such that each node in the circuit decrypts (i.e., strips off) its respective layer and relays the result to the next node in the circuit. The terminal node, being either the client or exit node, decrypts its layer and validates the authenticity. If valid, then it processes the message; otherwise, it rejects it. We wish to satisfy the following properties:

- **Authenticated encryption**: the client and exit node must be able to cryptographically authenticate the message
- **Length preservation**: because the cryptographic algorithm is applied recursively, the payload length must be preserved to avoid leaking semantic information about number of layers
- **Statefulness**: if a message is corrupted, then authentication must fail for all subsequent messages reaching the client and exit node

Deck-PLAIN is ruled out immediately since it incurs a per-encryption expansion due to explicit redundancy. Algorithm 1.1 does not fit the bill either because it requires a nonce and redundancy in every encryption. An important observation of figure 1 is that there is more than one way to satisfy the PRF constraints. We now present algorithms 1.2 and 1.3 satisfying the constraints in the onion AE setting.

In order to satisfy the PRF constraints, the client encryption oracle accepts only plaintexts that contain valid redundancy and its decryption oracle releases only plaintexts that contain valid redundancy. Because the mode is stateful, tampering will effectively poison the state of every node in the circuit such that all subsequent decryptions in the client and exit node will fail authentication. This concept is visualized in figure 2.

The ability for the client to send/receive messages to/from any node in the circuit is referred to as the leaky pipe architecture. Because figure 1 is not resistant to RUP, the client can send messages only to the exit node; all other nodes are strictly relays. We build a mode that supports the leaky pipe architecture in section 3.3.

Note that a session-based analog of algorithm 1.1 can be obtained by setting the number of clients to 1, and compressing metadata into the history if desired. If the key is not ephemeral, then simply ensure that a nonce is present in the metadata or first wrap's P_{L}.

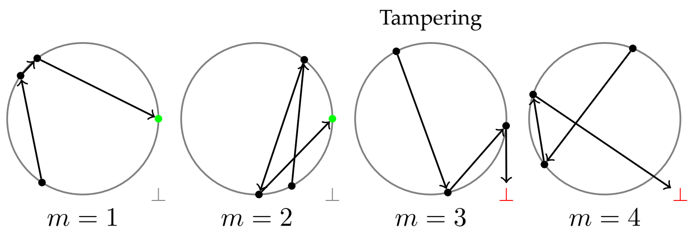

Fig. 2: Visualization of stateful onion AE with tampering. A gray circle represents the space of all possible plaintexts and ciphertexts, with $\perp$ representing the error value. The process of recursively encrypting/decrypting is effectively a random walk through the space, here visualized by vertices and edges. A green vertex indicates an authentic plaintext. Here, four messages are shown and the circuit is composed of the client plus three nodes.

Algorithm 1.2: Onion session AE mode for client

```
Definition : Farfalle instance $\mathcal{F}$
Definition : Security level $s = 128$ bits
constructor init(ephemeral node keys $K_*$)
    $\mathcal{G}_* \leftarrow \mathcal{F}_{K_*}$
Precondition : $P_\mathrm{R}$ contains $s$ bits of valid redundancy
function wrap(plaintext $(P_\mathrm{L}, P_\mathrm{R})$) : ciphertext
    $P'_\mathrm{L} \leftarrow \mathrm{pad}(P_\mathrm{L})$ such that $|P'_\mathrm{L}| \geq s$
    for $j = |\mathcal{G}| - 1$ through $0$ do
        $C_\mathrm{R} \leftarrow P_\mathrm{R} \oplus \mathcal{G}_j(P'_\mathrm{L} \| \mathtt{00} \circ \mathrm{history}^{\downarrow}_j)$
        $C_\mathrm{L} \leftarrow P'_\mathrm{L} \oplus \mathcal{G}_j(C_\mathrm{R} \| \mathtt{01} \circ \mathrm{history}^{\downarrow}_j)$
        $\mathrm{history}^{\downarrow}_j \leftarrow P'_\mathrm{L} \| \mathtt{00} \circ \mathrm{history}^{\downarrow}_j$
        $(P'_\mathrm{L}, P_\mathrm{R}) \leftarrow (C_\mathrm{L}, C_\mathrm{R})$
    return $(C_\mathrm{L}, C_\mathrm{R})$
Precondition : $|C_\mathrm{L}| \geq s$ and $|C_\mathrm{R}| \geq s$
function unwrap(ciphertext $(C_\mathrm{L}, C_\mathrm{R})$) : plaintext or $\bot$
    for $j = 0$ through $|\mathcal{G}| - 1$ do
        $P_\mathrm{L} \leftarrow C_\mathrm{L} \oplus \mathcal{G}_j(C_\mathrm{R} \| \mathtt{11} \circ \mathrm{history}^{\uparrow}_j)$
        $P_\mathrm{R} \leftarrow C_\mathrm{R} \oplus \mathcal{G}_j(P_\mathrm{L} \| \mathtt{10} \circ \mathrm{history}^{\uparrow}_j)$
        $\mathrm{history}^{\uparrow}_j \leftarrow P_\mathrm{L} \| \mathtt{10} \circ \mathrm{history}^{\uparrow}_j$
        $(C_\mathrm{L}, C_\mathrm{R}) \leftarrow (P_\mathrm{L}, P_\mathrm{R})$
    if invalid redundancy in $P_\mathrm{R}$ then return $\bot$
    return $(\mathrm{pad}^{-1}(P_\mathrm{L}), P_\mathrm{R})$
```

Features

- **Encrypted redundancy**: the redundancy is encrypted, allowing embedded redundancy to be used without risk of information leak
- **Performance**: asymptotically performs only a single read- and write-pass over the plaintext, per node
- **Early rejection**: by keeping P_L short and placing the verifiable redundancy at the beginning of P_R, early rejection in the client and exit node can be realized
- **Timing-insensitive authentication**: because the redundancy is non-malleable, a non-constant-time equality function can be used to validate redundancy

Security proof We defer the security proof to a future revision.

3 Three-round Feistel cipher

We first present a three-round Feistel cipher depicted in figure 3, parameterized by Farfalle instance $\mathcal{F}$. We then present two concrete modes built using the

Algorithm 1.3: Onion session AE mode for node

```
Definition : Farfalle instance $\mathcal{F}$
Definition : Security level $s = 128$ bits
1  constructor init(ephemeral key $K$)
2      $\mathcal{G} \leftarrow \mathcal{F}_K$
   Precondition : $\begin{cases} P_\mathrm{R} \text{ contains } s \text{ bits of valid redundancy}, & \text{if exit-node;} \\ |P_\mathrm{L}| \geq s \text{ and } |P_\mathrm{R}| \geq s, & \text{otherwise.} \end{cases}$
3  function wrap(plaintext $(P_\mathrm{L}, P_\mathrm{R})$) : ciphertext
4      if exit-node then
5          $P'_\mathrm{L} \leftarrow \mathrm{pad}(P_\mathrm{L})$ such that $|P'_\mathrm{L}| \geq s$
6      else
7          $P'_\mathrm{L} \leftarrow P_\mathrm{L}$
8      $C_\mathrm{R} \leftarrow P_\mathrm{R} \oplus \mathcal{G}(P'_\mathrm{L} \| \mathtt{10} \circ \mathrm{history}^\uparrow)$
9      $C_\mathrm{L} \leftarrow P'_\mathrm{L} \oplus \mathcal{G}(C_\mathrm{R} \| \mathtt{11} \circ \mathrm{history}^\uparrow)$
10     $\mathrm{history}^\uparrow \leftarrow P'_\mathrm{L} \| \mathtt{10} \circ \mathrm{history}^\uparrow$
11     return $(C_\mathrm{L}, C_\mathrm{R})$
   Precondition : $|C_\mathrm{L}| \geq s$ and $|C_\mathrm{R}| \geq s$
12 function unwrap(ciphertext $(C_\mathrm{L}, C_\mathrm{R})$) : plaintext or $\perp$
13     $P_\mathrm{L} \leftarrow C_\mathrm{L} \oplus \mathcal{G}(C_\mathrm{R} \| \mathtt{01} \circ \mathrm{history}^\downarrow)$
14     $P_\mathrm{R} \leftarrow C_\mathrm{R} \oplus \mathcal{G}(P_\mathrm{L} \| \mathtt{00} \circ \mathrm{history}^\downarrow)$
15     $\mathrm{history}^\downarrow \leftarrow P_\mathrm{L} \| \mathtt{00} \circ \mathrm{history}^\downarrow$
16     if exit-node then
17         if invalid redundancy in $P_\mathrm{R}$ then return $\perp$
18         return $(\mathrm{pad}^{-1}(P_\mathrm{L}), P_\mathrm{R})$
19     return $(P_\mathrm{L}, P_\mathrm{R})$
```

cipher. The first mode is a stateless nonce- and redundancy-encrypting mode with RUP resistance, and the second is a stateful onion AE mode with leaky pipe architecture.

3.1 Details

The Feistel cipher in figure 3 can be used as a building block in secure modes of operation that provide security under RUP. In figure 1, note of the inverse that P_L is malleable. To achieve RUP resistance, we need only communicate a digest of C_L into C_R to ensure that $P \sim \mathrm{PRF}(A, C)$. In spite of this additional round, note that a single read- and write-pass over the plaintext is still achieved by keeping P_L short (i.e., one block maximum).

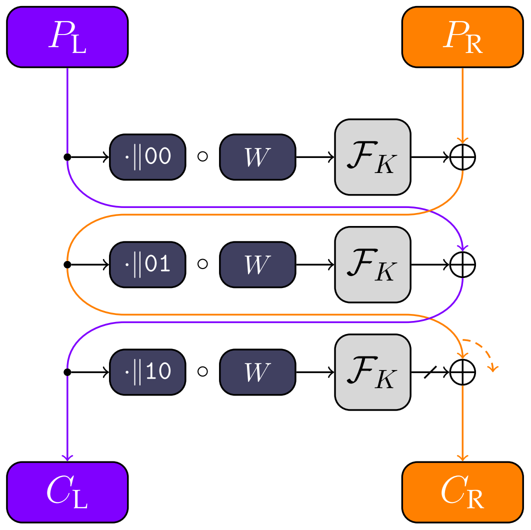

Fig. 3: Three-round Feistel cipher

3.2 Mode: Nonce- and redundancy-encrypting AEAD with RUP resistance

We now present the RUP-resistant analog of algorithm 1.1 in algorithm 1.4. This mode adds RUP resistance and the ability to overlap the nonce and redundancy. The benefit of having both the nonce and redundancy reside in non-malleable P_L is that they are treated on equal footing. For example, if a timestamp comprises P_L, then an implementation can rely on it as a nonce while also validating the timestamp against a fixed time window at decryption time. With a 64-bit timestamp and an allowed time window of 256 seconds, $64 - \log_2 256 = 56$ bits of authenticity is already achieved. This saves space in the plaintext by reducing the need for a separate nonce and redundancy.

Features

- **Encrypted nonce and redundancy**: both the nonce and redundancy are encrypted, allowing embedded nonces and redundancy to be used without risk of information leak
- **Performance**: for short P_L, asymptotically performs only a single read- and write-pass over the plaintext
- **Online encryption**: for short P_L, the encryption oracle produces the majority of the ciphertext bits immediately after compressing P_L
- **Early rejection**: by placing the verifiable redundancy at the beginning of P_L, early rejection can be realized
- **Timing-insensitive authentication**: because the redundancy is non-malleable, a non-constant-time equality function can be used to validate redundancy

Algorithm 1.4: Nonce- and redundancy-encrypting AEAD mode with RUP resistance

```
Definition : Farfalle instance $\mathcal{F}$
Definition : Security level $s = 128$ bits
Precondition : Nonce $(K, A, P_\mathrm{L})$
Precondition : $P_\mathrm{L}$ contains $s$ bits of valid redundancy
function encrypt(key $K$, metadata $A$, plaintext $(P_\mathrm{L}, P_\mathrm{R})$) : ciphertext
    $P'_\mathrm{L} \leftarrow \mathrm{pad}(P_\mathrm{L})$ such that $|P'_\mathrm{L}| \geq 2s$
    $P'_\mathrm{R} \leftarrow \mathrm{pad}(P_\mathrm{R})$ such that $|P'_\mathrm{R}| \geq 2s$
    $C_\mathrm{R} \leftarrow P'_\mathrm{R} \oplus \mathcal{F}_K(P'_\mathrm{L} \| \texttt{00} \circ A)$
    $C_\mathrm{L} \leftarrow P'_\mathrm{L} \oplus \mathcal{F}_K(C_\mathrm{R} \| \texttt{01} \circ A)$
    $C_\mathrm{R}[..2s) \leftarrow C_\mathrm{R}[..2s) \oplus \mathcal{F}_K(C_\mathrm{L} \| \texttt{10} \circ A)$
    return $(C_\mathrm{L}, C_\mathrm{R})$

Precondition : $|C_\mathrm{L}| \geq 2s$ and $|C_\mathrm{R}| \geq 2s$
function decrypt(key $K$, metadata $A$, ciphertext $(C_\mathrm{L}, C_\mathrm{R})$) : plaintext or $\perp$
    $C_\mathrm{R}[..2s) \leftarrow C_\mathrm{R}[..2s) \oplus \mathcal{F}_K(C_\mathrm{L} \| \texttt{10} \circ A)$
    $P_\mathrm{L} \leftarrow C_\mathrm{L} \oplus \mathcal{F}_K(C_\mathrm{R} \| \texttt{01} \circ A)$
    if invalid redundancy in $P_\mathrm{L}$ then return $\perp$
    $P_\mathrm{R} \leftarrow C_\mathrm{R} \oplus \mathcal{F}_K(P_\mathrm{L} \| \texttt{00} \circ A)$
    return $(\mathrm{pad}^{-1}(P_\mathrm{L}), \mathrm{pad}^{-1}(P_\mathrm{R}))$

Precondition : $|C_\mathrm{L}| \geq 2s$ and $|C_\mathrm{R}| \geq 2s$
function leak(key $K$, metadata $A$, ciphertext $(C_\mathrm{L}, C_\mathrm{R})$) : plaintext or $\top$
    $C_\mathrm{R}[..2s) \leftarrow C_\mathrm{R}[..2s) \oplus \mathcal{F}_K(C_\mathrm{L} \| \texttt{10} \circ A)$
    $P_\mathrm{L} \leftarrow C_\mathrm{L} \oplus \mathcal{F}_K(C_\mathrm{R} \| \texttt{01} \circ A)$
    if valid redundancy in $P_\mathrm{L}$ then return $\top$
    $P_\mathrm{R} \leftarrow C_\mathrm{R} \oplus \mathcal{F}_K(P_\mathrm{L} \| \texttt{00} \circ A)$
    return $(P_\mathrm{L}, P_\mathrm{R})$
```

- **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**: the mode does not break down if unverified plaintext is released from the decryption oracle
- **Overlapped nonce and redundancy**: the nonce and redundancy can be overlapped, resulting in a space savings

Security proof We defer the security proof to a future revision.

3.3 Mode: Onion AE with leaky pipe

In section 2.3, we describe a mode that solves a special case of the onion AE problem where the client can send/receive messages only to/from the exit node. Now we wish to solve for the more general case; that is, we wish to support the leaky pipe architecture, visualized in figure 4.

In the leaky pipe architecture, the client can send/receive messages to/from any node, not just the exit node. Because intermediate nodes in this model conditionally pass their decrypted output to the next node, such a mode must satisfy the additional property of RUP resistance. We now present algorithms 1.5 and 1.6 satisfying the aforementioned goals.

Note that a session-based analog of algorithm 1.4 can be obtained by setting the number of clients to 1, and compressing metadata into the history if desired. If the key is not ephemeral, then simply ensure that a nonce is present in the metadata or first wrap's P_L.

Algorithm 1.5: Onion session AE mode for client, with leaky pipe

```
Definition : Farfalle instance $\mathcal{F}$
Definition : Security level $s = 128$ bits
constructor init(ephemeral node keys $K_*$)
    $\mathcal{G}_* \leftarrow \mathcal{F}_{K_*}$
Precondition : $0 \le i < |\mathcal{G}|$
Precondition : $P_\mathrm{L}$ contains $s$ bits of valid redundancy
function wrap(target $i$, plaintext $(P_\mathrm{L}, P_\mathrm{R})$) : ciphertext
    $P'_\mathrm{R} \leftarrow \mathrm{pad}(P_\mathrm{R})$ such that $|P'_\mathrm{R}| \ge s$
    for $j = i$ through $0$ do
        $C_\mathrm{R} \leftarrow P'_\mathrm{R} \oplus \mathcal{G}_j(P_\mathrm{L} \| \mathtt{000} \circ \mathrm{history}_j^{\downarrow})$
        $C_\mathrm{L} \leftarrow P_\mathrm{L} \oplus \mathcal{G}_j(C_\mathrm{R} \| \mathtt{001} \circ \mathrm{history}_j^{\downarrow})$
        $C_\mathrm{R}[..s) \leftarrow C_\mathrm{R}[..s) \oplus \mathcal{G}_j(C_\mathrm{L} \| \mathtt{010} \circ \mathrm{history}_j^{\downarrow})$
        $\mathrm{history}_j^{\downarrow} \leftarrow C_\mathrm{R} \| \mathtt{001} \circ \mathrm{history}_j^{\downarrow}$
        $(P_\mathrm{L}, P'_\mathrm{R}) \leftarrow (C_\mathrm{L}, C_\mathrm{R})$
    return $(C_\mathrm{L}, C_\mathrm{R})$
Precondition : $|C_\mathrm{L}| \ge s$ and $|C_\mathrm{R}| \ge s$
function unwrap(ciphertext $(C_\mathrm{L}, C_\mathrm{R})$) : (source, plaintext) or $\perp$
    for $j = 0$ through $|\mathcal{G}| - 1$ do
        $C_\mathrm{R}[..s) \leftarrow C_\mathrm{R}[..s) \oplus \mathcal{G}_j(C_\mathrm{L} \| \mathtt{110} \circ \mathrm{history}_j^{\uparrow})$
        $P_\mathrm{L} \leftarrow C_\mathrm{L} \oplus \mathcal{G}_j(C_\mathrm{R} \| \mathtt{101} \circ \mathrm{history}_j^{\uparrow})$
        $P_\mathrm{R} \leftarrow C_\mathrm{R} \oplus \mathcal{G}_j(P_\mathrm{L} \| \mathtt{100} \circ \mathrm{history}_j^{\uparrow})$
        $\mathrm{history}_j^{\uparrow} \leftarrow C_\mathrm{R} \| \mathtt{101} \circ \mathrm{history}_j^{\uparrow}$
        if valid redundancy in $P_\mathrm{L}$ then
            return $(j, P_\mathrm{L}, \mathrm{pad}^{-1}(P_\mathrm{R}))$
        $(C_\mathrm{L}, C_\mathrm{R}) \leftarrow (P_\mathrm{L}, P_\mathrm{R})$
    return $\perp$
```

Features

Algorithm 1.6: Onion session AE mode for node, with leaky pipe

```
Definition : Farfalle instance $\mathcal{F}$
Definition : Security level $s = 128$ bits
constructor init(ephemeral key $K$)
    $\mathcal{G} \leftarrow \mathcal{F}_K$
Precondition : from-me = true, if exit-node
Precondition : $\begin{cases} P_\mathrm{L} \text{ contains } s \text{ bits of valid redundancy}, & \text{if from-me;} \\ |P_\mathrm{L}| \geq s \text{ and } |P_\mathrm{R}| \geq s, & \text{otherwise.} \end{cases}$
function wrap(from-me, plaintext $(P_\mathrm{L}, P_\mathrm{R})$) : ciphertext
    if from-me then
        $P'_\mathrm{R} \leftarrow \mathrm{pad}(P_\mathrm{R})$ such that $|P'_\mathrm{R}| \geq s$
    else
        $P'_\mathrm{R} \leftarrow P_\mathrm{R}$
    $C_\mathrm{R} \leftarrow P'_\mathrm{R} \oplus \mathcal{G}(P_\mathrm{L} \| \mathtt{100} \circ \mathrm{history}^\uparrow)$
    $C_\mathrm{L} \leftarrow P_\mathrm{L} \oplus \mathcal{G}(C_\mathrm{R} \| \mathtt{101} \circ \mathrm{history}^\uparrow)$
    $C_\mathrm{R}[..s) \leftarrow C_\mathrm{R}[..s) \oplus \mathcal{G}(C_\mathrm{L} \| \mathtt{110} \circ \mathrm{history}^\uparrow)$
    $\mathrm{history}^\uparrow \leftarrow C_\mathrm{R} \| \mathtt{101} \circ \mathrm{history}^\uparrow$
    return $(C_\mathrm{L}, C_\mathrm{R})$
Precondition : $|C_\mathrm{L}| \geq s$ and $|C_\mathrm{R}| \geq s$
function unwrap(ciphertext $(C_\mathrm{L}, C_\mathrm{R})$) : (to-me, plaintext) or $\perp$
    $C_\mathrm{R}[..s) \leftarrow C_\mathrm{R}[..s) \oplus \mathcal{G}(C_\mathrm{L} \| \mathtt{010} \circ \mathrm{history}^\downarrow)$
    $P_\mathrm{L} \leftarrow C_\mathrm{L} \oplus \mathcal{G}(C_\mathrm{R} \| \mathtt{001} \circ \mathrm{history}^\downarrow)$
    $P_\mathrm{R} \leftarrow C_\mathrm{R} \oplus \mathcal{G}(P_\mathrm{L} \| \mathtt{000} \circ \mathrm{history}^\downarrow)$
    $\mathrm{history}^\downarrow \leftarrow C_\mathrm{R} \| \mathtt{001} \circ \mathrm{history}^\downarrow$
    if valid redundancy in $P_\mathrm{L}$ then
        return (true, $P_\mathrm{L}, \mathrm{pad}^{-1}(P_\mathrm{R})$)
    if exit-node then return $\perp$
    return (false, $P_\mathrm{L}, P_\mathrm{R}$)
```

- **Encrypted redundancy**: the redundancy is encrypted, allowing embedded redundancy to be used without risk of information leak
- **Performance**: for short P_L, asymptotically performs only a single read- and write-pass over the plaintext, per node
- **Early rejection**: by placing the verifiable redundancy at the beginning of P_L, early rejection in the client and exit node can be realized
- **Timing-insensitive authentication**: because the redundancy is non-malleable, a non-constant-time equality function can be used to validate redundancy
- **Leaky pipe architecture**: any node can securely send and receive messages

Security proof We defer the security proof to a future revision.

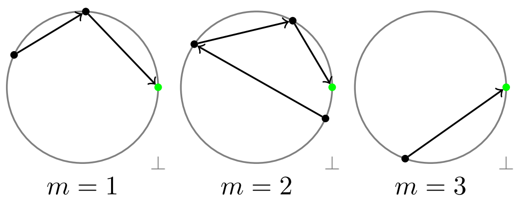

Fig. 4: Visualization of stateful onion AE with leaky pipe architecture. A gray circle represents the space of all possible plaintexts and ciphertexts, with $\perp$ representing the error value. The process of recursively encrypting/decrypting is effectively a random walk through the space, here visualized by vertices and edges. A green vertex indicates an authentic plaintext. Here, three messages are shown and the circuit is composed of the client plus three nodes. The client sends a message destined for the second node, then the third node, then the first node.

4 Conclusions

The presented modes are a testament to Farfalle's flexibility and power. We described two Feistel ciphers, along with concrete modes for AEAD and onion AE. The presented modes not only solve the encrypted nonce and onion AE goals but are efficient as well, asymptotically requiring only a single 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. Bellare, M., Ng, R., Tackmann, B.: Nonces are noticed: Aead revisited. Cryptology ePrint Archive, Paper 2019/624 (2019), https://eprint.iacr.org/2019/624, https://eprint.iacr.org/2019/624
3. Bertoni, G., Daemen, J., Hoffert, S., Peeters, M., Van Assche, G., Van Keer, R.: Farfalle: parallel permutation-based cryptography. IACR Trans. Symmetric Cryptol. 2017(4), 1–38 (2017)
4. 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