Elephant v1

Submitters:
- Tim Beyne (KU Leuven and imec-COSIC, Belgium)
- Yu Long Chen (KU Leuven and imec-COSIC, Belgium)
- Christoph Dobraunig (Radboud University, The Netherlands)
- Bart Mennink (Radboud University, The Netherlands)

Corresponding submitter:
- Bart Mennink
  elephant@cs.ru.nl
  +31 24 3652672
  Radboud University, The Netherlands
  Toernooiveld 212
  6525 EC Nijmegen
  The Netherlands

February 25, 2019
1 Introduction

We introduce the Elephant authenticated encryption scheme. The mode of Elephant is a nonce-based encrypt-then-MAC construction, where encryption is performed using counter mode and message authentication using a variant of the Wegman-Carter-Shoup [10,82,92] MAC function. Both modes internally use a cryptographic permutation masked using LFSRs, akin to the masked Even-Mansour construction of Granger et al. [49].

The mode is permutation-based and only evaluates this permutation in the forward direction. As such, there is no need to implement multiple primitives or the inverse of the primitive, unlike in OCB-based [58, 78, 79] authenticated encryption schemes. Furthermore, this allows us to rely and build on the extensive literature of permutations used for sponge-based lightweight hashing [6,21,51]. That said, Elephant itself is not sponge-based: on the contrary, it departs from the conventional approach of serial permutation-based authenticated encryption. Elephant is parallelizable by design, easy to implement due to the use of LFSRs for masking (no need for finite field multiplication), and finally, it is efficient due to elegant decisions on how the masking should be performed exactly. A security analysis in the ideal permutation model demonstrates that the mode of Elephant is structurally sound.

Due to the parallelizability of Elephant, there is no need to instantiate Elephant with a large permutation: we can go as small as 160-bit permutations while still matching the security goals recommended by the NIST lightweight call [72]. In detail, the Elephant scheme consists of three instances:

1. Dumbo: Elephant-Spongent-$\pi[160]$. This instance meets the minimum permutation size as dictated by the security analysis: it achieves 112-bit security provided that the online complexity is at most around $2^{46}$ blocks. This instance is particularly well-suited for hardware, as Spongent [21] itself is;

2. Jumbo: Elephant-Spongent-$\pi[176]$. This is a slightly more conservative instance of Elephant: it is based on the same permutation family, yet achieves 127-bit security under the same conditions on the online complexity. We note, in particular, that Spongent-$\pi[176]$ is ISO/IEC standardized [21,54];

3. Delirium: Elephant-Keccak-$f[200]$. This variant is developed more towards software use, although it still performs reasonably well in hardware. Elephant instantiated with Keccak-$f[200]$ also achieves 127-bit security, with a higher bound of around $2^{70}$ blocks on the online complexity. The permutation is the smallest instance of the NIST SHA-3 standard [14,47] that fits our need.

Dumbo is the primary member of the submission. Dumbo and Jumbo are named after two famous elephants; Delirium is named after a Belgian beer, whose logo is a pink elephant. As each of the permutations is relatively small, all versions of Elephant have a small state size, despite its support for parallelism. The
LFSRs used for masking are tailored to the specific instance, one for each, and are developed to operate well with the specific cryptographic permutation. For example, the LFSRs paired with the Spongent instances have been chosen to minimize the number of XOR operations that have to be performed for a state-update, while the Keccak-based instance has been selected to perform well on software platforms.

We note that the three cryptographic permutations in Elephant can also be used for cryptographic hashing – in fact, Spongent [21] and Keccak [14] themselves are sponges – but due to our quest for small permutations, these cryptographic hash functions cannot meet the 112-, or 127-bit security level guaranteed by our authenticated encryption schemes. In contrast, in order to perform sponge-based hashing with at least 112-bit security, a cryptographic permutation of size at least 225 bits must be used.

2 Algorithmic Specification

The generic Elephant mode is presented in Section 2.2, and the three primitives used within the mode are presented in Sections 2.3-2.5. Before going to the mode, we briefly describe the notation used in 2.1.

2.1 Notation

For \( n \in \mathbb{N} \), we let \( \{0,1\}^n \) denote the set of \( n \)-bit strings and \( \{0,1\}^* \) the set of arbitrarily length strings. For \( X \in \{0,1\}^* \), we define

\[
X_1 \ldots X_\ell \leftarrow^n X
\]

(1)

to be the function that partitions \( X \) into \( \ell = \lceil |X|/n \rceil \) blocks of size \( n \) bits, where the last block is appended with 0s. The expression “\( A \implies B : C \)” equals \( B \) if \( A \) is true, and equals \( C \) if \( A \) is false. For \( x \in \{0,1\}^n \) and \( i \leq n \), we denote by \( x \ll i \) (resp., \( x \gg i \)) a shift of \( x \) to the left (resp., right) over \( i \) positions. We likewise denote by \( x \ll i \) (resp., \( x \gg i \)) a rotation of \( x \) to the left (resp., right) over \( i \) positions. We denote by \( [x]_i \) the \( i \) left-most bits of \( x \).

2.2 Elephant Authenticated Encryption Mode

Let \( k,m,n,t \in \mathbb{N} \) with \( k,m,t \leq n \). Let \( P : \{0,1\}^n \rightarrow \{0,1\}^n \) be an \( n \)-bit permutation, and \( \varphi_1 : \{0,1\}^n \rightarrow \{0,1\}^n \) be an LFSR. Define \( \varphi_2 = \varphi_1 \oplus \text{id} \). Define the function \( \text{mask} : \{0,1\}^k \times \mathbb{N}^2 \rightarrow \{0,1\}^n \) as follows:

\[
\text{mask}_{K}^{a,b} = \text{mask}(K,a,b) = \varphi_2^b \circ \varphi_1^a \circ P(K||0^{n-k})
\]

(2)

We will describe the generic authenticated encryption mode of Elephant. It consists of two algorithms: encryption \text{enc} and decryption \text{dec}.
Algorithm 1 Elephant encryption algorithm enc

| Input: | \((K, N, A, M) \in \{0, 1\}^k \times \{0, 1\}^m \times \{0, 1\}^* \times \{0, 1\}^*\) |
| Output: | \((C, T) \in \{0, 1\}^{\left| M \right|} \times \{0, 1\}^t\) |

1. \(M_1 \ldots M_{\ell_M} \leftarrow M\)
2. for \(i = 1, \ldots, \ell_M\) do
3. \(C_i \leftarrow M_i \oplus P(N \parallel 0^{n-m} \oplus \text{mask}_K^{i-1,0}) \oplus \text{mask}_K^{i-1,0}\)
4. \(C \leftarrow \lfloor C_1 \ldots C_{\ell_M} \rfloor_{\left| M \right|}\)
5. \(T = 0\)
6. \(A_1 \ldots A_{\ell_A} \leftarrow N \parallel A \parallel 1\)
7. \(C_1 \ldots C_{\ell_C} \leftarrow C \parallel 1\)
8. for \(i = 1, \ldots, \ell_A\) do
9. \(T \leftarrow T \oplus P(A_i \oplus \text{mask}_K^{i-1,2}) \oplus \text{mask}_K^{i-1,2}\)
10. for \(i = 1, \ldots, \ell_C\) do
11. \(T \leftarrow T \oplus P(C_i \oplus \text{mask}_K^{i-1,1}) \oplus \text{mask}_K^{i-1,1}\)
12. return \((C, [T]_t)\)

2.2.1 Encryption

Encryption enc gets as input a key \(K \in \{0, 1\}^k\), a nonce \(N \in \{0, 1\}^m\), associated data \(A \in \{0, 1\}^*\), and a message \(M \in \{0, 1\}^*\), and it outputs a ciphertext \(C \in \{0, 1\}^*\) and a tag \(T \in \{0, 1\}^t\). The description of enc is given in Algorithm 1, and it is depicted in Figure 1.

2.2.2 Decryption

Decryption dec gets as input a key \(K \in \{0, 1\}^k\), a nonce \(N \in \{0, 1\}^m\), associated data \(A \in \{0, 1\}^*\), a ciphertext \(C \in \{0, 1\}^*\), and a tag \(T \in \{0, 1\}^t\), and it outputs a message \(M \in \{0, 1\}^*\) if the tag is correct, or a dedicated \(\perp\)-sign otherwise. The description of dec is given in Algorithm 2.

2.3 160-Bit Permutation and LFSR

Section 2.3.1 defines the Spongent-\(\pi[160]\) permutation. The 160-bit masking LFSR \(\varphi_1\) is defined in Section 2.3.2. These components are used in Dumbo.

2.3.1 Spongent Permutation

We denote by Spongent-\(\pi[160]\): \(\{0, 1\}^{160} \rightarrow \{0, 1\}^{160}\) the 80-round Spongent permutation of Bogdanov et al. [21]. It operates on a 160-bit input \(X\) as follows:

\[
\text{for } i = 1, \ldots, 80 \text{ do}
\]
\[
X \leftarrow X \oplus 0^{153} || \text{Counter}_{160}(i) \oplus \text{rev}(0^{153} || \text{Counter}_{160}(i))
\]
\[
X \leftarrow \text{sBoxLayer}_{160}(X)
\]
\[
X \leftarrow \text{pLayer}_{160}(X)
\]
where the function \( \text{rev} \) reverses the order of the bits of its input, and where the functions \( \text{lCounter}_{160} \), \( \text{sBoxLayer}_{160} \), and \( \text{pLayer}_{160} \) are defined as follows:

- \( \text{lCounter}_{160} \): this function is a 7-bit LFSR defined by the primitive polynomial \( p(x) = x^7 + x^6 + 1 \) and initialized with “1000101”;
- \( \text{sBoxLayer}_{160} \): this function consists of an S-box \( S : \{0,1\}^4 \to \{0,1\}^4 \) applied 40 times in parallel. In hexadecimal notation, this S-box is defined as

| X | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| \( S(X) \) | E | D | B | 0 | 2 | 1 | 4 | F | 7 | A | 8 | 5 | 9 | C | 3 | 6 |

- \( \text{pLayer}_{160} \): this function moves the \( j \)-th bit of its input to bit position \( P_{160}(j) \), where

\[
P_{160}(j) = \begin{cases} 
40 \cdot j \mod 159, & \text{if } j \in \{0, \ldots, 158\}, \\
159, & \text{if } j = 159.
\end{cases}
\]

2.3.2 LFSR

For generating the masks of our scheme, we use the approach of Granger et al. [49]. We define \( \varphi_1 \) as the following \( \mathbb{F}_2 \)-linear map, where the \( x_i \)’s correspond
Algorithm 2 Elephant decryption algorithm dec

Input: \((K, N, A, C, T) \in \{0, 1\}^k \times \{0, 1\}^m \times \{0, 1\}^* \times \{0, 1\}^* \times \{0, 1\}^\ell\)

Output: \(M \in \{0, 1\}^{C|\ell|} \) or \(\bot\)

1: \(C_1 \ldots C_{\ell_M} \leftarrow C\)
2: for \(i = 1, \ldots, \ell_M\) do
3: \(M_i \leftarrow C_i \oplus P(N \parallel 0^{n-m} \oplus \text{mask}_{i-1,0}^K) \oplus \text{mask}_{K}^{i-1,0}\)
4: \(M \leftarrow \lfloor M_1 \ldots M_{\ell_M} \rfloor_{|C|}\)
5: \(\bar{T} = 0\)
6: \(A_1 \ldots A_{\ell_A} \leftarrow N \parallel A\parallel 1\)
7: \(C_1 \ldots C_{\ell_C} \leftarrow C\parallel 1\)
8: for \(i = 1, \ldots, \ell_A\) do
9: \(\bar{T} \leftarrow \bar{T} \oplus P(A_i \oplus \text{mask}_{i-1,2}^K) \oplus \text{mask}_{K}^{i-1,2}\)
10: for \(i = 1, \ldots, \ell_C\) do
11: \(\bar{T} \leftarrow \bar{T} \oplus P(C_i \oplus \text{mask}_{K}^{i-1,1}) \oplus \text{mask}_{K}^{i-1,1}\)
12: return \(\lfloor \bar{T} \rfloor_{\ell} = T \oplus M : \bot\)

To 8-bit words:

\[(x_0, \ldots, x_{19}) \mapsto (x_1, \ldots, x_{19}, x_0 \ll 3 \oplus x_3 \ll 7 \oplus x_{13} \gg 7).\] (3)

2.4 176-Bit Permutation and LFSR

Section 2.4.1 defines the Spongent-\(\pi\)[176] permutation. The 176-bit masking LFSR \(\varphi_1\) is defined in Section 2.4.2. These components are used in Jumbo.

2.4.1 Spongent Permutation

We denote by \(\text{Spongent-\(\pi\)[176]} : \{0, 1\}^{176} \to \{0, 1\}^{176}\) the 90-round Spongent permutation of Bogdanov et al. [21]. It operates on a 176-bit input \(X\) as follows:

for \(i = 1, \ldots, 90\) do
\(X \leftarrow X \oplus 0^{169} \parallel \text{Counter}_{176}(i) \oplus \text{rev}(0^{169} \parallel \text{Counter}_{176}(i))\)
\(X \leftarrow \text{sBoxLayer}_{176}(X)\)
\(X \leftarrow \text{pLayer}_{176}(X)\)

where, as before, the function \(\text{rev}\) reverses the order of the bits of its input. The function \(\text{Counter}_{176}\) is the same as \(\text{Counter}_{160}\) of Section 2.3 but initialized with “1111010”, the function \(\text{sBoxLayer}_{176}\) consists of the function \(S\) of Section 2.3 applied 44 times in parallel, and \(\text{pLayer}_{176}\) is now defined as the function that moves the \(j\)-th bit of its input to bit position \(P_{176}(j)\), where

\[P_{176}(j) = \begin{cases} 44 \cdot j \mod 175, & \text{if } j \in \{0, \ldots, 174\}, \\ 175, & \text{if } j = 175. \end{cases}\]
2.4.2 LFSR

For generating the masks of our scheme, we use the approach of Granger et al. [49]. The LFSR $\varphi_1$ is defined as the following $\mathbb{F}_2$-linear map, where the $x_i$’s correspond to 8-bit words:

$$(x_0, \ldots, x_{21}) \mapsto (x_1, \ldots, x_{21}, x_0 \ll 1 \oplus x_3 \ll 7 \oplus x_{19} \gg 7).$$

(4)

2.5 200-Bit Permutation and LFSR

Section 2.5.1 defines the Keccak-$f[200]$ permutation. The 200-bit masking LFSR $\varphi_1$ is defined in Section 2.5.2. These components are used in Delirium.

2.5.1 Keccak Permutation

We denote by Keccak-$f[200] : \{0, 1\}^{200} \rightarrow \{0, 1\}^{200}$ the 18-round Keccak permutation of Bertoni et al. [14, 47]. The state $X \in \{0, 1\}^{200}$ is represented as a 5-by-5-by-8 array $a \in \{0, 1\}^{5 \times 5 \times 8}$, where for $(x, y, z) \in \mathbb{Z}_5 \times \mathbb{Z}_5 \times \mathbb{Z}_8$ the bit at position $(x, y, z)$ is set as

$$a[x, y, z] = X[8(5y + x) + z].$$

Keccak-$f[200]$ operates on a 200-bit input $X$ as follows:

for $i = 1, \ldots, 18$ do

$X \leftarrow \iota \circ \chi \circ \pi \circ \rho \circ \theta(X)$

where the functions $\theta$, $\rho$, $\pi$, $\chi$, and $\iota$ are defined as follows:

$$\theta : a[x, y, z] \leftarrow a[x, y, z] \oplus \bigoplus_{y'=0}^{4} a[x - 1, y', z] \oplus \bigoplus_{y'=0}^{4} a[x + 1, y', z - 1],$$

$$\rho : a[x, y, z] \leftarrow a[x, y, z + t[x, y]],$$

$$\pi : a[x, y, z] \leftarrow a[x + 3y, x, z],$$

$$\chi : a[x, y, z] \leftarrow a[x, y, z] \oplus (a[x + 1, y, z] \oplus 1)a[x + 2, y, z],$$

$$\iota : a[x, y, z] \leftarrow a[x, y, z] \oplus RC[i, x, y, z].$$

For $\rho$, the function $t[x, y]$ is defined as

<table>
<thead>
<tr>
<th>$t$</th>
<th>$x = 3$</th>
<th>$x = 4$</th>
<th>$x = 0$</th>
<th>$x = 1$</th>
<th>$x = 2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y = 2$</td>
<td>153</td>
<td>231</td>
<td>3</td>
<td>10</td>
<td>171</td>
</tr>
<tr>
<td>$y = 1$</td>
<td>55</td>
<td>276</td>
<td>36</td>
<td>300</td>
<td>6</td>
</tr>
<tr>
<td>$y = 0$</td>
<td>28</td>
<td>91</td>
<td>0</td>
<td>1</td>
<td>190</td>
</tr>
<tr>
<td>$y = 4$</td>
<td>120</td>
<td>78</td>
<td>210</td>
<td>66</td>
<td>253</td>
</tr>
<tr>
<td>$y = 3$</td>
<td>21</td>
<td>136</td>
<td>105</td>
<td>45</td>
<td>15</td>
</tr>
</tbody>
</table>

and for $\iota$, the round constants are given by

$$RC[i, x, y, z] = \begin{cases} 
    rc[j + 7i], & \text{if } (x, y, z) = (0, 0, 2^j - 1), \\
    0, & \text{otherwise}
\end{cases}.$$
where $rc$ is computed from a binary LFSR defined by the primitive polynomial $p(x) = x^8 + x^6 + x^5 + x^4 + 1$.

### 2.5.2 LFSR

For generating the masks of our scheme, we use the approach of Granger et al. [49]. The LFSR $\varphi_1$ is now defined as the following $\mathbb{F}_2$-linear map, where the $x_i$'s correspond to 8-bit words:

$$ (x_0, \ldots, x_{24}) \mapsto (x_1, \ldots, x_{24}, x_0 \ll 1 \oplus x_2 \ll 1 \oplus x_{13} \ll 1). \quad (5) $$

### 3 Parameterization of Elephant

Elephant consists of three instances, namely those built from instantiating the mode using the permutation and LFSR of Sections 2.3, 2.4, and 2.5, respectively. In more detail, we restrict our focus to $n \in \{160, 176, 200\}$. We also set $m = 96$, i.e. we restrict to nonces of size 96 bits. Parameters $k, t \in \mathbb{N}$ are still tunable. We propose the following three instances of Elephant (with Dumbo being the primary member):

<table>
<thead>
<tr>
<th>instance</th>
<th>$k$</th>
<th>$m$</th>
<th>$n$</th>
<th>$t$</th>
<th>$P$</th>
<th>$\varphi_1$</th>
<th>expected limit on security complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dumbo</td>
<td>128</td>
<td>96</td>
<td>160</td>
<td>64</td>
<td>Spongent-$\pi$[160] (3)</td>
<td>$2^{112}$</td>
<td>$2^{50} / (n/8)$</td>
</tr>
<tr>
<td>Jumbo</td>
<td>128</td>
<td>96</td>
<td>176</td>
<td>64</td>
<td>Spongent-$\pi$[176] (4)</td>
<td>$2^{127}$</td>
<td>$2^{50} / (n/8)$</td>
</tr>
<tr>
<td>Delirium</td>
<td>128</td>
<td>96</td>
<td>200</td>
<td>128</td>
<td>Keccak-$f$[200] (5)</td>
<td>$2^{127}$</td>
<td>$2^{74} / (n/8)$</td>
</tr>
</tbody>
</table>

Here, the online complexity is in terms of the number of $n$-bit blocks (hence all instances support an online complexity of $2^{50}$ bytes), and the strength is measured in the offline complexity, i.e., the number of primitive evaluations that the adversary can make.

In Appendix B, we give a formal security analysis of the Elephant authenticated encryption mode in the ideal permutation model, and prove that the advantage of a nonce-based adversary in breaking security of either of the schemes is at most

$$ \text{Adv}_{\text{Elephant}}^\text{ae}(\mathcal{A}) \leq \ell \left( \frac{q_e}{2} \right)^2/2^n + \frac{2^n - q_d}{2^n - 1} e^{(q_e + 1)q_e/2^n} + \frac{4\sigma^2 + 4\sigma p + 4\sigma + p}{2^n} + \frac{p}{2^k}, $$

where $q_e$ expresses an upper bound on the number of evaluations of the encryption function, $q_d$ the number of decryption queries, $\ell$ the maximum length of a single query in blocks, $\sigma$ the total online complexity in blocks, and $p$ the number of evaluations of the random primitive $P$. Note that the dominating term in the bound is $4\sigma p/2^n$. By capping $\sigma \leq 2^{n-114}$, this term is less than 1 as long as $p \leq 2^{112}$. Likewise, by capping $\sigma \leq 2^{n-130}$, this term is less than 1 as long as $p \leq 2^{128}$. However, one also needs to take the other terms of the bound...
into account. Most of the terms are negligible compared to $4\sigma p/2^n$, and are covered by taking a slightly stricter condition on $\sigma$ (note that $2^{50}/(n/8) < 2^{16}$ and $2^{74}/(n/8) \leq 2^{70}$ for each of the instances). There is one exception to these negligible terms, namely the factor $p/2^k$ for Jumbo and Delirium: it equals 1 for $p = 2^{128}$. This term thus accounts for a factor 2 loss in the security strength of Jumbo and Delirium, and we must restrict the offline complexity for these variants by a factor 2, as indicated in above table.

We stress that these security claims only hold in the nonce-respecting setting: the adversary may not evaluate the encryption function twice under the same nonce (it may make decryption queries for a reused nonce, though). If the nonce is reused for two different evaluations of $\text{enc}$, security is void. In particular, if the nonce uniqueness condition is released, trivial confidentiality and integrity attacks can be mounted. This is not considered to be a flaw in the scheme. We also do not claim security in case unverified plaintext is released [5]; we note, however, that in practice decryption of the ciphertext $C$ into the message $M$ takes place only after the tag (in turn, computed from the nonce, associated data, and ciphertext) has been verified. Finally, security decreases in the multi-key or related-key setting.

4 Design Rationale

The Elephant mode is an encrypt-then-MAC mode, where encryption is performed by counter mode and message authentication by a variant of Wegman-Carter-Shoup [82, 92], both implicitly instantiated using a simplification of the masked Even-Mansour (MEM) tweakable block cipher of Granger et al. [49]. This tweakable block cipher, in turn, is based on a Spongent [21] or Keccak [14] permutation. We explain the design rationale of Elephant at the following three levels of granularity: the generic mode in Section 4.1, how the mode uses the permutation, i.e., the masking scheme, in Section 4.2, and the choice of particular primitives in Section 4.3. Finally, Section 4.4 briefly discusses implementation aspects.

4.1 Mode

Generically, encrypt-then-MAC is the most secure approach [9, 71]: unlike its alternatives encrypt-and-MAC and MAC-then-encrypt, this approach yields integrity of ciphertexts. Stated differently, malformed ciphertexts yield failure upon MAC verification, and for these no decryption is needed. This prevents unintended leakage from verification failures. The approach also makes it possible to easily prevent leakage due to release of unverified plaintext: simply do not start decrypting before the tag is verified. Note that for the generic alternatives encrypt-and-MAC and MAC-then-encrypt, such a simple countermeasure is impossible. This makes the encrypt-then-MAC mode of Elephant preferable over its alternatives, not only in the lightweight setting but also for general purpose.

The counter encryption mode and Wegman-Carter-Shoup MAC mode within
**Elephant**, in turn, are both fully parallelizable and only evaluate the underlying permutation \( P \) in forward direction. The fact that **Elephant** evaluates its primitive in forward direction is important in the lightweight setting: it allows for smaller implementations, since there is no need to implement the inverse of \( P \). Note, in particular, that due to the rise of the sponge, various cryptographic permutations, including Ascon [42], Gimli [12], Keccak [14], and XOODOO [33], are developed to be particularly efficient in forward direction.

By being parallelizable, **Elephant** distinguishes itself from a wide range of authenticated encryption schemes that employ a serial permutation-based mode of operation, such as APE [3], Beetle [28], or the Duplex construction [13,34,66]. To support parallelism, we need to store the internal state value, but on the upside, it turns out to give various elegant implementation advantages (see Section 4.2 and Section 4.4) and it means that there is no strict need to employ larger permutations.

The mode is nonce-based: each of the members of **Elephant** uses a 96-bit nonce. The nonce is prepended to the associated data, which is then padded into \( n \)-bit blocks \( A_1 \ldots A_\ell \) (see line 6 of Algorithm 1). This way, the scheme is optimized for the parameters specified in the NIST call [72]: the nonce is 96 bits, and in order to avoid a waste of \( n - 96 \) bits due to padding (where \( n \in \{160,176,200\} \)), the nonce is appended with the first \( n - 96 \) bits of the associated data. Caution must be paid here, namely that the **nonce is always of fixed length of 96 bits**. If variable-length nonces were allowed, the scheme would be vulnerable to trivial padding attacks. Also, as the mode is nonce-based, security is guaranteed *only if* the adversary does not repeat nonces for encryption queries.

### 4.2 Masking

As specified in Section 2.2, the inputs to and outputs of the permutation \( P \) are masked using \( \text{mask}^{a,b}_K \) of (2). The masking function is defined using two LFSRs \( \varphi_1, \varphi_2 : \{0,1\}^n \rightarrow \{0,1\}^n \) that satisfy \( \varphi_2 = \varphi_1 \oplus \text{id} \), and it is parameterized by \((a,b)\) which are used in a manner so as to assure that every occurrence of the masking in the **Elephant** mode gets different parameters.

The LFSR-based masking technique is taken from Granger et al. [49], and so is the security analysis (although different state sizes, discrete logarithm computations, LFSRs, and tweak domains are considered). Granger et al. have argued in favor of this technique over its alternatives for various reasons: (i) the approach is simpler to implement, as the masking is purely linear and does not use finite field multiplication, (ii) it is more efficient (depending on the primitive used), and (iii) the masking is constant time.

The latter point is important in the lightweight setting where resistance against timing attacks comes at a cost. In this respect, the LFSR-based masking approach compares favorably with another, and very popular, masking technique, namely powering-up-based masking (simplified to allow for fair compar-
ison with (2)):

\[3^b 2^a P(K \| 0^{n-k})\]

where 2 and 3 are coordinates in the monomial basis in the finite field \( \mathbb{F}_{2^n} \).

The technique was introduced by Rogaway [78] in the context of OCB2, and it has seen many applications, including CAESAR submissions AES-OTR [67], AEZ [52], COLM [4], Minalpher [81], POET [2], and SHELL [91]. Note that, in \( \mathbb{F}_{2^n} \), a multiplication by 2 can be implemented as a conditional XOR, which is variable-time and which can only be made constant time at a certain cost. A similar issue occurs for multiplication by 3. For comparison, the masking functions \( \varphi_1 \) and \( \varphi_2 \) are constant time by design.

A related masking approach is that of OCB3 [58] and OMD [31], which use masking based on Gray coding. In detail, Gray coding-based masks can be updated as

\[G(i) = G(i-1) \oplus 2^{\text{ntz}(i)}\]

where \( \text{ntz}(i) \) is the number of trailing zeros in the binary representation of \( i \). The masking, unlike powering-up, does not need a conditional XOR, but it requires \( \log_2(i) \) field doublings (which may be precomputed). As the LFSR-based masking used in Elephant does not incur such a cost, it also compares favorably with this technique.

The particular choice of masking, namely \((a, b) = (i, 0)\) in the encryption layer, \((a, b) = (i, 1)\) for ciphertext authentication, and \((a, b) = (i, 2)\) for associated data authentication, allows maskings to cancel out nicely in the implementation. To see this, consider the authentication of ciphertext \( C_i \) (for \( i < \ell_M \leq \ell_C \)), and more detailed the contribution \( T_i \) it makes to tag \( T \). This value is computed as

\[T_i = P\left(M_i \oplus P(N \| 0^{n-m} \oplus \text{mask}^{i-1,0}_K \oplus \text{mask}^{i-1,1}_K) \oplus \text{mask}^{i-1,1}_K \right)\]

By definition of \( \text{mask}^{a,b}_K \), and as \( \varphi_2 = \varphi_1 \oplus \text{id} \), we have

\[\text{mask}^{i-1,0}_K \oplus \text{mask}^{i-1,1}_K = \varphi_1^{i-1} \circ P(K \| 0^{n-k}) \oplus (\varphi_1 \oplus \text{id}) \circ \varphi_1^{i-1} \circ P(K \| 0^{n-k}) = \varphi_1 \circ P(K \| 0^{n-k})\]

This, not surprisingly, is the mask used for the encryption of the next message block \( M_{i+1} \).

Another optimization in mask management is in the masks that contribute to the tag, i.e., the sum of all masks that appear in the final tag \( T \). The contribution coming from the ciphertext authentication equals

\[\left( \bigoplus_{i=1}^{\ell_C} \text{mask}^{i-1,1}_K \right) = \left( \bigoplus_{i=1}^{\ell_C} (\varphi_1 \oplus \text{id}) \circ \varphi_1^{i-1} \circ P(K \| 0^{n-k}) \right) = (\varphi_1^{\ell_C} \oplus \text{id}) \circ P(K \| 0^{n-k})\]

and that coming from the associated data likewise equals

\[\left( \bigoplus_{i=1}^{\ell_A} \text{mask}^{i-1,2}_K \right) = (\varphi_1^{\ell_A+1} \oplus \varphi_1^{\ell_A} \oplus \varphi_1 \oplus \text{id}) \circ P(K \| 0^{n-k})\].

11
This feature of the masking may be useful if Elephant is used for fixed-length data, in which case the (6) and (7) could be precomputed.

4.3 Primitives

4.3.1 Dumbo and Jumbo

Both the 160-bit and 176-bit instance of Elephant are based on a Spongent permutation [21]: the 160-bit instance is based on the Spongent-$\pi[160]$ permutation, and the 176-bit instance is based on the Spongent-$\pi[176]$ permutation. The choice for Spongent is natural: it is particularly well-suited for hardware, and the existing third-party analysis (see Section 5.2) does not indicate any weakness of the Spongent family relevant for our use-case. We have used the 160-bit version of Spongent as this is the smallest possible permutation that can be used to efficiently meet the NIST call for proposals. The 176-bit Spongent permutation offers a slightly more comfortable 127-bit security margin. In addition, this particular Spongent permutation is part of the ISO/IEC standard on lightweight hash functions [54].

Bogdanov et al. [21] do not explicitly specify the number of rounds of the 160-bit version of the Spongent permutation; we opt for 80 rounds since this ensures that at least 160 S-boxes are differentially active. This is in accordance with the Spongent design strategy. Note further that this implies that the 7-bit LFSR specified in [21] should be used (with initial value 0x75) to generate the round constants for the permutation.

The LFSRs of both instances aim to minimize the area required when implemented in hardware. In particular, in addition to the shift register, only two 2-bit XOR gates are needed. Hence, these choices of LFSRs are in line with the strength of the Spongent permutations, making a perfect match for small area hardware implementations. Despite the particular suitability of both LFSRs for small area hardware implementations, it is still possible to implement them rather efficiently on 8-bit platforms.

4.3.2 Delirium

The 200-bit instance of Elephant is based on the Keccak-$f[200]$ permutation [14]. The 200-bit instance is the smallest of the instances in the NIST standard [47] that fits our need; it is still reasonable in hardware, and particularly good in software on 8-bit platforms, considering that it is naturally defined using 8-bit lanes [16,56]. As such, it is complementary to the Spongent-based instantiation of Elephant.

This LFSR shows its full potential when implemented on 8-bit platforms. A state update within the LFSR just updates one byte, while the content of the other 24 bytes is not changed and basically just relabeled. The single updated byte is computed as the XOR sum of 3 bytes other state bytes that are just

\[^1\text{Beyond birthday bound solutions may use even smaller permutations, but only at an efficiency penalty.}\]
rotated or shifted by one bit position. Hence, the essential operations that have
to be performed on 8-bit platforms are 3 XOR operations, two rotations by one
bit to the left plus one shift by one bit to the left.

4.4 Implementation

As discussed in Section 4.1, the Elephant mode allows for a high degree of parallelism. For the hardware-oriented variants of Elephant (Dumbo and Jumbo), this makes it easy to trade-off area for additional throughput. Hardware implementations of the 176-bit Spongent permutation are given by Bogdanov et al. [21], e.g., just needing 1329 GE to implement the Spongent-160 hash function, which is based on the 176-bit Spongent permutation. The 200-bit variant of Elephant primarily targets (embedded) software, but the same remarks concerning hardware implementations apply as, e.g., demonstrated by an implementation of a hash function based on the 200-bit Keccak permutation needing just 2520 GE by Kavun and Yalçın [56].

Software implementations of 200-bit Elephant (Delirium) can also exploit parallelism. If multiple cores are available, several blocks can be processed concurrently – but this is only useful for long messages. More importantly, on processors with a word size above 16 bits, the available parallelism makes it possible to increase the efficiency of the implementation by combining two or more calls to the Keccak permutation. For mid- and high-end processors with SIMD instructions, the same technique can be used to obtain even greater speed-ups.

An increasingly common requirement is the ability to protect implementations against side-channel attacks. As discussed in Section 4.2, the masking scheme is constant time by design. The same applies to the Spongent and Keccak permutations. In addition, all variants of Elephant are well-suited for Boolean masking techniques such as threshold implementations [75].

Finally, it is worth mentioning that a few specific use-cases of Elephant allow for additional optimizations. As discussed in Section 4.2, the contribution of the mask values to the tag can be precomputed for fixed-length messages. In addition, if one or more blocks of associated data are static, it is possible to precompute their contribution to the tag – with the exception of the first block, which involves the nonce.

A reference implementation of Dumbo, Jumbo, and Delirium written in C99 can be found at https://github.com/TimBeyne/Elephant.

5 Summary of Known Cryptanalytic Attacks

After briefly reviewing security aspects of the generic Elephant mode in Section 5.1, we discuss the main cryptanalytic results on Spongent in Section 5.2, and on Keccak in Section 5.3.
5.1 Generic Mode

In Appendix B, we prove that the generic mode of Elephant, based on a tweakable block cipher, is secure. The security proof is standard, and it builds among others on ideas of Bellare and Namprempre [9] and Namprempre et al. [71] (for insights in the encrypt-then-MAC approach), and Bernstein [10] (for insights in the Wegman-Carter-Shoup MAC mode). The analysis of the underlying tweakable block cipher, in turn, builds on Granger et al. [49].

5.2 Spongent Permutation

We discuss the main known cryptanalytic results in detail, and refer to Appendix A.1 for a complete list.

Differential Cryptanalysis. The following result of Bogdanov et al. [22] provides a lower bound on the number of active S-boxes in any differential characteristic of Spongent-π[b] with b ≥ 64. The result and its proof are similar to those for the block cipher PRESENT [23].

Theorem 5.1 (Theorem 1 of Bogdanov et al. [22].) Any 5-round differential characteristic of Spongent-π[b] with b ≥ 64 involves at least 10 differentially active S-boxes.

Theorem 5.1 implies that after r rounds of Spongent-π[b] with b ≥ 64, at least 2r S-boxes are differentially active. Since the S-box is differentially 4-uniform, it follows that the probability of any r-round characteristic is at most 2−4r.

Note that the number of rounds of Spongent-π[b] is determined such that at least b S-boxes are differentially active [22]. Equivalently, Spongent-π[b] should have at least b/2 rounds.

More rounds can be attacked by relying on truncated differentials. For example, for b = 176, Zhang and Liu [94] presented a 46-round truncated differential with (marginally) significant probability. These properties are derived from multidimensional linear approximations, following Blondeau and Nyberg [20]. In the next section, linear approximations are discussed in more detail.

In conclusion, (truncated) differential cryptanalysis does not threaten full-round Spongent-π[b], for neither b = 160 nor b = 176. In addition, one should keep in mind that many of the best reduced-round distinguishers require more data than is allowed to be processed by the Elephant mode (i.e., no more than 2^{47} chosen plaintexts).

Linear Cryptanalysis. In order to assess the security of the permutation Spongent-π[b] against linear cryptanalysis, we follow the approach used by Bogdanov et al. [22]: rather than computing only the correlation of individual trails, the correlation of linear approximations will be estimated. Previous work has shown that 1-bit (per round) trails are dominant in PRESENT-like designs [30, 61], meaning that one can estimate the correlation of all 1-bit linear approximations over r rounds by computing the product of r sparse matrices of
size \( b \times b \). Table 1 shows the resulting estimates, where \( c_r \) denotes the maximum absolute correlation after \( r \) rounds.

Table 1: Estimated maximum correlation of linear approximations of \( \text{Spongent-} \pi[b] \) with \( b \in \{160, 176\} \). The total number of rounds is denoted by \( R \) (that is, \( R = 80 \) for \( b = 160 \) and \( R = 90 \) for \( b = 176 \)).

<table>
<thead>
<tr>
<th>( b )</th>
<th>( c_{40} )</th>
<th>( c_{44} )</th>
<th>( c_R )</th>
</tr>
</thead>
<tbody>
<tr>
<td>160</td>
<td>( 2^{-80} )</td>
<td>( 2^{-88} )</td>
<td>( 2^{-160} )</td>
</tr>
<tr>
<td>176</td>
<td>( 2^{-80} )</td>
<td>( 2^{-88} )</td>
<td>( 2^{-180} )</td>
</tr>
</tbody>
</table>

The estimates in Table 1 could be improved by taking into account additional trails. For example, Abdelraheem [1] gives improved estimates by taking into account all trails with at most four linearly active S-boxes per round. This yields slightly improved distinguishers in some cases, but still covering at most one or two additional rounds.

The results above imply that full-round \( \text{Spongent-} \pi[b] \) is not threatened by linear attacks, statistical saturation attacks, or multidimensional linear attacks [30, 32]. As for differential cryptanalysis, it should be remarked that the security margin remains large, especially because even the reduced-round distinguishers typically require more data than the \textbf{Elephant} mode can securely process.

**Integral Cryptanalysis.** Division properties of \( \text{Spongent-} \pi[b] \) have been analyzed to some extent, in particular for \( b = 88 \) [46, 88, 89]. Eskandari et al. [46] built a SAT-solver based tool to find, or show the absence of, division properties. They use this tool to show that \( \text{Spongent-} \pi[176] \) does not have a bit-based division property covering 12 rounds or more. It was verified that the same holds for \( \text{Spongent-} \pi[160] \).

It is often possible to setup a distinguisher that covers more rounds, by starting from the middle of the permutation and extending the division property in the forward and backward direction. For example, Sun et al. [89] presented a zero-sum distinguisher for 21 rounds of \( \text{Spongent-} \pi[160] \) requiring \( 2^{159} \) data. Remark that even this reduced-round distinguisher far exceeds the data limits imposed for \textbf{Elephant}.

We now discuss the ramifications of the above results in the context of impossible differentials and zero-correlation linear approximations, by relying on a result of Sun et al. [87]. Sun et al. demonstrated that a nontrivial zero-correlation linear approximation of a permutation constructively implies the existence of an integral distinguisher. They furthermore demonstrated that, as \( \text{Spongent-} \pi[b] \) has a bitwise (hence self-dual) linear layer, one can conclude that for (round-reduced) \( \text{Spongent-} \pi[b] \), any nontrivial impossible differential that does not depend on the choice of the S-box constructively implies the existence of an integral distinguisher.
It can be concluded that Spongent has a very large margin against integral-type distinguishers. The same applies to zero correlation linear approximations and impossible differentials (not relying on the S-box structure), due to their links with integral properties.

5.3 Keccak Permutation

We discuss the main known cryptanalytic results in detail, and refer to Appendix A.2 for a complete list.

Differential Cryptanalysis. The differential properties of the permutation Keccak-f[200] have been extensively analyzed and no significant differential distinguishers are expected to exist [14, 35, 65]. Due to Keccak’s weak alignment [15], there are no known analytic upper bounds on the probability of differential characteristics. Instead, computer assistance is required to determine bounds.

The analysis in the Keccak reference [14] leads to lower bounds on the weight of symmetric characteristics in Keccak-f – remark that Keccak-f[200] characteristics are symmetric by definition. The results are summarized in the first three rows of Table 2. Improved bounds are presented by Mella, Daemen, and Van Assche [65] based on a dedicated search algorithm. For the characteristics corresponding to the lower bounds in Table 2, the reader is referred to Table 3 of [65].

Table 2: Lower and upper bounds on the minimum weight of differential characteristics in Keccak-f[200] [14, 65].

<table>
<thead>
<tr>
<th>Rounds</th>
<th>Lower bound</th>
<th>Upper bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>4</td>
<td>46</td>
<td>46</td>
</tr>
<tr>
<td>5</td>
<td>50</td>
<td>89</td>
</tr>
<tr>
<td>6</td>
<td>92</td>
<td>142</td>
</tr>
<tr>
<td>18</td>
<td>276</td>
<td>——</td>
</tr>
</tbody>
</table>

Of course, the lack of high probability differential characteristics need not imply that all differentials have low probability. Bertoni et al. [15] argue that clustering of 2-round characteristics is prevented by weak alignment. This means that the propagation of differentials does not respect cell-boundaries in Keccak. Weak alignment leads the authors of Keccak to believe that it is unlikely that truncated differentials can be successfully exploited [15].

Linear Cryptanalysis. The Keccak reference [14] provides lower bounds on the weight of linear trails, where the weight of a linear trail equals minus the logarithm of the square of its correlation. These bounds are listed in Table 3.
The lower bound for full-round Keccak-f[200] is 204, corresponding to a correlation which is only slightly smaller than the variance of the correlation of linear approximations in a random permutation. It should be emphasized that 204 is a rather rough lower bound, and the true minimum weight is expected to be much larger.

As in the case of differential cryptanalysis, Bertoni et al. [15] provide arguments against clustering of linear trails based on Keccak's weak alignment.

Table 3: Lower and upper bounds on the minimum weight of linear trails in Keccak-f[200] [14].

<table>
<thead>
<tr>
<th>Rounds</th>
<th>Lower bound</th>
<th>Upper bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>4</td>
<td>46</td>
<td>46</td>
</tr>
<tr>
<td>18</td>
<td>204</td>
<td>—–</td>
</tr>
</tbody>
</table>

**Attacks Exploiting Algebraic Degree.** For keyed instances that use variants of Keccak-f, such as Ketje [18] and Keyak [17], the attacks covering the highest number of rounds typically exploit the algebraic degree, e.g., cube [41], cube-like [40], or conditional cube attacks [53]. In the case of Ketje Jr., that builds on a round-reduced version of Keccak-f[200], those attacks can cover up to 6 rounds [83]. If we take a broader look at constructions that use bigger variants of Keccak-f, and also allow the attacker more degrees of freedom in placing the cube variables, those attacks usually lie in the region of 8 rounds [19,40,43,53,85] considering a targeted security level of 128-bits. Since Keccak-f[200] used in Delirium has 18 rounds, we have a huge security margin against this type of attacks.

**ACKNOWLEDGMENTS.** This work was supported in part by the Research Council KU Leuven: GOA TENSE (C16/15/058). Yu Long Chen is supported by a Ph.D. Fellowship from the Research Foundation - Flanders (FWO). Christoph Dobrämig is supported by the Austrian Science Fund (FWF): J 4277-N38. Bart Mennink is supported by a postdoctoral fellowship from the Netherlands Organisation for Scientific Research (NWO) under Veni grant 016.Veni.173.017.

**References**


[47] FIPS 202: SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions (August 2015)


[72] National Institute of Standards and Technology (NIST): Submission requirements and evaluation criteria for the lightweight cryptography standardization process (Aug 2018)


[80] Saha, D., Kuila, S., Chowdhury, D.R.: SymSum: Symmetric-Sum Distinguishers Against Round Reduced SHA3. IACR Transactions on Symmetric


A List of Cryptanalysis

A.1 Spongent Permutation


A.2 Keccak


B Security of Elephant Mode

We describe the security model in Section B.1, introduce a simplified version of masked Even-Mansour in Section B.2, and state the formal security result on Elephant in Section B.3. We discuss the implication of this result for the three instances Dumbo, Jumbo, and Delirium in Section B.4.

B.1 Security Model

For a finite set $T$, we denote by $\text{perm}(n)$ the set of all $n$-bit permutations and by $\text{perm}(T, n)$ the set of all families of permutations indexed by $T \in T$. For a finite set $S$, we denote by $s \leftarrow S$ the uniform random sampling of an element $s$ from $S$.

An adversary $A$ is an algorithm that is given access to one or more oracles $O$, and after interaction with $O$ it outputs a bit $b \in \{0, 1\}$. This event is denoted as $A^O \rightarrow b$. In our work, we will be concerned with computationally unbounded adversaries $A$; their complexities are only measured by the amount of oracle queries. For two randomized oracles $O, P$, we denote the advantage of an adversary $A$ in distinguishing both by

$$\Delta_A(O : P) = \Pr(A^O \rightarrow 1) - \Pr(A^P \rightarrow 1).$$

Finally, let $k, m, n, t \in \mathbb{N}$ with $k, m, t \leq n$ throughout.

B.1.1 Authenticated Encryption

An authenticated encryption scheme $AE$ consists of two algorithms $\text{enc}$ and $\text{dec}$. Encryption $\text{enc}$ gets as input a key $K \in \{0, 1\}^k$, a nonce $N \in \{0, 1\}^m$, associated data $A \in \{0, 1\}^*$, and a message $M \in \{0, 1\}^*$, and it outputs a ciphertext $C \in \{0, 1\}^{\lfloor M \rfloor}$ and a tag $T \in \{0, 1\}^t$. Decryption $\text{dec}$ gets as input a key $K \in \{0, 1\}^k$, a nonce $N \in \{0, 1\}^m$, associated data $A \in \{0, 1\}^*$, a ciphertext $C \in \{0, 1\}^*$, and a tag $T \in \{0, 1\}^t$, and it outputs a message $M \in \{0, 1\}^{\lfloor C \rfloor}$ if the tag is correct, or a dedicated ⊥-sign otherwise. The two functions are required to satisfy

$$\text{dec}(K, N, A, \text{enc}(K, N, A, M)) = M.$$
security of AE against an adversary $A$ is defined as
\[
\text{Adv}_{\text{AE}}^\text{ae}(A) = \Delta_A \left( \text{enc}_K, \text{dec}_K, P^\pm ; \text{rand}, \bot, P^\pm \right),
\]
where the randomness of the oracles is taken over $K \leftarrow \{0, 1\}^k$, $P \leftarrow \text{perm}(n)$, and the function $\text{rand}$ that for each input $(N, A, M)$ returns a random string of size $|M| + t$ bits. The function $\bot$ returns the $\bot$-sign for each query.

We only consider nonce-respecting adversaries: $A$ is not allowed to make two encryption queries for the same nonce. It is also not allowed to relay the output of the encryption oracle (\text{enc}_K in the real world and $\text{rand}$ in the ideal world) to the decryption oracle (\text{dec}_K in the real world and $\bot$ in the ideal world).

**B.1.2 Tweakeable Block Ciphers**

A tweakable block cipher $\tilde{E}$ is a function that gets as input a key $K \in \{0, 1\}^k$, tweak $T \in T$, and message $M \in \{0, 1\}^n$, and it outputs a ciphertext $C \in \{0, 1\}^n$. The tweakable block cipher is required to be bijective for any fixed $(K, T)$.

In our application, we will not make use of the inverse $\tilde{E}^{-1}$. More importantly, for our authenticated encryption scheme it suffices to use a tweakable block cipher that is secure against adversaries that only have access to $\tilde{E}$, and not to $\tilde{E}^{-1}$. The tweakable block cipher considered in this work is based on an $n$-bit permutation $P$, which is modeled as a random permutation: $P \leftarrow \text{perm}(n)$.

The security of $\tilde{E}$ against an adversary $A$ is defined as
\[
\text{Adv}_{\tilde{E}}^\text{ipn}(A) = \Delta_A \left( \tilde{E}_K, P^\pm ; \tilde{\pi}, P^\pm \right),
\]
where the randomness of the oracles is taken over $K \leftarrow \{0, 1\}^k$, $P \leftarrow \text{perm}(n)$, and $\tilde{\pi} \leftarrow \text{perm}(T, n)$.

**B.2 Simplified Masked Even-Mansour**

The Elephant authenticated encryption family uses its underlying permutation in a “Masked Even-Mansour” (MEM) construction [49]: the input to and output of the permutation $P$ are masked using an LFSR evaluated on the secret key. However, the tweakable block cipher used in our proposal is simpler than the original construction in two ways: (i) the tweak only consists of the exponents of the LFSRs and not the nonce and (ii) in our application, the tweakable block cipher is only evaluated in the forward direction. The changes are not huge, but they do allow for a simpler description, security analysis, and bound. We will refer to this scheme as SiM (Simplified MEM). For generality, we will keep the formalization for an arbitrary amount of LFSRs, even though we will only use it for two LFSRs.

\footnote{In our application, the tweak space is of a specific form and cannot be conveniently expressed as a set of binary strings.}
B.2.1 Specification

Let $k, n, z \in \mathbb{N}$. Let $P \in \text{perm}(n)$ be an $n$-bit permutation, and let $\varphi_1, \ldots, \varphi_z : \{0,1\}^n \rightarrow \{0,1\}^n$ be $z$ LFSRs. Let $T \subseteq \mathbb{N}^2$ be a finite tweak space. Define the function $\text{mask} : \{0,1\}^k \times T \rightarrow \{0,1\}^n$ as follows:

$$\text{mask}_{K}^{a_1, \ldots, a_z} = \text{mask}(K, a_1, \ldots, a_z) = \varphi_z^{a_z} \circ \cdots \circ \varphi_1^{a_1} \circ P(K\|0^{n-k}) .$$

We define the tweakable block cipher $\text{SiM} : \{0,1\}^k \times T \times \{0,1\}^n \rightarrow \{0,1\}^n$ as

$$\text{SiM}(K, (a_1, \ldots, a_z), M) = P(M \oplus \text{mask}_{K}^{a_1, \ldots, a_z} \oplus \text{mask}_{K}^{a_1, \ldots, a_z} .$$

B.2.2 Security of SiM

We need a restriction on the tweak space $T$ in order for $\text{SiM}$ to be a secure tweakable block cipher. As Granger et al. [49], we say that $T$ is $2^{-\alpha}$-proper with respect to $(\varphi_1, \ldots, \varphi_z)$ if the function $L \mapsto \varphi_z^{a_z} \circ \cdots \circ \varphi_1^{a_1}(L)$ is $2^{-\alpha}$-uniform and $2^{-\alpha}$-XOR-uniform.

**Definition B.1.** Let $n, z \in \mathbb{N}$. Let $\varphi_1, \ldots, \varphi_z : \{0,1\}^n \rightarrow \{0,1\}^n$ be $z$ LFSRs. The tweak space $T$ is called $2^{-\alpha}$-proper with respect to $(\varphi_1, \ldots, \varphi_z)$ if the following two properties hold:

1. For any $Y \in \{0,1\}^n$ and $(a_1, \ldots, a_z) \in T \cup \{(0, \ldots, 0)\},$

$$\Pr \left( L \overset{\$}{\leftarrow} \{0,1\}^n : \varphi_z^{a_z} \circ \cdots \circ \varphi_1^{a_1}(L) = Y \right) \leq 2^{-\alpha} ;$$

2. For any $Y \in \{0,1\}^n$ and distinct $(a_1, \ldots, a_z), (a_1', \ldots, a_z') \in T \cup \{(0, \ldots, 0)\},$

$$\Pr \left( L \overset{\$}{\leftarrow} \{0,1\}^n : \varphi_z^{a_z} \circ \cdots \circ \varphi_1^{a_1}(L) \oplus \varphi_z^{a_z'} \circ \cdots \circ \varphi_1^{a_1'}(L) = Y \right) \leq 2^{-\alpha} .$$

In Section C, we will prove Theorem B.2, which says that if the tweak space is $2^{-\alpha}$-proper for sufficiently small $2^{-\alpha}$ (note that $2^{-\alpha}$ cannot be smaller than $2^{-n}$), then $\text{SiM}$ is a secure tweakable block cipher. The proof is a direct simplification of Granger et al.’s analysis of MEM [49], due to the changes described in the introductory text of Section B.2. These simplifications allow us to derive a slightly improved bound on the advantage, noting for comparison that Granger et al. [49] proved security up to $(4.5q^2 + 3qp)/2^n + p/2^k$.

**Theorem B.2.** Let $k, n, z \in \mathbb{N}$. Let $\varphi_1, \ldots, \varphi_z : \{0,1\}^n \rightarrow \{0,1\}^n$ be $z$ LFSRs, and let $T$ be a $2^{-\alpha}$-proper tweak space with respect to $(\varphi_1, \ldots, \varphi_z)$. Consider $\text{SiM}$ of (12) based on random permutation $P \overset{\$}{\leftarrow} \text{perm}(n)$. For any adversary $A$ making at most $q \leq 2^{n-1}$ construction queries and $p$ primitive queries,

$$\text{Adv}_{\text{SiM}}^{\text{prp}}(A) \leq \frac{q^2 + 2qp}{2^n} + \frac{2q + p}{2^n} + \frac{p}{2^k} .$$

The proof is given in Section C.
B.3 Security of Elephant

We will prove security of Elephant of Section 2 for any $2^{-n}$-proper tweak space. The specific choice of tweak space for the three instances of Elephant will be discussed in Section B.4.

**Theorem B.3.** Let $k, m, n, t \in \mathbb{N}$ with $k, m, t \leq n$. Let $\varphi_1, \varphi_2 : \{0, 1\}^n \rightarrow \{0, 1\}^n$ be LFSRs, and let $T$ be a $2^{-n}$-proper tweak space with respect to $(\varphi_1, \varphi_2)$. Consider Elephant = (enc, dec) of Section 2 based on random permutation $P \leftarrow \text{perm}(n)$. For any adversary $A$ making at most $q_e \leq 2^n - 1$ construction encryption queries, $q_d$ construction decryption queries, each query at most $\ell$ padded nonce and associated data and message blocks, and in total at most $\sigma$ padded nonce and associated data and message blocks, and $p$ primitive queries,

$$\text{Adv}^\text{ac}_{\text{Elephant}}(A) \leq \ell \left( \frac{q_e}{2^n} + \frac{2^{n-t}q_d}{2^n - 1} \right) \leq \text{Adv}^\text{tprp}_{\text{SIM}}(A'),$$

for some $A'$ that makes $2\sigma$ construction queries and $p$ primitive queries.

The proof is given in Section D.

B.4 Implication for Dumbo, Jumbo, and Delirium

B.4.1 Dumbo: 160-Bit Elephant

We will prove that the 160-bit LFSR defined by (3) has maximal length, and that the tweak space used in Elephant with this LFSR is $2^{-n}$-proper with respect to $(\varphi_1, \varphi_2)$.

**Proposition B.4.** Let $n = 160$. Let $\varphi_1 : \{0, 1\}^{160} \rightarrow \{0, 1\}^{160}$ be the LFSR given in (3), and $\varphi_2 = \varphi_1 \oplus \text{id}$. The tweak space $T = T_1 \times T_2 = \{0, 1, \ldots, 2^{154}\} \times \{0, 1, 2\}$ is $2^{-n}$-proper with respect to $(\varphi_1, \varphi_2)$.

**Proof.** The proof is almost identical to [49, Lemma 4], with the main difference that a different discrete logarithm must be computed. Let $V$ be the $160 \times 160$ matrix over $\mathbb{F}_2$ that represents $\varphi_1$ of (3). As shown in [49, Lemma 3], $\varphi_1(L) = V^t \cdot L$ is $2^{-n}$-proper for $i \in \{0, \ldots, 2^{n} - 2\}$ if the minimal polynomial of $V$ is primitive and of degree $n$. A quick computation using Sage [90] shows that this polynomial

$$p(x) = x^{160} + x^{136} + x^{83} + x^{53} + 1$$

is irreducible and primitive.

Next, let $\ell = \log_2(x + 1)$ in the field $\mathbb{F}_2[x]/p(x)$. We have to show that $\varphi_3 \circ \varphi_1(L) = (V + I)^b \cdot V^a \cdot L = V^\ell \cdot V^a \cdot L$ is unique for any distinct set of tweaks. A simple Sage computation gives the following values for $\ell$ and $2\ell$:

$$\ell = 74280011654209447488264356271465075847453684889 \approx 2^{159.02}$$

$$2\ell = 24098595753286031561602292713018497293140826803 \approx 2^{154.08}.$$

37
If we consider that \( b \in \{0, 1, 2\} \) divides the tweak space into three sets, the smallest difference is between the set with \( b = 0 \) and the set corresponding to \( b = 2 \), which is bigger than \( 2^{154} \). Hence, by ensuring that \( 0 \leq a \leq 2^{154} \), we have that for any two distinct \((a, b), (a', b') \in \{0, 1, \ldots, 2^{154}\} \times \{0, 1, 2\}\), \( \phi_2^b \circ \phi_1^a \neq \phi_2^{b'} \circ \phi_1^{a'} \).

Finally, using both of the above observations, one can easily observe that \( T \) is \( 2^{-n} \)-proper in light of Definition B.1.

We directly obtain that Dumbo is secure in the random permutation model.

**Corollary B.5.** Let \((k, m, n, t) = (128, 96, 160, 64)\). Let \( T = \{0, 1, \ldots, 2^{154}\} \times \{0, 1, 2\}\). Consider Dumbo: Elephant = (enc, dec) of Section 2 based on Spongent-\(\pi[160]\), modeled as a random 160-bit permutation, and on \( \phi_1: \{0, 1\}^{160} \to \{0, 1, 2\} \) of (3). For any adversary \( A \) making at most \( q_e \) construction encryption queries, \( q_d \) construction decryption queries, each query at most \( \ell \) padded nonce and associated data and message blocks, and in total at most \( \sigma \leq 2^{158} \) padded nonce and associated data and message blocks, and \( p \) primitive queries,

\[
\text{Adv}^\text{sec}_{\text{Dumbo}}(A) \leq \frac{\ell \left( \frac{q_e}{2} \right)^2}{2^{160}} + \frac{3 q_d}{2^{160} - 1} e^{\alpha_e + 1} e^{2^{160}} + 4 \sigma + 4 \sigma + p \frac{2^{128}}{2^{160}}.
\]

Recall that NIST’s call for lightweight authenticated encryption schemes [72] requested security up to an online complexity of around \( 2^{50} \) bytes. By limiting the total online complexity \( \sigma \) to \( 2^{50}/(n/8) \) blocks, the bound of Corollary B.5 is at most 1 for \( p \leq 2^{112} \).

**B.4.2 Jumbo: 176-Bit Elephant**

We will prove that the 176-bit LFSR defined by (4) has maximal length, and that the tweak space used in Elephant with this LFSR is \( 2^{-n} \)-proper with respect to \((\phi_1, \phi_2)\).

**Proposition B.6.** Let \( n = 176 \). Let \( \phi_1: \{0, 1\}^{176} \to \{0, 1\}^{176} \) be the LFSR given in (4), and \( \phi_2 = \phi_1 \circ \text{id} \). The tweak space \( T = T_1 \times T_2 = \{0, 1, \ldots, 2^{173}\} \times \{0, 1, 2\} \) is \( 2^{-n} \)-proper with respect to \((\phi_1, \phi_2)\).

**Proof.** The proof is identical to that of Proposition B.4, with the difference that for the \( 176 \times 176 \) matrix \( V \) that represents \( \phi_1 \) of (4), the corresponding polynomial

\[
p(x) = x^{176} + x^{154} + x^{135} + x^{19} + 1
\]

is irreducible and primitive. The discrete logarithm \( \ell = \log_{\mathbb{F}_2}(x + 1) \) in the field \( \mathbb{F}_2[x]/p(x) \) and its related \( 2\ell \) are computed as

\[
\ell = 188813761514037867774146343209450294100461562220699 \approx 2^{173.66},
\]

\[
2\ell = 3776275230280757355496292686405890058820092312441398 \approx 2^{174.66}.
\]
Again, dividing the tweak space into 3 sets according to the value \( b \in \{0, 1, 2\} \),
the smallest difference is between set \( b = 0 \) and set \( b = 1 \) or \( b = 1 \) and \( b = 2 \),
which is bigger than \( 2^{173} \). Hence, by ensuring that \( 0 \leq a \leq 2^{173} \), we have that for any two distinct \((a, b), (a', b') \in \{0, 1, \ldots, 2^{173}\} \times \{0, 1, 2\}, \varphi_2^b \circ \varphi_1^a \neq \varphi_2^{b'} \circ \varphi_1^{a'}\). □

We directly obtain that Jumbo is secure in the random permutation model.

**Corollary B.7.** Let \((k, m, n, t) = (128, 96, 176, 64)\). Let \( T = \{0, 1, \ldots, 2^{173}\} \times \{0, 1, 2\} \). Consider Jumbo: Elephant = (enc, dec) of Section 2 based on Spongents-\(π|176\), modeled as a random 176-bit permutation, and on \( \varphi_1 : \{0, 1\}^{176} \to \{0, 1\}^{176} \) of (4). For any adversary \( A \) making at most \( q_c \) construction encryption queries, \( q_d \) construction decryption queries, each query at most \( \ell \) padded nonce and associated data and message blocks, and in total at most \( \sigma \leq 2^{174} \) padded nonce and associated data and message blocks, and \( p \) primitive queries,

\[
\text{Adv}_{\text{Jumbo}}(A) \leq \ell \left( \frac{q_c}{2} \right)^{176} + \frac{2^{112} q_d}{2^{176} - 1} e^{(n+1)q_c/2^{176}}
+ \frac{4\sigma^2 + 4\sigma p + 4\sigma + p}{2^{176}} + \frac{p}{2^{128}},
\]

As before, limiting the total online complexity \( \sigma \) to \( 2^{50}/(n/8) \) blocks, the bound of Corollary B.7 is at most 1 for \( p \leq 2^{127} \).

**B.4.3 Delirium: 200-Bit Elephant**

We will prove that the 200-bit LFSR defined by (5) has maximal length, and that the tweak space used in Elephant with this LFSR is \( 2^{-n} \)-proper with respect to \((\varphi_1, \varphi_2)\).

**Proposition B.8.** Let \( n = 200 \). Let \( \varphi_1 : \{0, 1\}^{200} \to \{0, 1\}^{200} \) be the LFSR given in (5), and \( \varphi_2 = \varphi_1 \oplus \text{id} \). The tweak space \( T = T_1 \times T_2 = \{0, 1, \ldots, 2^{107}\} \times \{0, 1, 2\} \) is \( 2^{-n} \)-proper with respect to \((\varphi_1, \varphi_2)\).

**Proof.** The proof is identical to that of Proposition B.4, with the difference that for the \( 200 \times 200 \) matrix \( V \) that represents \( \varphi_1 \) of (5), the corresponding polynomial

\[
p(x) = x^{200} + x^{93} + x^{91} + x^{82} + x^{78} + x^{71} + x^{69} + x^{67} + x^{65}
+ x^{60} + x^{52} + x^{49} + x^{47} + x^{41} + x^{39} + x^{38} + x^{34} + x^{30} + x^{27}
+ x^{26} + x^{25} + x^{23} + x^{21} + x^{19} + x^{17} + x^{16} + x^{15} + x^{13} + 1
\]

is irreducible and primitive. The discrete log \( \ell = \log_2(x + 1) \) in the field \( F_2[x]/p(x) \) and its related \( 2\ell \) are computed as

\[
\ell = 69218606625676931900534627786122994390018641930530681719698
\approx 2^{198.78},
\]

\[
2\ell = 138436121325135386380106925557224598878003728386106363439396
\approx 2^{199.78}.
\]
Again, dividing the tweak space into 3 sets according to the value \( b \in \{0, 1, 2\} \), the smallest difference is between set \( b = 2 \) and set \( b = 0 \), which is bigger than \( 2^{197} \). Hence, by ensuring that \( 0 \leq a \leq 2^{197} \), we have that for any two distinct \((a, b), (a', b') \in \{0, 1, \ldots, 2^{197}\} \times \{0, 1, 2\}, \varphi_2^b \circ \varphi_1^b \neq \varphi_2^{b'} \circ \varphi_1^{b'} \).

We directly obtain that Delirium is secure in the random permutation model.

**Corollary B.9.** Let \((k, m, n, t) = (128, 96, 200, 128)\). Let \( \mathcal{T} = \{0, 1, \ldots, 2^{197}\} \times \{0, 1, 2\} \). Consider Delirium: Elephant = (enc, dec) of Section 2 based on Keccak-f\([200]\), modeled as a random 200-bit permutation, and on \( \varphi_1 : \{0, 1\}^{200} \to \{0, 1\}^{200} \) of (5). For any adversary \( A \) making at most \( q_e \) construction encryption queries, \( q_d \) construction decryption queries, each query at most \( \ell \) padded nonce and associated data and message blocks, and in total at most \( \sigma \leq 2^{198} \) padded nonce and associated data and message blocks, and \( p \) primitive queries,

\[
\text{Adv}_{\text{Delirium}}^\text{ae}(A) \leq \ell \left( \frac{q_e}{2} \right) / \ell^{200} + \left( \frac{2^{72} q_d}{\ell^{200}} - 1 \right) \epsilon(q_e+1) q_e / \ell^{200} + \frac{4 \sigma^2 + 4 \sigma p + 4 \sigma + p}{\ell^{200}} + \frac{p}{2^{128}},
\]

As before, limiting the total online complexity \( \sigma \) to \( 2^{74} / (n/8) \) blocks, the bound of Corollary B.9 is at most 1 for \( p \leq 2^{127} \).

### C Proof of Theorem B.2 (on SiM)

The proof closely follows Granger et al. [49] and is performed using the H-coefficient technique [29, 76].

Let \( K \leftarrow \{0, 1\}^k \), \( P \leftarrow \text{perm}(n) \), and \( \pi \leftarrow \text{perm}(\mathcal{T}, n) \), where \( \mathcal{T} \) is \( 2^{-\alpha} \)-proper with respect to LFSRs \((\varphi_1, \ldots, \varphi_z)\). Consider a computationally unbounded adversary \( A \) that tries to distinguish \( O := (\mathcal{E}_K^\pi, P^\pm) \) from \( P := (\pi, P^\pm) \). Without loss of generality, we can consider it to be deterministic: for any probabilistic adversary there exists a deterministic one that has at least the same success probability. The interaction of \( A \) with its oracle \((O \text{ or } P)\) is gathered in a view \( \nu \). Denote by \( D_O \) (resp., \( D_P \)) the probability distribution of views in interaction with \( O \) (resp., \( P \)). Denote by \( V \) the set of “attainable views”, i.e., views \( \nu \) such that \( \Pr(D_P = \nu) > 0 \).

**Lemma C.1** (H-coefficient technique). Consider a partition \( V = V_{\text{good}} \cup V_{\text{bad}} \) of the set of views into “good” and “bad” views. Let \( \varepsilon \in [0, 1] \) be such that

\[
\frac{\Pr(D_O = \nu)}{\Pr(D_P = \nu)} \geq 1 - \varepsilon \text{ for all } \nu \in V_{\text{good}}.
\]

Then,

\[
\Delta_A(O \mid P) \leq \varepsilon + \Pr(D_P \in V_{\text{bad}}).
\]

For view \( \nu = \{(x_1, y_1), \ldots, (x_q, y_q)\} \) consisting of \( q \) input/output tuples, we denote by \( O \vdash \nu \) the event that oracle \( O \) satisfies that \( O(x_i) = y_i \) for all \( i = 1, \ldots, q \).
The remainder of the proof is structured as follows. We specify the views of an adversary in Section C.1 and define the bad views in Section C.2. The probability of bad views is analyzed in Section C.3 and the probability ratio for good views is considered in Section C.4. Section C.5 concludes the proof.

C.1 Views

The adversary can make \( q \) construction queries to \( \tilde{E}_K \) or \( \tilde{\pi} \), all in forward direction only. Each such query is made for some tweak \( \bar{a}_i = (a_1, \ldots, a_z)_i \) and message input \( M_i \), and results in an output \( C_i \). The \( q \) queries are summarized in a view

\[
\nu_c = \{(\bar{a}_1, M_1, C_1), \ldots, (\bar{a}_q, M_q, C_q)\}.
\]

The adversary can make \( p \) primitive queries to \( P^\pm \), and these are likewise summarized in a view

\[
\nu_p = \{(X_1, Y_1), \ldots, (X_p, Y_p)\}.
\]

After the conversation of \( A \) with its oracle, but before it makes its final decision, we reveal the key material used in the interaction. This can be done without loss of generality; it only improves the adversarial success probability. The first value that is revealed is a value \( K \). In the real world, this is the key \( K \leftarrow \{0,1\}^k \) that is actually used by the construction oracle; in the ideal world, it is a dummy key \( K \leftarrow \{0,1\}^k \). The second value that is revealed is a value \( L \in \{0,1\}^n \). In the real world, it is the value \( L = P(K \parallel 0^n\_k) \); in the ideal world, it is a dummy key \( L \leftarrow \{0,1\}^n \). The revealed data is summarized in a view

\[
\nu_k = \{(K, L)\}.
\]

The complete view is defined as \( \nu = (\nu_c, \nu_p, \nu_k) \). We assume that the adversary never makes any duplicate query, hence \( \nu_c \) and \( \nu_p \) contain no duplicate elements.

C.2 Definition of Good and Bad Views

In the real world, all tuples in \( \nu_p \) define exactly one input-output pair for \( P \). Likewise, the sole tuple in \( \nu_k \) is an input-output pair for \( P \). Using this tuple, one can observe that any tuple \( (\bar{a}_i, M_i, C_i) \in \nu_c \) also defines an input-output pair for \( P \), namely

\[
(M_i \oplus \text{mask}_{K}^{\bar{a}_i}, C_i \oplus \text{mask}_{K}^{\bar{a}_i}).
\]

If among all these \( q + p + 1 \) input-output pairs defined by \( \nu \), there are two that have colliding input or output values, we consider \( \nu \) to be a bad view. The

\[\text{In the original analysis of MEM [49], the mask involves a computation } P(K\|N) \text{ for nonce } N. \text{ This not only complicates the values that have to be revealed; it also results in a larger view and hence a higher collision probability among tuples in the view.} \]
reason for this is that such a view never occurs in the real world, making the ratio in Lemma C.1 only valid for $\varepsilon = 1$. Therefore, formally, $\nu$ is called “bad” if one of the following conditions is satisfied, where we recall that $\nu_k = \{(K, L)\}$ is a singleton:

- $\text{bad}_{c,c}$: for some distinct $(\bar{a}, M, C), (\bar{a}', M', C') \in \nu_c$:
  $\text{mask}_K^a(L) \oplus \text{mask}_K^{a'}(L) \in \{M \oplus M', C \oplus C'\}$,

- $\text{bad}_{c,p}$: for some $(\bar{a}, M, C) \in \nu_c$ and $(X, Y) \in \nu_p$:
  $\text{mask}_K^a(L) \in \{M \oplus X, C \oplus Y\}$,

- $\text{bad}_{c,k}$: for some $(\bar{a}, M, C) \in \nu_c$:
  $\text{mask}_K^a(L) \in \{M \oplus K \parallel 0^{n-k}, C \oplus L\}$,

- $\text{bad}_{p,k}$: for some $(X, Y) \in \nu_p$:
  $X = K \parallel 0^{n-k}$ or $Y = L$.

We write $\text{bad} = \text{bad}_{c,c} \lor \text{bad}_{c,p} \lor \text{bad}_{c,k} \lor \text{bad}_{p,k}$.

### C.3 Probability of Bad View in Ideal World

Our goal is to bound $\Pr(D_P \in \nu_{\text{bad}})$, the probability of a bad view in the ideal world $P = (\tilde{\pi}, P^\pm)$. For brevity, denote by $D_P \propto \text{bad}$ the event that $D_P$ satisfies bad. By the union bound,

$$
\Pr(D_P \propto \text{bad}) = \Pr(D_P \propto \text{bad}_{c,c} \lor \text{bad}_{c,p} \lor \text{bad}_{c,k} \lor \text{bad}_{p,k}) \\
\leq \Pr(D_P \propto \text{bad}_{c,c}) + \Pr(D_P \propto \text{bad}_{c,p}) \\
+ \Pr(D_P \propto \text{bad}_{c,k}) + \Pr(D_P \propto \text{bad}_{p,k}).
$$

(14)

We will analyze the four probabilities separately, thereby noticing that (i) $K \leftarrow \{0, 1\}^k$ and $L \leftarrow \{0, 1\}^n$ are random variables, and (ii) as the adversary only makes forward construction queries, each tuple $(\bar{a}, M, C) \in \nu_c$ satisfies that $C$ is randomly drawn from a set of size at least $2^n - q$.

**Event** $\text{bad}_{c,c}$. For bad$_{c,c}$, let $(\bar{a}, M, C), (\bar{a}', M', C') \in \nu_c$ be any two distinct tuples. If $\bar{a} = \bar{a}'$, then necessarily $M \neq M'$ and $C \neq C'$, and bad$_{c,c}$ holds with probability 0. Otherwise, if $\bar{a} \neq \bar{a}'$, we can deduce from $2^{-\alpha}$-properness of $T$, namely property 2 of Definition B.1, that event bad$_{c,c}$ holds with probability at most $2/2^{\alpha}$. Thus, summing over all $\binom{q}{2}$ possible choices of queries,

$$
\Pr(D_P \propto \text{bad}_{c,c}) \leq \frac{q(q-1)}{2^{\alpha}}.
$$

**Event** $\text{bad}_{c,p}$. For bad$_{c,p}$, let $(\bar{a}, M, C) \in \nu_c$ and $(X, Y) \in \nu_p$ be any two tuples. We can deduce from $2^{-\alpha}$-properness of $T$, namely property 1 of Definition B.1, that event bad$_{c,p}$ holds with probability at most $2/2^{\alpha}$. Thus, summing
over all \( qp \) possible choices of queries,

\[
\Pr(D_P \propto \text{bad}_{c,p}) \leq \frac{2qp}{2^\alpha}.
\]

Event \( \text{bad}_{c,k} \). For \( \text{bad}_{c,k} \), let \((\bar{a}, M, C) \in \nu_c\) be any tuple. We consider the two equations of \( \text{bad}_{c,k} \) separately. For the first equation,

\[
\text{mask}_{K}^\bar{a}(L) = M \oplus K||0^{n-k},
\]

we will use that \( L \xleftarrow{} \{0,1\}^n \) is a randomly generated value independent of \( K \). We can deduce from \( 2^{-\alpha} \)-properness of \( T \), namely property 1 of Definition B.1, that this equation holds with probability at most \( 1/2^\alpha \).

For the second equation,

\[
\text{mask}_{K}^\bar{a}(L) = C \oplus L,
\]

we will use that all construction queries are made in forward direction, and that \( C \) is randomly drawn from a set of size at least \( 2^n - q \) elements. Above equation thus holds with probability at most \( 1/(2^n - q) \).

Thus, summing over all \( q \) possible choices of queries,

\[
\Pr(D_P \propto \text{bad}_{c,k}) \leq \frac{q}{2^n} + \frac{q}{2^n - q}.
\]

Event \( \text{bad}_{p,k} \). For \( \text{bad}_{p,k} \), let \((X, Y) \in \nu_p\) be any tuple. As \( K \xleftarrow{} \{0,1\}^k \) and \( L \xleftarrow{} \{0,1\}^n \), the tuple sets \( \text{bad}_{p,k} \) with probability at most \( 1/2^k + 1/2^n \). Thus, summing over all \( p \) possible choices of queries,

\[
\Pr(D_P \propto \text{bad}_{p,k}) \leq \frac{p}{2^k} + \frac{p}{2^n}.
\]

Conclusion. Concluding, we obtain for (14):

\[
\Pr(D_P \propto \text{bad}) \leq \frac{q^2 + 2qp}{2^n} + \frac{2q + p}{2^n} + \frac{p}{2^k}.
\]  

(15)

using that \( 2^n - q \geq 2^{n-1} \).

C.4 Probability Ratio for Good Views

Consider any good view \( \nu \in \mathcal{V}_{\text{good}} \). We will prove the inequality \( \Pr(D_\emptyset = \nu) \geq \Pr(D_P = \nu) \). The proof is a direct simplification of that of Granger et al. [49], noting that in our case, \( \nu_k \) consists of just one element. The proof is included for completeness.
Real World. In the real world $\mathcal{O} = (\tilde{E}_K, P^\pm)$, goodness of the view means that $\nu = (\nu_c, \nu_p, \nu_k)$ defines exactly $q + p + 1$ input-output pairs for $P$ and $\nu_k$ consists of a random value $K \xleftarrow{\$} \{0, 1\}^k$, and there are no two of them that collide on the input or output. Therefore, we obtain:

$$\Pr(D_\mathcal{O} = \nu) = \Pr \left( K' \xleftarrow{\$} \{0, 1\}^k : K' = K \right) \cdot \Pr \left( P \xleftarrow{\$} \text{perm}(n) : \tilde{E}_K^P \vdash \nu_c \land P \vdash \nu_p \land P \vdash \nu_k \right)$$

$$= \frac{1}{2^k} \cdot \frac{(2^n - (q + p + 1))!}{2^n!} .$$ (16)

Ideal World. In the ideal world $\mathcal{P} = (\tilde{\pi}, P^\pm)$, the view $\nu = (\nu_c, \nu_p, \nu_k)$ consists of three lists of independent tuples: $\nu_c$ defines exactly $q$ input-output pairs for $\tilde{\pi}$, $\nu_p$ defines exactly $p$ input-output pairs for $P$, and $\nu_k$ consists of two random values $(K, L) \xleftarrow{\$} \{0, 1\}^k \times \{0, 1\}^n$. For counting, it is convenient to group the tuples in $\nu_c$ depending on the tweak value $\bar{a}$. For $T \in \mathcal{T}$, define

$$q_T = \left| \{(\bar{a}, M, C) \in \nu_c \mid \bar{a} = T\} \right| ,$$

where $\sum_{T \in \mathcal{T}} q_T = q$. We obtain:

$$\Pr(D_\mathcal{P} = \nu) = \Pr \left( (K', L') \xleftarrow{\$} \{0, 1\}^k \times \{0, 1\}^n : (K', L') = (K, L) \right) \cdot \Pr \left( \tilde{\pi} \xleftarrow{\$} \text{perm}(\mathcal{T}, n) : \tilde{\pi} \vdash \nu_c \right) \cdot \Pr \left( P \xleftarrow{\$} \text{perm}(n) : P \vdash \nu_p \right)$$

$$= \frac{1}{2^{k+n}} \cdot \prod_{T \in \mathcal{T}} \frac{(2^n - q_T)!}{2^n!} \cdot \frac{(2^n - p)!}{2^n!}$$

$$= \frac{1}{2^k} \cdot \frac{(2^n - 1)!}{2^n!} \cdot \prod_{T \in \mathcal{T}} \frac{(2^n - q_T)!}{2^n!} \cdot \frac{(2^n - p)!}{2^n!}$$

$$\leq \frac{1}{2^k} \cdot \frac{(2^n - (q + p + 1))!}{2^n!} ,$$ (17)

using that for any $\sigma, \tau \leq 2^n$ we have $\frac{(2^n - \sigma)!}{2^n!} \cdot \frac{(2^n - \tau)!}{2^n!} \leq \frac{(2^n - (\sigma + \tau))!}{2^n!}$.

Conclusion. Combining (16) and (17), we obtain that for any good view $\nu \in \mathcal{V}_{\text{good}}$:

$$\frac{\Pr(D_\mathcal{O} = \nu)}{\Pr(D_\mathcal{P} = \nu)} \geq 1 .$$ (18)
C.5 Conclusion

By the H-coefficient technique (Lemma C.1), we directly obtain from (15) and (18):

\[
\text{Adv}^\text{trp}_E(A) \leq 0 + \frac{q^2 + 2qp}{2^n} + \frac{2q + p}{2^n} + \frac{p}{2^n}.
\]

D Proof of Theorem B.3 (on Elephant)

Let \( k \xleftarrow{} \{0, 1\}^k \), \( P \xleftarrow{} \text{perm}(n) \), and \( \text{rand} \) be a function that for each input \((N, A, M)\) returns a random string of size \(|M| + t\) bits. Consider a deterministic computationally unbounded adversary \( A \) that tries to distinguish \( O := (\text{enc}^P_K, \text{dec}^P_K, P^\pm) \) from \( P := (\text{rand}, \bot, P^\pm) \):

\[
\text{Adv}_{\text{Elephant}}^a(A) = \Delta_{\text{A}}(\text{enc}^P_K, \text{dec}^P_K, P^\pm; \text{rand}, \bot, P^\pm).
\]  (19)

As a first step, we will describe an alternative authenticated encryption scheme \( \text{AE}' \) based on a tweakable permutation \( \pi' \xleftarrow{} \text{perm}(T, n) \), where \( T \) is \( 2^{-\alpha} \)-proper with respect to LFSRs \( (\varphi_1, \varphi_2) \). Its encryption function \( \text{enc} \) and decryption function \( \text{dec} \) are given in Algorithms 3 and 4, respectively. Unlike the original functions \( \text{enc} \) and \( \text{dec} \) of Algorithms 1 and 2, the functions \( \text{enc} \) and \( \text{dec} \) are not explicitly keyed, but are instead implicitly keyed by the use of random secret tweakable permutation \( \pi' \).

---

**Algorithm 3** encryption

**Input:** \((N, A, M)\)

**Output:** \((C, T)\)

1. \( M_1 \ldots M_{\ell_M} \xleftarrow{} M \)
2. **for** \( i = 1, \ldots, \ell_M \) **do**
3. \( C_i \leftarrow M_i \oplus \pi'((i-1, 0), N\|0^n-m) \)
4. \( C \leftarrow [C_1 \ldots C_{\ell_M}]_{|M|} \)
5. \( T = 0 \)
6. \( A_1 \ldots A_{\ell_A} \xleftarrow{} N\|A\|1 \)
7. \( C_1 \ldots C_{\ell_C} \xleftarrow{} C\|1 \)
8. **for** \( i = 1, \ldots, \ell_A \) **do**
9. \( T \leftarrow T \oplus \pi'((i-1, 2), A_i) \)
10. **for** \( i = 1, \ldots, \ell_C \) **do**
11. \( T \leftarrow T \oplus \pi'((i-1, 1), C_i) \)
12. **return** \((C, [T]_{\ell_C})\)

---

**Algorithm 4** decryption

**Input:** \((N, A, C, T)\)

**Output:** \( M \) or \( \bot \)

1. \( C_1 \ldots C_{\ell_M} \xleftarrow{} C \)
2. **for** \( i = 1, \ldots, \ell_M \) **do**
3. \( M_i \leftarrow C_i \oplus \pi'((i-1, 0), N\|0^n-m) \)
4. \( M \leftarrow [M_1 \ldots M_{\ell_M}]_{|C|} \)
5. \( T = 0 \)
6. \( A_1 \ldots A_{\ell_A} \xleftarrow{} N\|A\|1 \)
7. \( C_1 \ldots C_{\ell_C} \xleftarrow{} C\|1 \)
8. **for** \( i = 1, \ldots, \ell_A \) **do**
9. \( T \leftarrow T \oplus \pi'((i-1, 2), A_i) \)
10. **for** \( i = 1, \ldots, \ell_C \) **do**
11. \( T \leftarrow T \oplus \pi'((i-1, 1), C_i) \)
12. **return** \([T]_{\ell_T} = T \land M : \bot\)
By a simple hybrid argument, we obtain for the distance of (19):

\[
\begin{align*}
(19) & \leq \Delta_A \left( \text{enc}^P_{K}, \text{dec}^P_{K}, P^\pm \ ; \ \text{enc}^{SIM}_{K}, \text{dec}^{SIM}_{K}, P^\pm \right) \\
& \quad + \Delta_A \left( \text{enc}^{SIM}_{K}, \text{dec}^{SIM}_{K}, P^\pm \ ; \ \text{enc}^{\pi}, \text{dec}^{\pi}, P^\pm \right) \\
& \quad + \Delta_A \left( \text{enc}^{\pi}, \text{dec}^{\pi}, P^\pm \ ; \ \text{rand}, \bot, P^\pm \right).
\end{align*}
\]

The first distance of (20) equals 0 by design of AE'. The second distance of (20) is at most \(\Delta_A' \left( \text{Sim}^P_{K}, P^\pm \ ; \ \tilde{\pi}, P^\pm \right) = \text{Adv}^{\text{tprp}}_{\text{Sim}}(A')\), where \(A'\) is an adversary that makes 2\(\sigma\) construction queries and \(p\) primitive queries in order to simulate \(A'\)'s oracles. For the third distance of (20), access to \(P\) does not help the adversary, and the oracle can be dropped. We obtain from (20):

\[
(19) \leq \text{Adv}^{\text{tprp}}_{\text{Sim}}(A') + \Delta_A \left( \text{enc}^{\tilde{\pi}}, \text{dec}^{\tilde{\pi}} \ ; \ \text{rand}, \bot \right)
\]

\[
\leq \text{Adv}^{\text{tprp}}_{\text{Sim}}(A') + \Delta_A \left( \text{enc}^{\tilde{\pi}}, \text{dec}^{\tilde{\pi}} \ ; \ \text{enc}^{\tilde{\pi}}, \bot \right)
\]

\[
+ \Delta_A \left( \text{enc}^{\tilde{\pi}}, \bot \ ; \ \text{rand}, \bot \right).
\]

In order to upper bound the two remaining distances of (21), we will introduce the following two functions. First, define \(h : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^t\) as

\[
h(X, Y) = \left[ \bigoplus_{i=1}^{\ell_X} \tilde{\pi}((i, 2), X_i) \right] \oplus \left[ \bigoplus_{i=1}^{\ell_Y} \tilde{\pi}((i - 1, 1), Y_i) \right],
\]

where \(X_1 \ldots X_{\ell_X} \leftarrow X \parallel 1\) and \(Y_1 \ldots Y_{\ell_Y} \leftarrow Y \parallel 1\). For permutation \(\pi \leftarrow \text{perm}(n)\), define the MAC function

\[
\text{mac}^{\pi,h}(Z, X, Y) = |\pi(Z)|_t \oplus h(X, Y),
\]

and let \(\text{vfy}^{\pi,h}\) be the corresponding verification function. We will use a result of Bernstein [10] on Wegman-Carter-Shoup [82, 92] authenticators, translated to our setting.

**Lemma D.1.** Let \(\pi \leftarrow \text{perm}(n)\), and \(h : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^t\) be 2\(^{-\alpha}\) XOR-uniform and independent of \(\pi\). Consider the message authentication code \(\text{mac}^{\pi,h}\) and its corresponding verification function \(\text{vfy}^{\pi,h}\) of (22). For any adversary \(A\) making at most \(q_e \leq 2^{n-1}\) MAC queries and \(q_d\) forgery attempts,

\[
\Delta_A \left( \text{mac}^{\pi,h}, \text{vfy}^{\pi,h} \ ; \ \text{mac}^{\pi,h}, \bot \right) \leq q_d \cdot 2^{-\alpha} \cdot e^{(q_d+1)q_e/2^n}.
\]

The proof will be given in Section D.1.

One can reduce a distinguishing attack for the first distance of (21) to a forgery on \(\text{mac}^{\pi,h}\) with \(\pi := \tilde{\pi}((0, 2), \cdot)\). Hence, using Lemma D.1 along with
the fact that $h$ is $2^{n-t}(2^n - 1)^{-1}$-XOR-uniform, we obtain
\[
\Delta_A \left( \text{enc}^\pi, \text{dec}_K^\pi; \text{rand} \right) \leq \Delta_{A'} \left( \text{mac}^\pi, h^\pi; \text{mac}^\pi, h \right) \\
\leq 2^{n-t} q_d \epsilon \frac{(q_e + 1) q_e}{2^n},
\] (23)
where $A'$ has the same resources as $A$.

For the second distance of (21), we remark that every query is made for a unique nonce, and in more detail:

- The $i$-th block of ciphertext equals $\tilde{\pi}(((i - 1, 0), N) \oplus M_i$, where $M_i$ is the $i$-th block of plaintext;
- The tag equals $\lfloor \tilde{\pi}((0, 2) \parallel A'), C \rfloor \oplus h(A''), C)$, where $A'$ equals the first $n - m$ bits of padded associated data and $A''$ equals the rest, and where $h$ never evaluates $\tilde{\pi}$ for tweak $(\cdot, 0)$ or $(0, 2)$.

The tweakable permutation $\tilde{\pi}$ is independent for different tweaks, but two different inputs for the same tweak never collide. Therefore, this second distance of (21) satisfies
\[
\Delta_A \left( \text{enc}^\pi, \bot; \text{rand}, \bot \right) \leq \ell \left( \frac{q_e}{2} \right) / 2^n.
\] (24)

We thus obtain from (21), (23), and (24):
\[
(19) \leq \text{Adv}_{\text{SIM}}^{\text{prp}}(A') + \frac{2^{n-t} q_d \epsilon (q_e + 1) q_e}{2^n} \sqrt{q_e} / 2^n + \ell \left( \frac{q_e}{2} \right) / 2^n,
\]
and this completes the proof of Theorem B.3.

**D.1 Proof of Lemma D.1 (On $\text{mac}^{\pi,h}$)**

We write $f_t(N) = [\pi(N)]_t$ for brevity. Define the maximum $k$-interpolation probability of $f_t$ as the maximum of
\[
\text{Pr} (f_t(x_1) = y_1, \ldots, f_t(x_k) = y_k)
\]
(25)
taken over any distinct $x_1, \ldots, x_k \in \{0, 1\}^n$ and any $y_1, \ldots, y_k \in \{0, 1\}^t$.

Bernstein [10, Theorem 5.1] states that if $f_t$ has maximum $q_e$-interpolation probability at most $\delta/2^{q_e}$ and maximum $(q_e + 1)$-interpolation probability at most $2^{-\alpha/2^{q_e}}$, then the message authentication code $\text{mac}^{\pi,h}$ of (22) satisfies
\[
\Delta_A \left( \text{mac}^{\pi,h}, \text{vfy}^{\pi,h}; \text{mac}^{\pi,h}, \bot \right) \leq q_d \cdot 2^{-\alpha} \cdot \delta.
\]
\(^{4}\)

\(^{4}\) A sharp eye may note that the size of the range of $f_t$ is at most the size of its domain, therewith violating the condition “$\#N \leq \#G$” in [10, Theorem 5.1]. However, close inspection of the proof reveals that the condition is not used.
The maximum $k$-interpolation probability of $f_t$, for $k \leq q_e + 1 \leq 2^{n-1} + 1$, satisfies:

$$
\Pr (f_t(x_1) = y_1, \ldots, f_t(x_k) = y_k) \leq \prod_{i=1}^{k} \frac{2^{n-t}}{2^n - (i-1)}
$$

$$
= 2^{-tk} \cdot \prod_{i=1}^{k} \left( 1 + \frac{i - 1}{2^n - (i - 1)} \right)
$$

$$
\leq 2^{-tk} \cdot \prod_{i=1}^{k} \left( 1 + \frac{2(i - 1)}{2^n} \right)
$$

$$
\leq 2^{-tk} \cdot e^{\sum_{i=1}^{k} \frac{i - 1}{2^n}}
$$

$$
= e^{k(k-1)/2^n} / 2^{tk},
$$

where we used that $k - 1 \leq 2^{n-1}$. As $2^{-\alpha} \geq 2^{-t}$, the bound satisfies the constraints put forward by Bernstein for $\delta = e^{(q_e + 1)q_e/2^n}$.

We remark that for $t = n$, i.e., for $f_n$ an injective function, Bernstein computed the same maximum $k$-interpolation probability in [10, Theorem 4.2] and derived a similar bound on the security of $\text{mac}^{s,h}$ in [10, Theorem 5.3].