Improving the Sphinx Mix Network

Abstract. Secure mix networks consider the presence of multiple nodes that relay encrypted messages from one node to another in such a way that anonymous communication can be achieved. We consider the Sphinx mix formatting protocol by Danezis and Goldberg (IEEE Security and Privacy 2009), and analyze its use of symmetric-key cryptographic primitives. We scrutinize the reliance on multiple distinct primitives, as well as the use of the ancient LIONESS cipher, and suggest various paths towards improving the security and efficiency of the protocol.

Keywords: Sphinx; mix network; LIONESS; authenticated encryption; sponge

1 Introduction

With the large growth of Internet services, modern users rely more on digital and ubiquitous communications. In this digital domain, privacy needs to be protected against the very nature of the communications, which tend to be easily traceable and produce massive amounts of metadata. Such a task requires some cryptographic methods. Hiding the metadata of communications is hard, but there are some systems, called mix networks, that provide such capabilities. One recent example is the Sphinx [22] mix network, that is in wide use in privacy protecting applications. The Sphinx mix network format provides security against powerful adversaries and also good communications possibilities such as replies, which are not easily available in other mix network formats.

The Sphinx protocol (see Section 2) uses internally a plurality of symmetric-key primitives. At first, there is the SHA-2 hash function used for hashing. The function additionally includes a HMAC mode [9] to support message authentication. Then, Sphinx relies on the LIONESS blockcipher [1], an encryption functionality that is made out of the SEAL stream cipher [52] and a keyed version of the SHA-1 hash function, and evaluates these functions on the message via a Feistel structure. In addition, Sphinx uses a pseudorandom generator to generate entropy for the key. All of these symmetric-key primitives are used in a strongly intertwined manner.

The LIONESS blockcipher is proven to be secure, under the assumption that the underlying primitives SEAL and SHA-1 are sufficiently secure [1], therewith making it particularly useful for Sphinx because of its goal to achieve provable security. However, LIONESS dates back to 1996, and has been outpaced by reality. Attacks on SEAL [27] and SHA-1 [24,36,54,55,56], the most recent result being a free-start collision attack on the full SHA-1, shine a negative light on the security of the LIONESS mode.

1.1 Our Contribution

Most importantly, we suggest to replace the encryption and authentication functionalities by one *authenticated encryption* functionality. This optimization allows for improved security as, unlike in the original Sphinx, the payload now gets authenticated without any efficiency cost. Various existing solutions to authenticated encryption exist (see Section 1.2 for a discussion on relevant works in this direction), but not all schemes are suitable. In this work, we suggest two approaches for authenticated encryption that are particularly suited for Sphinx. As a bonus, both approaches allow for an elimination of various symmetric-key primitive calls, or more formally, for merging various calls into one. This, consequently, contributes to the simplicity and efficiency of the design.

The first proposal in Section 4 is based on the keyed version of the sponge, which has found recent interest in the design of SHA-3 [16,13]. In more detail, the authenticated encryption scheme can be seen as an adaption of the full-state SpongeWrap [15,45]. It internally uses a large unkeyed permutation, the state of which is separated into a capacity and a rate. The capacity determines the security bound, and the rate determines the speed at which data is processed. By using a large permutation, one can make a proper balance between the capacity and the rate, and therewith achieve a high level of security. However, the scheme inherently requires nonce respecting behavior: every new evaluation of the authenticated encryption scheme should be initialized with a unique nonce.

The second proposal in Section 5 is blockcipher based, and is resistant to nonce reuse. The design of our second proposal is somewhat orthogonal to that of conventional nonce reuse resistant authenticated encryption schemes such as [5,46,23,31]: while other designs consist of a mode built on top of AES, we follow a "tweakable tweakable blockcipher" approach. In other words, we give a powerful construction of a tweakable blockcipher mode on top of a tweakable blockcipher, in such a way that the scheme allows for sufficiently large message and associated data, while still being quite simple and nonce reuse resistant. For a specific instantiation of the construction, we suggest Threefish, a tweakable blockcipher with 1024-bit state by Ferguson et al. used for the Skein hash function family [26]. (Alternative solutions are the use of (i) a large permutation (such as the Keyak [17] or Prøst [37] permutation) in a tweakable Even-Mansour mode [21] and (ii) the TWEAKEY tweakable blockcipher by Jean et al. [32,33,34]. However, the Keyak permutation is known to have an expensive inverse, and the other solutions have a state size which is too small for our purposes.) Threefish has withstood a wide variety of cryptanalysis [7,20,26,38,39,40,42,57].

In Section 6, we apply our schemes of Section 4 and Section 5 to the Sphinx format. The new Sphinx format of Figure 4 improves over the earlier one (i) in terms of simplicity, (ii) in terms of efficiency, and (iii) in terms of security.

1.2 Related Work on Authenticated Encryption

Authenticated encryption (AE) enjoys a long and steady line of research, and the ongoing CAESAR competition [19] for the design of new AE schemes has induced

renewed attention to the field. The classical approach to design AE schemes is to build the generic mode of operation on top of a blockcipher (usually AES) in order to process data blocks iteratively [5,12,23,31,41,46,50,51]. A more novel approach is to design AE based on permutations. The most well-known approach is SpongeWrap by Bertoni et al. [15] which got recently generalized by Jovanovic et al. [35] and Mennink et al. [45], and various CAESAR submissions follow this idea [2,8,17,29,25,47,53]. Different permutation based approaches include APE [3] and PAEQ [18].

The sponge based proposal in this work follows the literature. Regarding our blockcipher based approach, we have deviated from the state of the art. The reason is that conventional modes often entail overhead, e.g., in the form of padding, and the security level is in the end dominated by what the underlying primitive offers. For blockcipher based modes, using AES internally delivers at most 128-bit security, and often there exist already distinguishability attacks in complexity of about 2^{64} (cf. Bellare et al. [10]). Note that for messages of, say, 1024 bits, a classical AES based mode still requires at least 8 AES evaluations. We remark that also AEZ [31], or more detailed the latest version v4 in the CAESAR competition [30], is also inherently a mode based on 4 and 10 rounds of AES, and has 64-bit security as well. Recent cryptanalysis on AEZ [28,44] has moreover shined a negative light on its security.

2   Sphinx Mix Format

Mix networks rely on mix message formats that provide efficiency and security properties. Sphinx [22] is the most compact cryptographic mix message format, which is provably secure and efficient. To deliver such properties, Sphinx relies on the *Sphinx blinding logic* technique for generating a session key consisting of nested MAC computations over the public pseudonyms of each predecessor mix. The private key associated to the public key (i.e., pseudonym) is only known by the user, while the session key is used for the encryption of the payload message. A high-level depiction of Sphinx is given in Figure 1.

Internally, Sphinx uses a plurality of cryptographic primitives. First, there are five hash functions, which are used to cryptographically hash group elements to key bit strings. Then, it uses a pseudorandom generator PRG and a MAC function for the computation of the nested MAC. Finally, it uses an encryption scheme ENC that encrypts the payload at every mix.

The hash functions are instantiated using appropriately truncated SHA256 hash functions, and SHA256-HMAC-128 is used as the MAC function. For the encryption, Sphinx relies on the LIONESS blockcipher by Anderson and Biham [1]. This blockcipher is made out of the SEAL stream cipher and a keyed version of the SHA-1 hash function, and evaluates these functions on the message via a Feistel structure. In more detail, denote the stream cipher by S_k and the keyed hash function by H_k. Consider a LIONESS key $k = (k_1, k_2, k_3, k_4)$, where k_1, k_3 will be used to key the stream cipher and k_2, k_4 to key the hash function. To encrypt a message m, LIONESS first splits it into two blocks
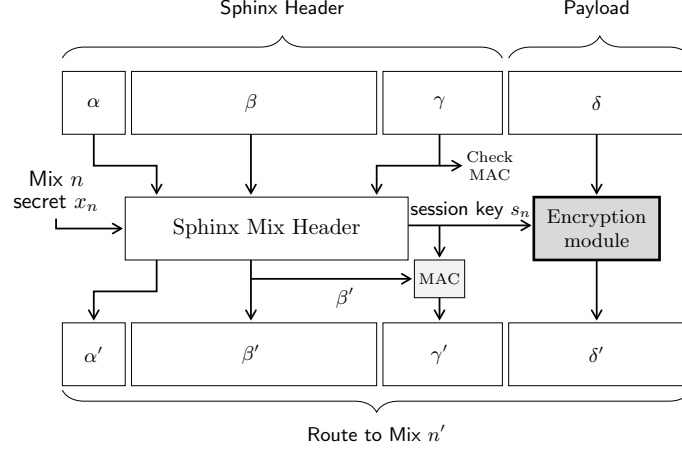
Fig. 1. High-level description of Sphinx [22] to process a Sphinx message $((\alpha, \beta, \gamma), \delta)$ into $((\alpha', \beta', \gamma'), \delta')$ at Mix n with the secret x_n. Key derivation $s_n = \mathsf{keyderive}(\alpha, x_n)$ and the blinding $(\alpha', \beta') = \mathsf{blinding}(\alpha, \beta, \gamma, s_n)$ are omitted from the picture.

$m_l \| m_r \leftarrow m$. These two blocks are then transformed using a four-round Feistel structure: $m_r \leftarrow m_r \oplus S_{k_1}(m_l)$, $m_l \leftarrow m_l \oplus H_{k_2}(m_r)$, $m_r \leftarrow m_r \oplus S_{k_3}(m_l)$, $m_l \leftarrow m_l \oplus H_{k_4}(m_r)$. The updated $m_l \| m_r$ constitutes the ciphertext c.

Due to the security parameter choices, the Sphinx construction needs an encryption scheme with a state of at least 1408 bits plus the message length. Based on this, LIONESS appears to be a good option as it has the potential to have a large state and thus act as the permutation required by the Sphinx system. In addition, LIONESS enjoys a security proof if the underlying hash function SHA-1 and stream cipher SEAL are secure [1]. However, the security of LIONESS is undermined by the results mentioned in the introduction.

Besides the doubtful use of LIONESS in the first place, it is noteworthy that Sphinx uses different symmetric-key primitives for various purposes: i.e., SHA-1 is used in LIONESS and SHA-2 for hashing and MACing. These functions are often intertwined, and particularly, three of the cryptographic hash functions are used to transform a secret non-identity group element s to secret keys to the PRG, MAC, and ENC. In other words, denoting these three hash functions as H_{PRG}, H_{MAC}, and H_{ENC}, Sphinx calls the PRG, MAC, and ENC functionalities as follows:

$$\mathrm{PRG}(H_{\mathrm{PRG}}(s))\,,$$
$$\mathrm{MAC}(H_{\mathrm{MAC}}(s), m)\,,$$
$$\mathrm{ENC}(H_{\mathrm{ENC}}(s), m)\,,$$

where s is the secret group element, the secret session key, and m denotes the data to be MACed or ENCed. The synergy between MAC and H_{MAC} is striking, given the designers' choice to instantiate those with SHA256-HMAC-128, and SHA256,

respectively. For the case of encryption, the situation is not much clearer, given that LIONESS uses SHA-1 while H_{ENC} is instantiated with SHA256.

Finally, from Figure 1, it becomes apparent that γ is a MAC of β (using session key s), and δ is the encryption of the payload (under session key s). It turns out that, by merging these two functionalities into one *authenticated encryption* scheme that authenticates β and δ and that encrypts δ, one obtains the following improvements:

- Authentication of β and encryption of δ still persists, but authentication of δ is *for free*;
- The session key needs to be processed *only once*;
- There is no need to implement two distinct algorithms.

As such, our main goal in this work will be to introduce an authenticated encryption scheme that suits Sphinx, which will be done in Sections 3-5. The potential employment of the new schemes in Sphinx will be considered in Section 6 in such a way that the remaining above-mentioned issues (such as the redundant usage of cryptographic primitives) are resolved on the fly.

3 Authenticated Encryption

For $n \in \mathbb{N}$, $\{0,1\}^n$ is the set of n-bit strings, and $\{0,1\}^{\leq n} = \bigcup_{i=0}^{n}\{0,1\}^i$. For two bit strings M, N, their concatenation is denoted by $M\|N$ and $M \oplus N$ denotes their bitwise XOR. Furthermore, if $M \in \{0,1\}^{\leq n-1}$, then $\mathsf{pad}_n(M) = M\|10^{n-1-|M|}$. For a string $N \in \{0,1\}^n$, we define by $\mathsf{unpad}_n(N)$ the unique string $M \in \{0,1\}^{\leq n-1}$ such that $\mathsf{pad}_n(M) = N$. For $m \leq n$ and $N \in \{0,1\}^n$, we denote by $\lceil N \rceil_m$ the leftmost m bits and by $\lfloor N \rfloor_{n-m}$ the rightmost $n-m$ bits of N, in such a way that $N = \lceil N \rceil_m \| \lfloor N \rfloor_{n-m}$.

3.1 Definition of Authenticated Encryption

Let $\mu, \nu, \alpha, \tau, \sigma \in \mathbb{N}$ be size values that satisfy $\mu \leq \nu$. Here, μ denotes the size of the message, ν the size of the ciphertext, τ the size of the associated data, and σ the size of the nonce. The value α determines the size of the authentication tag. If no authentication is needed, we have $\alpha = 0$.

An authenticated encryption scheme AE is composed of three algorithms: `KeyGen`, `Enc`, and `Dec`. `KeyGen` is a randomized algorithm that gets as input $\kappa \in \mathbb{N}$ and outputs a random key $key \leftarrow \{0,1\}^\kappa$. The `Enc` and `Dec` algorithms are defined as follows:

$$\texttt{Enc}: \{0,1\}^\kappa \times \{0,1\}^{\leq \mu} \times \{0,1\}^{\leq \tau} \times \{0,1\}^\sigma \rightarrow \{0,1\}^{\leq \nu} \times \{0,1\}^\alpha,$$
$$(key, msg, meta, nonce) \mapsto (ctxt, auth),$$

$$\texttt{Dec}: \{0,1\}^\kappa \times \{0,1\}^{\leq \nu} \times \{0,1\}^\alpha \times \{0,1\}^{\leq \tau} \times \{0,1\}^\sigma \rightarrow \{0,1\}^{\leq \mu} \cup \{\bot\},$$
$$(key, ctxt, auth, meta, nonce) \mapsto msg/\bot.$$

Dec outputs the unique msg satisfying $\texttt{Enc}(key, msg, meta, nonce) = (ctxt, auth)$, or it returns $\bot$ if no such message exists. Enc also outputs $meta$ and $nonce$. We allow for a small amount of ciphertext expansion (from μ to ν bits), as long as the encrypted ciphertext $(ctxt, auth, meta, nonce)$ is of size at most $\lambda_{\max}$.

3.2   Threat Model

We consider an adversary $\mathcal{A}$ to be any entity attempting to passively access the shared information by monitoring the communication channel, with no incentive to tamper with the content. However, $\mathcal{A}$ is allowed to generate encryptions under a secret and unknown key. In this case, $\mathcal{A}$ should not learn the encrypted content, beyond that revealed in the associated data.

   More technically, adversary $\mathcal{A}$ has query access to Enc under a secret key key, and it tries to find irregularities among the queries, i.e., some relation that is not likely to hold for a random function. For a function F, let $\mathsf{Func}(F)$ be the set of all functions f with the same interface as F. The advantage of an adversary $\mathcal{A}$ in breaking the secrecy of an authenticated encryption scheme AE is defined as:

$$\mathbf{Adv}_{\mathrm{AE}}^{\mathrm{cpa}}(\mathcal{A}) = \left| \Pr\left(key \xleftarrow{\$} \mathtt{KeyGen}(\kappa) \; : \; \mathcal{A}^{\mathtt{Enc}_{key}} = 1 \right) - \right. $$
$$\left. \Pr\left(\$ \xleftarrow{\$} \mathsf{Func}(\mathtt{Enc}_{key}) \; : \; \mathcal{A}^{\$} = 1 \right) \right| .$$

We denote by $\mathbf{Adv}_{\mathrm{AE}}^{\mathrm{cpa}}(Q, T)$ the maximum advantage over all adversaries that make at most Q encryption queries and operate in time T. Depending on the scheme, the adversary $\mathcal{A}$ may be limited to being nonce respecting, so that every query must be made under a different nonce. We will always explicitly note if this is the case.

   For the authenticity of AE, we consider $\mathcal{A}$ to have access to the encryption functionality Enc under a secret key key, and say that $\mathcal{A}$ *forges* an authentication tag if it manages to output a tuple $(ctxt, auth, meta, nonce) \in \{0,1\}^{\leq \nu} \times \{0,1\}^{\alpha} \times \{0,1\}^{\leq \tau} \times \{0,1\}^{\sigma}$ such that $\texttt{Dec}(key, ctxt, auth, meta, nonce) = msg \neq \bot$ and $(msg, meta, nonce)$ was never queried to Enc before. The forgery attempt may be made under a nonce $nonce$ that has appeared before. The advantage of $\mathcal{A}$ in breaking the authenticity of authenticated encryption scheme AE is defined as:

$$\mathbf{Adv}_{\mathrm{AE}}^{\mathrm{auth}}(\mathcal{A}) = \Pr\left(key \xleftarrow{\$} \mathtt{KeyGen}(\kappa) \; : \; \mathcal{A}^{\mathtt{Enc}_{key}} \text{ forges} \right) .$$

We denote by $\mathbf{Adv}_{\mathrm{AE}}^{\mathrm{auth}}(Q, R, T)$ the maximum advantage over all adversaries that make at most Q encryption queries, R forgery attempts, and operate in time T.

4   Solution 1: Sponge

The Sponge functions were originally introduced by Bertoni et al. [16] for cryptographic hashing, but can also be used in a broad spectrum of keyed applications,
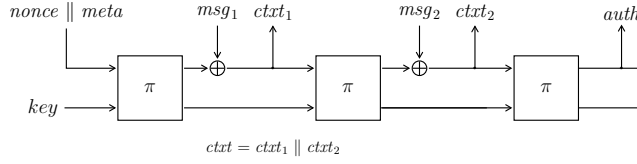
Fig. 2. AE based on a Sponge. Padding of data is excluded from the figure.

Algorithm 1 $\text{Enc}^{\pi,\ell,n}$

Input: $(key, msg, meta, nonce) \in \{0,1\}^{\kappa} \times \{0,1\}^{\leq \mu} \times \{0,1\}^{\leq \tau} \times \{0,1\}^{\sigma}$
Output: $(ctxt, auth) \in \{0,1\}^{\leq \mu} \times \{0,1\}^{\alpha}$
1: $\ell' \leftarrow \lceil (|msg| + 1)/n \rceil$
2: $msg_1 \| \cdots \| msg_{\ell'} \xleftarrow{n\text{-blocks}} \mathsf{pad}_{\ell' \cdot n}(msg)$
3: $s_0 \leftarrow \mathsf{pad}_n(nonce\|meta) \| 0^{b-n-|key|} \| key$
4: **for** $i = 1, \ldots, \ell'$ **do**
5:     $s_i \leftarrow \pi(s_{i-1})$
6:     $s_i \leftarrow s_i \oplus (msg_i \| 0^*)$
7:     $ctxt_i \leftarrow \lceil s_i \rceil_n$
8: $s_{\ell'+1} \leftarrow \pi(s_{\ell'})$
9: **return** $(\lceil ctxt_1 \| \cdots \| ctxt_{\ell'} \rceil_{|msg|}, \lceil s_{\ell'+1} \rceil_\alpha)$

---

including message authentication [6,14,48,49] and stream encryption [15,45]. We will use the keyed sponge in the full-state duplex mode [45], to describe an authenticated encryption scheme that is suited for the use in Sphinx. As keyed Sponges are merely stream based encryption, a unique nonce is required for every encryption.

The realization of our authenticated encryption scheme using the sponge is dubbed $\text{AE}^{\pi,\ell,n}$. It is indexed by a permutation π of width b and parameters ℓ and $n \leq b$ which specify the parsing of the message blocks: it considers at most ℓ message blocks of n bits. The parameter ℓ can be arbitrarily large, but it is used to show how the length affects the security bound. $\text{AE}^{\pi,\ell,n}$ operates on keys of size $\kappa \leq b-n$ bits, messages and ciphertexts can be of length at most $\mu = \ell \cdot n - 1$ (note that the scheme does not use ciphertext expansion, hence $\mu = \nu$), and the sizes of the associated data and nonce should satisfy $\sigma + \tau \leq n-1$. The size of the authentication tag is $\alpha \leq n$ (this bound is for simplicity, the scheme generalizes to $\alpha > n$). $\text{AE}^{\pi,\ell,n}$ is depicted in Figure 2. The formal encryption and decryption functionalities are given in Algorithms 1 and 2.

4.1 Security

$\text{AE}^{\pi,\ell,n}$ is in fact a full-state duplex construction [45], but it is easier to explain the security of the construction in terms of the Inner-Keyed Sponge (IKS) of Andreeva et al. [6]. This construction gets as input a key k, an arbitrarily sized message m, and a natural number ρ, and it outputs a digest z of size ρ. That is $\text{IKS}^{\pi}(k, m, \rho) = z \in \{0,1\}^{\rho}$. It is defined as the classical Sponge with an outer

---

Algorithm 2 $\text{Dec}^{\pi,\ell,n}$

Input: $(key, ctxt, auth, meta, nonce) \in \{0,1\}^\kappa \times \{0,1\}^{\leq \mu} \times \{0,1\}^\alpha \times \{0,1\}^{\leq \tau} \times \{0,1\}^\sigma$
Output: $msg \in \{0,1\}^{\leq \mu}$ or $\bot$
1: $\ell' \leftarrow \lceil(|ctxt| + 1)/n\rceil$
2: $ctxt_1 \| \cdots \| ctxt_{\ell'} \xleftarrow{n\text{-blocks}} \mathsf{pad}_{\ell' \cdot n}(ctxt)$
3: $s_0 \leftarrow \mathsf{pad}_n(nonce\|meta) \| 0^{b-n-|key|} \| key$
4: **for** $i = 1, \ldots, \ell' - 1$ **do**
5:     $s_i \leftarrow \pi(s_{i-1})$
6:     $msg_i \leftarrow \lceil s_i\rceil_n \oplus ctxt_i$
7:     **if** $i < \ell'$ **then**
8:         $s_i \leftarrow ctxt_i \| \lfloor s_i\rfloor_{b-n}$
9:     **else**
10:         $s_i \leftarrow \lceil ctxt_i\rceil_{|ctxt| \bmod n} \| \lfloor s_i\rfloor_{b-(|ctxt| \bmod n)}$
11: $msg \leftarrow \lceil msg_1 \| \cdots \| msg_{\ell'}\rceil_{|ctxt|}$
12: $s_{\ell'+1} \leftarrow \pi(s_{\ell'})$
13: **return** $\lceil s_{\ell'+1}\rceil_\alpha = auth \,?\, msg : \bot$

---

part of size n and an inner part of size $b - n$, and with the capacity part being initialized using the key. We consider a specific case of IKS where $\rho \leq n$, which means that the squeezing part of the Sponge takes exactly one round.

The security of variable-input-length IKS : $\mathcal{K} \times \{0,1\}^* \to \{0,1\}^n$ based on a permutation π is slightly different from the CPA security of Section 3; first, IKS is variable length, so it is compared with a random oracle $\mathcal{RO} : \{0,1\}^* \to \{0,1\}^n$, and second, it is based on an underlying idealized permutation π and the adversary has two-sided oracle access to π. Denote by $\mathsf{Perm}(\{0,1\}^b)$ the set of b-bit permutations. Abusing notation, we refer to the security of IKS against an adversary that has access to either $(\text{IKS}, \pi^\pm)$ or $(\mathcal{RO}, \pi^\pm)$, where $k \xleftarrow{\$} \mathcal{K}$, $\pi \xleftarrow{\$} \mathsf{Perm}(\{0,1\}^b)$, and $\mathcal{RO}$ is a random oracle, by $\mathbf{Adv}_{\text{IKS}}^{\text{cpa}}(\mathcal{A})$. We define by $\mathbf{Adv}_{\text{IKS}}^{\text{cpa}}(Q, S)$ the maximum advantage over all adversaries with total complexity Q, that make at most S primitive queries to $\pi^\pm$. The total complexity Q counts the number of *fresh calls* to π if $\mathcal{A}$ were conversing with IKS.

Note that if no authentication is needed, then $\text{Enc}^{\pi,\ell,n}$ and $\text{Dec}^{\pi,\ell,n}$ do not require the computation of $s_{\ell'+1} \leftarrow \pi(s_{\ell'})$ at the end, which saves a permutation call. Related to this, we define ℓ_α as ℓ, if $\alpha = 0$, and $\ell + 1$ otherwise.

Theorem 1. *Assume $\pi \xleftarrow{\$} \mathsf{Perm}(\{0,1\}^b)$ is an ideal permutation. We have*

$$\mathbf{Adv}_{\text{AE}^{\pi,\ell,n}}^{\text{cpa}}(Q, T) \leq \frac{(\ell_a Q)^2}{2^{b-n}} + \frac{\ell_a Q S}{2^\kappa},$$

$$\mathbf{Adv}_{\text{AE}^{\pi,\ell,n}}^{\text{auth}}(Q, R, T) \leq \frac{(\ell_a Q)^2}{2^{b-n}} + \frac{\ell_a Q S}{2^\kappa} + \frac{R}{2^\alpha}.$$

where S is the maximal number of evaluations of π that can be made in time T. The adversaries are required to be nonce respecting.

Proof. Let $\mathcal{A}$ be an adversary that makes Q queries and runs in time T. It has access to either Enc_{key} or \$. Consider any evaluation Enc_{key} on input of

(msg, $meta$, $nonce$). If we define $k = 0^{b-n-|key|} \| key$, then its output is as follows,

$$\begin{aligned}
ctxt = {} & \text{IKS}^\pi(k, \text{pad}_n(nonce \| meta), n) \oplus msg_1 \; \| \\
& \text{IKS}^\pi(k, \text{pad}_n(nonce \| meta) \| msg_1, n) \oplus msg_2 \; \| \\
& \cdots \\
& \text{IKS}^\pi(k, \text{pad}_n(nonce \| meta) \| msg_1 \cdots msg_{\ell-2}, n) \oplus msg_{\ell-1} \; \| \\
& \lceil \text{IKS}^\pi(k, \text{pad}_n(nonce \| meta) \| msg_1 \cdots msg_{\ell-1}, n) \oplus msg_\ell \rceil_{|msg| \bmod n} \, , \\
auth = {} & \text{IKS}^\pi(k, \text{pad}_n(nonce \| meta) \| msg_1 \cdots msg_\ell, \alpha)
\end{aligned}$$

Where abusing notation, $|msg| \bmod n \in \{1, \dots, n\}$. In other words, any evaluation of Enc_{key} entails ℓ_α evaluations of IKS (the computation of $auth$ is omitted if $\alpha = 0$). Each of these evaluations adds 1 to the complexity (as it is simply an extension of the previous one). Thus, after Q evaluations of Enc_{key}, IKS is evaluated with a total complexity $\ell_a Q$. We replace IKS by a random oracle $\mathcal{RO}$. This step costs us $\mathbf{Adv}_{\text{IKS}}^{\text{cpa}}(\ell_a Q, S)$, where S is as described in the theorem statement.

Now, for the case of secrecy, recall that $\mathcal{A}$ is nonce respecting. Consequently, all evaluations of $\mathcal{RO}$ are made for a different input. This is clear for the $\ell + 1$ queries for a single evaluation; different evaluations of Enc_{key} are made under a different $nonce$ as the adversary is nonce respecting. Consequently, every query to Enc_{key} is responded with a uniformly randomly generated $|msg|$-bit value, and thus,

$$\mathbf{Adv}_{\text{AE}^{\pi,\ell,n}}^{\text{cpa}}(Q, T) \le \mathbf{Adv}_{\text{IKS}}^{\text{cpa}}(\ell_a Q, S) \, .$$

Next, for authenticity, it suffices to only focus on the value $auth$. Consider any forgery attempt ($ctxt$, $auth$, $meta$, $nonce$). Let msg be the message that is derived by $\text{Dec}^{\pi,\ell,n}$. As the forgery needs to be non-trivial, $\mathcal{RO}$ has not been queried on

$$\text{pad}_n(nonce \| meta) \| msg_1 \cdots msg_\ell$$

before. Its response $auth$ is thus a randomly generated value and the forgery is successful with probability $1/2^\alpha$. Again using [11],

$$\mathbf{Adv}_{\text{AE}^{\pi,\ell,n}}^{\text{auth}}(Q, R, T) \le \mathbf{Adv}_{\text{IKS}}^{\text{cpa}}(\ell_a Q, S) + \frac{R}{2^\alpha} \, .$$

Now, in [6] it is proven that[1] $\mathbf{Adv}_{\text{IKS}}^{\text{cpa}}(Q', S) \le \frac{(Q')^2}{2^{b-n}} + \frac{Q'S}{2^\kappa}$, which completes the proof of both secrecy and authenticity. $\qquad\square$

5 Solution 2: Tweakable Blockcipher Based

The second approach is to apply a large tweakable blockcipher. A tweakable blockcipher $\widetilde{E} : \mathcal{K} \times \mathcal{T} \times \mathcal{M} \to \mathcal{M}$ takes as input a key $k \in \mathcal{K}$, a tweak $t \in \mathcal{T}$,

---

[1] We have slightly re-interpreted the result in order to accommodate the different key length.

Algorithm 3 $\mathtt{Enc}^{\mathrm{LRW[3fish]}}$

Input: $(key, msg, meta, nonce) \in \{0,1\}^\kappa \times \{0,1\}^{\leq \mu} \times bint \times \{0,1\}^\sigma$
Output: $(ctxt, auth) \in \{0,1\}^\nu \times \{0,1\}^\alpha$
 1: $c \leftarrow \mathrm{LRW[3fish]}(key,\ \mathsf{pad}_{1024}(meta), \mathsf{pad}_{128}(nonce), \mathsf{pad}_{1024}(msg))$
 2: **return** $(\lceil c \rceil_\nu, \lfloor c \rfloor_\alpha)$

Algorithm 4 $\mathtt{Dec}^{\mathrm{LRW[3fish]}}$

Input: $(key, ctxt, auth, meta, nonce) \in \{0,1\}^\kappa \times \{0,1\}^\nu \times \{0,1\}^\alpha \times \{0,1\}^{\leq \tau} \times \{0,1\}^\sigma$
Output: $msg \in \{0,1\}^{\leq \mu}$ or $\perp$
 1: $m \leftarrow \mathrm{LRW[3fish]}^{-1}(key,\ \mathsf{pad}_{1024}(meta), \mathsf{pad}_{128}(nonce), ctxt \| auth)$
 2: $msg \leftarrow \mathsf{unpad}_{\mu+1}(\lceil m \rceil_{\mu+1})$
 3: **return** $\lfloor m \rfloor_\alpha = 0$? msg : $\perp$

---

and a message $m \in \mathcal{M}$, and outputs a ciphertext $c \in \mathcal{M}$. It is a permutation for every choice of (k, t).

For our authenticated encryption functionality AE, we will need a tweakable blockcipher with a large state $\mathcal{M}$. To this end, we suggest *Threefish*, a tweakable blockcipher by Ferguson et al. used for the Skein hash function family [26]. Threefish supports block sizes of 256, 512, and 1024 bits. The key size equals the block size, and the tweak size is 128 bits. We focus on the largest variant, Threefish-1024, which for readability we simply denote 3fish:

$$3\mathrm{fish} : \{0,1\}^{1024} \times \{0,1\}^{128} \times \{0,1\}^{1024} \to \{0,1\}^{1024},$$
$$(k, t, m) \mapsto c.$$

3fish can be used for authenticated encryption directly, i.e., by placing the associated data and nonce into the tweak, and encrypting based on the secret key and this tweak. While the state size of 3fish is reasonably large enough (for certain applications), the tag size is not. One way to resolve this is to employ a random oracle that maps the associated data and nonce to a string of size 128 bits, but this would degrade the security of the construction as forgeries can be found in a complexity 2^{64}. Another way to enlarge the tweak space without adjusting the cipher itself is by using it in a tweakable mode of operation.

Liskov et al. [43] introduced two tweakable modes of operation: while these constructions are originally designed to add a tweak input to a blockcipher, they can equally well be applied to tweakable blockciphers themselves to enlarge the tweak space. We will consider one of these constructions, which makes two evaluations of the underlying cipher:[2]

$$\mathrm{LRW[3fish]} : \{0,1\}^{1024} \times \{0,1\}^{1024} \times \{0,1\}^{128} \times \{0,1\}^{1024} \to \{0,1\}^{1024},$$
$$(k, t, t', m) \mapsto 3\mathrm{fish}(k, t', 3\mathrm{fish}(k, t', m) \oplus t).$$

---

[2] The other construction is less relevant as it requires an additional key and needs a universal hash function with a 1024-bit range (or smaller, in which case the security of the construction degrades).
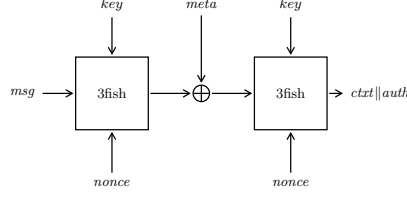
Fig. 3. AE based on LRW[3fish]. Padding of data is excluded from the figure

This construction can be used to realize $\text{AE}^{\text{LRW}[\text{3fish}]}$ as illustrated in Figure 3 and described in Algorithms 3 and 4. It operates on keys of size $\kappa = 1024$ bits, messages can be of arbitrary length but of size at most $\mu = 1023 - \alpha$, the nonce should be of size $\sigma \leq 127$, and the associated data should be of size at most $\tau \leq 1023$. The ciphertexts are of size *exactly* $\nu = 1024 - \alpha$ bits, where α is the size of the authentication tag. The latter is required to make decryption possible.

5.1 Security

We formally derive the security of $\text{AE}^{\text{LRW}[\text{3fish}]}$ under the assumption that 3fish is a secure tweakable blockcipher. The security of a tweakable blockcipher $\widetilde{E} : \mathcal{K} \times \mathcal{T} \times \mathcal{M} \to \mathcal{M}$ is captured by an adversary $\mathcal{A}$ that has adaptive two-sided oracle access to either $\widetilde{E}_k$ for some secret key $k \xleftarrow{\$} \mathcal{K}$, or ideal tweakable permutation $\widetilde{\pi}$ with tweak space $\mathcal{T}$ and message space $\mathcal{M}$, and tries to distinguish both worlds. Denote by $\widetilde{\text{Perm}}(\mathcal{T}, \mathcal{M})$ the set of tweakable permutations. We define the strong PRP security of $\widetilde{E}$ as

$$\mathbf{Adv}_{\widetilde{E}}^{\widetilde{\text{sprp}}}(\mathcal{A}) = \left| \Pr\left(k \xleftarrow{\$} \mathcal{K} \ : \ \mathcal{A}^{\widetilde{E}_k^{\pm}} = 1 \right) - \Pr\left(\widetilde{\pi} \xleftarrow{\$} \widetilde{\text{Perm}}(\mathcal{T}, \mathcal{M}) \ : \ \mathcal{A}^{\widetilde{\pi}^{\pm}} = 1 \right) \right| .$$

By $\mathbf{Adv}_{\widetilde{E}}^{\widetilde{\text{sprp}}}(Q, T)$ we denote the maximum security advantage of any adversary $\mathcal{A}$ that makes Q queries and runs in time T.

The security of $\text{AE}^{\text{LRW}[\text{3fish}]}$ follows from a result from [43] on the security of LRW.

Theorem 2. *Let $n = 1024$ be the state size of* 3fish. *We have*

$$\mathbf{Adv}_{\text{AE}^{\text{LRW}[\text{3fish}]}}^{\text{cpa}}(Q, T) \leq \Theta\left(\frac{Q^2}{2^n} \right) + \mathbf{Adv}_{\text{3fish}}^{\widetilde{\text{sprp}}}(2Q, T') \,,$$

$$\mathbf{Adv}_{\text{AE}^{\text{LRW}[\text{3fish}]}}^{\text{auth}}(Q, R, T) \leq \Theta\left(\frac{(Q+R)^2}{2^n} \right) + \mathbf{Adv}_{\text{3fish}}^{\widetilde{\text{sprp}}}(2(Q+R), T') + \frac{R 2^{n-\alpha}}{2^n - Q} \,,$$

where $T' \approx T$. The adversaries may be nonce reusing.

Proof. We start with the secrecy of $\text{AE}^{\text{LRW}[\text{3fish}]}$. Let $\mathcal{A}$ be an adversary that makes Q queries and runs in time T. It has access to either Enc_{key} or $\$$. Note that Q evaluations of AE^{3fish} induce Q evaluations of LRW[3fish]. We replace

LRW[3fish] by an ideal tweakable permutation $\widetilde{\pi} \xleftarrow{\$} \widetilde{\mathsf{Perm}}(\{0,1\}^{1024+128}, \{0,1\}^{1024})$. Now, any query $\mathtt{Enc}^{\widetilde{\pi}}(key, msg, meta, nonce)$ is responded with

$$ctxt \| auth = \widetilde{\pi}(key, \mathsf{pad}_{1024}(meta), \mathsf{pad}_{128}(nonce), \mathsf{pad}_{1024}(msg)) \,.$$

Note that every fresh $meta \| nonce$ initiates a new instance of $\widetilde{\pi}$ and results in a uniformly random 1024-bit value $ctxt \| auth$. On the other hand, if the adversary repeats a $meta \| nonce$, the value $ctxt \| auth$ will be generated uniformly at random from $\{0,1\}^n$ *without replacement*. In the ideal world, \$ always generates uniformly random responses from $\{0,1\}^{1024}$. This means that $\mathtt{Enc}^{\widetilde{\pi}}_{key}$ is indistinguishable from \$ up to bound $\binom{Q}{2}/2^n$. Thus, writing $n = 1024$:

$$\mathbf{Adv}^{\mathrm{cpa}}_{\mathrm{AE}^{\mathrm{LRW[3fish]}}}(Q, T) \leq \mathbf{Adv}^{\widetilde{\mathrm{sprp}}}_{\mathrm{LRW[3fish]}}(Q, T'') + \frac{\binom{Q}{2}}{2^n} \,,$$

where $T'' \approx T$.

For authenticity, the first part of the proof is identical: we replace LRW[3fish] by ideal tweakable permutation $\widetilde{\pi} \xleftarrow{\$} \widetilde{\mathsf{Perm}}(\{0,1\}^{1024+128}, \{0,1\}^{1024})$, where now the $Q + R$ evaluations of $\mathrm{AE}^{\mathrm{LRW[3fish]}}$ induce $Q + R$ evaluations of LRW[3fish]. It remains to consider the probability to forge an authentication tag for $\mathrm{AE}^{\widetilde{\pi}}$. By [11], it suffices to consider any attempt and sum over all R attempts. Consider any forgery attempt $(ctxt, auth, meta, nonce)$. The value

$$m = \widetilde{\pi}^{-1}(key, \mathsf{pad}_{1024}(meta), \mathsf{pad}_{128}(nonce), ctxt \| auth)$$

is randomly drawn from a set of size at least $2^n - Q$, as there have been at most Q encryption queries under associated data $meta$ and nonce $nonce$, and satisfies $\lfloor m \rfloor_\alpha = 0$ with probability at most $2^{n-\alpha}/(2^n - Q)$. Thus:

$$\mathbf{Adv}^{\mathrm{auth}}_{\mathrm{AE}^{\mathrm{LRW[3fish]}}}(Q, R, T) \leq \mathbf{Adv}^{\widetilde{\mathrm{sprp}}}_{\mathrm{LRW[3fish]}}(Q + R, T'') + \frac{R 2^{n-\alpha}}{2^n - Q} \,,$$

where $T'' \approx T$.

In [43] it is proven that $\mathbf{Adv}^{\widetilde{\mathrm{sprp}}}_{\mathrm{LRW[3fish]}}(Q, T'') \leq \Theta\left(\frac{Q^2}{2^n}\right) + \mathbf{Adv}^{\widetilde{\mathrm{sprp}}}_{\mathrm{3fish}}(2Q, T')$, where $T' \approx T''$. $\qquad\square$

We briefly remark that the construction is even secure under *release of unverified plaintext*, where msg is disclosed before tag verification is done [4]. We also note that, if one considers *nonce respecting* adversaries, the term $\binom{Q}{2}/2^n$ in the analysis of secrecy disappears, and $\frac{R 2^{n-\alpha}}{2^n - Q}$ can be replaced with $\frac{R 2^{n-\alpha}}{2^n - 1}$.

6 Improving the Sphinx

A naive solution to the state of affairs for Sphinx (Section 2) would be to replace SEAL by a more modern stream cipher and to replace SHA-1 by SHA-3, but there is little point in doing so: versatility of Sponges in general and SHA-3

in particular enables encryption using SHA-3 on the fly; putting a four-round Feistel construction on top of it is overkill. Instead, it makes more sense to simply *replace* LIONESS by a keyed version of the SHA-3. The construction of Section 4 is particularly suited for this purpose, as it is an AE scheme based on the SHA-3 permutation. As the construction offers authenticated encryption, it can also be used to replace the MAC. In other words, where the original Sphinx MACs β into authentication tag γ and encrypts the payload into δ (both using secret session key s), the construction of Section 4 neatly merges those into

$$(\delta, \gamma) = \text{AE}(s, \text{payload}, \beta), \tag{1}$$

where, for the sake of simplicity, β now represents the associated data and the nonce. We have henceforth obtained the security and efficiency improvement promised in Section 2.

It seems logical to also replace the remaining cryptographic functionalities in Sphinx by SHA-3. However, a second though reveals that, again, there is little point in doing so: first hashing a key through SHA-3 and then considering the keyed version of the SHA-3 based on this key is plainly less efficient *and* less secure than considering the keyed version of the SHA-3 based on the original key. Therefore, it suffices to have a simple mapping that transforms the secret session key, which is a group element by default, into a bit string.

The downside of the SHA-3 based approach is that the authenticated encryption scheme of Section 4 does not offer security against nonce reusing adversaries, and in the solution of (1), β represents the associated data as well as the nonce. In Sphinx, the β values are generated using the PRG, and henceforth random, but collisions may appear. Alternatively, one can use the Threefish based mode of Section 5, and use the Skein hash function family [26] to serve for hashing, as it already uses Threefish natively.

Either approach makes the encryption functionality of Sphinx more secure and more efficient. Figure 4 depicts our proposal of using authenticated encryption in Sphinx. Our authenticated encryption solutions support associated data as input which could be used for the processing of the header, it natively allows for authentication, and could potentially be used as MAC function. These advantages could be exploited to integrate part of the nested MAC functionality of Sphinx within the AE. As a bonus, Sphinx using our AE schemes *additionally* authenticates the payload for free, a feature that was missing in the original Sphinx.

References

1. Anderson, R.J., Biham, E.: Two practical and provably secure block ciphers: BEARS and LION. In: FSE '96. LNCS, vol. 1039, pp. 113–120. Springer (1996)
2. Andreeva, E., Bilgin, B., Bogdanov, A., Luykx, A., Mendel, F., Mennink, B., Mouha, N., Wang, Q., Yasuda, K.: PRIMATEs v1.02 (2015), submission to CAESAR competition
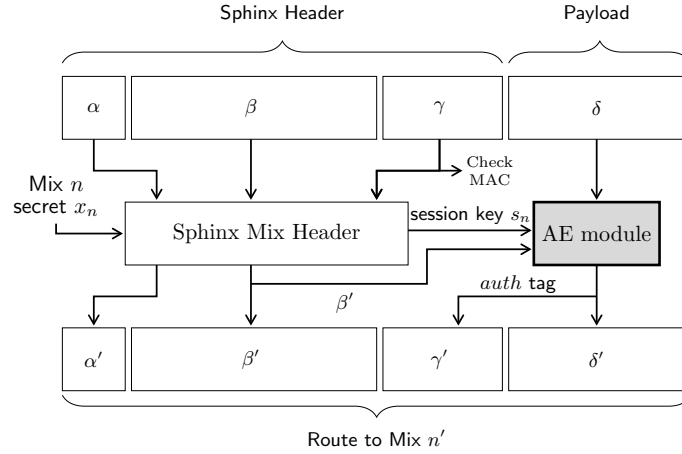
Fig. 4. Using the AE scheme of Section 4 or Section 5 in Sphinx. The mix header remains mostly unchanged. Message authentication and encryption are now merged into AE, where β functions as the associated data.

3. Andreeva, E., Bilgin, B., Bogdanov, A., Luykx, A., Mennink, B., Mouha, N., Yasuda, K.: APE: authenticated permutation-based encryption for lightweight cryptography. In: FSE 2014. LNCS, vol. 8540, pp. 168–186. Springer (2015)

4. Andreeva, E., Bogdanov, A., Luykx, A., Mennink, B., Mouha, N., Yasuda, K.: How to securely release unverified plaintext in authenticated encryption. In: ASIACRYPT 2014, Part I. LNCS, vol. 8873, pp. 105–125. Springer (2014)

5. Andreeva, E., Bogdanov, A., Luykx, A., Mennink, B., Tischhauser, E., Yasuda, K.: Parallelizable and authenticated online ciphers. In: ASIACRYPT 2013, Part I. LNCS, vol. 8269, pp. 424–443. Springer (2015)

6. Andreeva, E., Daemen, J., Mennink, B., Van Assche, G.: Security of keyed Sponge constructions using a modular proof approach. In: FSE 2015. LNCS, vol. 9054, pp. 364–384. Springer (2015)

7. Aumasson, J., Çalik, Çagdas., Meier, W., Özen, O., Phan, R., Varıcı, K.: Improved cryptanalysis of Skein. In: ASIACRYPT 2009. LNCS, vol. 5912, pp. 542–559. Springer (2009)

8. Aumasson, J., Jovanovic, P., Neves, S.: NORX v2.0 (2015), submission to CAESAR competition

9. Bellare, M., Canetti, R., Krawczyk, H.: Keying hash functions for message authentication. In: CRYPTO '96. LNCS, vol. 1109, pp. 1–15. Springer (1996)

10. Bellare, M., Desai, A., Jokipii, E., Rogaway, P.: A concrete security treatment of symmetric encryption. In: FOCS 1997. pp. 394–403. IEEE Computer Society (1997)

11. Bellare, M., Goldreich, O., Mityagin, A.: The power of verification queries in message authentication and authenticated encryption. Cryptology ePrint Archive, Report 2004/309 (2004)

12. Bellare, M., Rogaway, P., Wagner, D.: The EAX mode of operation. In: FSE 2004. LNCS, vol. 3017, pp. 389–407. Springer (2004)

13. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: The Keccak reference (January 2011)

14. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: On the security of the keyed Sponge construction. Symmetric Key Encryption Workshop (SKEW 2011) (2011)
15. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: Duplexing the Sponge: Single-pass authenticated encryption and other applications. In: SAC 2011. LNCS, vol. 7118, pp. 320–337. Springer (2012)
16. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: Sponge functions (ECRYPT Hash Function Workshop 2007)
17. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G., Van Keer, R.: Keyak v2 (2015), submission to CAESAR competition
18. Biryukov, A., Khovratovich, D.: PAEQ: parallelizable permutation-based authenticated encryption. In: ISC 2014. LNCS, vol. 8783, pp. 72–89. Springer (2014)
19. CAESAR: Competition for Authenticated Encryption: Security, Applicability, and Robustness (March 2014)
20. Chen, J., Jia, K.: Improved related-key boomerang attacks on round-reduced Threefish-512. In: ISPEC 2010. LNCS, vol. 6047, pp. 1–18. Springer (2010)
21. Cogliati, B., Lampe, R., Seurin, Y.: Tweaking even-mansour ciphers. In: CRYPTO 2015, Part I. LNCS, vol. 9215, pp. 189–208. Springer (2015)
22. Danezis, G., Goldberg, I.: Sphinx: A compact and provably secure mix format. In: IEEE S 2009. pp. 269–282. IEEE Computer Society (2009)
23. Datta, N., Nandi, M.: ELmE: A misuse resistant parallel authenticated encryption. In: ACISP 2014. LNCS, vol. 8544, pp. 306–321. Springer (2014)
24. De Cannière, C., Rechberger, C.: Finding SHA-1 characteristics: General results and applications. In: ASIACRYPT 2006. LNCS, vol. 4284, pp. 1–20. Springer (2006)
25. Dobraunig, C., Eichlseder, M., Mendel, F., Schläffer, M.: Ascon v1.1 (2015), submission to CAESAR competition
26. Ferguson, N., Lucks, S., Schneier, B., Whiting, D., Bellare, M., Kohno, T., Callas, J., Walker, J.: The Skein hash function family (2010), submission to NIST's SHA-3 competition
27. Fluhrer, S.R.: Cryptanalysis of the SEAL 3.0 pseudorandom function family. In: FSE 2001. LNCS, vol. 2355, pp. 135–143. Springer (2002)
28. Fuhr, T., Leurent, G., Suder, V.: Collision attacks against CAESAR candidates - forgery and key-recovery against AEZ and marble. In: ASIACRYPT 2015, Part II. LNCS, vol. 9453, pp. 510–532. Springer (2015)
29. Gligoroski, D., Mihajloska, H., Samardjiska, S., Jacobsen, H., El-Hadedy, M., Jensen, R.: π-Cipher v2.0 (2015), submission to CAESAR competition
30. Hoang, V.T., Krovetz, T., Rogaway, P.: AEZ v4: Authenticated Encryption by Enciphering (2015), submission to CAESAR competition
31. Hoang, V., Krovetz, T., Rogaway, P.: Robust authenticated-encryption AEZ and the problem that it solves. In: EUROCRYPT 2015, Part I. LNCS, vol. 9056, pp. 15–44. Springer (2015)
32. Jean, J., Nikolic, I., Peyrin, T.: Tweaks and keys for block ciphers: The TWEAKEY framework. In: ASIACRYPT 2014, Part II. LNCS, vol. 8874, pp. 274–288. Springer (2014)
33. Jean, J., Nikolić, I., Peyrin, T.: Deoxys v1.3 (2015), submission to CAESAR competition
34. Jean, J., Nikolić, I., Peyrin, T.: Joltik v1.3 (2015), submission to CAESAR competition
35. Jovanovic, P., Luykx, A., Mennink, B.: Beyond $2^{c/2}$ security in sponge-based authenticated encryption modes. In: ASIACRYPT 2014, Part I. LNCS, vol. 8873, pp. 85–104. Springer (2014)

36. Karpman, P., Peyrin, T., Stevens, M.: Practical free-start collision attacks on 76-step SHA-1. In: CRYPTO 2015, Part I. LNCS, vol. 9215, pp. 623–642. Springer (2015)
37. Kavun, E., Lauridsen, M., Leander, G., Rechberger, C., Schwabe, P., Yalçın, T.: Prøst v1 (2014), submission to CAESAR competition
38. Khovratovich, D., Nikolić, I.: Rotational cryptanalysis of ARX. In: FSE 2010. LNCS, vol. 6147, pp. 333–346. Springer (2010)
39. Khovratovich, D., Nikolić, I., Pieprzyk, J., Sokołowski, P., Steinfeld, R.: Rotational cryptanalysis of ARX revisited. In: FSE 2015. LNCS, vol. 9054, pp. 519–536. Springer (2015)
40. Khovratovich, D., Nikolić, I., Rechberger, C.: Rotational rebound attacks on reduced Skein. In: ASIACRYPT 2010. LNCS, vol. 6477, pp. 1–19. Springer (2010)
41. Krovetz, T., Rogaway, P.: The software performance of authenticated-encryption modes. In: FSE 2011. LNCS, vol. 6733, pp. 306–327. Springer (2011)
42. Leurent, G., Roy, A.: Boomerang attacks on hash function using auxiliary differentials. In: CT-RSA 2012. LNCS, vol. 7178, pp. 215–230. Springer (2012)
43. Liskov, M., Rivest, R.L., Wagner, D.: Tweakable block ciphers. In: CRYPTO 2002. LNCS, vol. 2442, pp. 31–46. Springer (2002)
44. Mennink, B.: Observation on weak keys for AEZ (2016), CAESAR mailing list
45. Mennink, B., Reyhanitabar, R., Vizár, D.: Security of full-state keyed and duplex sponge: Applications to authenticated encryption. In: ASIACRYPT 2015, Part II. LNCS, vol. 9453, pp. 465–489. Springer (2015)
46. Minematsu, K.: Parallelizable rate-1 authenticated encryption from pseudorandom functions. In: EUROCRYPT 2014. LNCS, vol. 8441, pp. 275–292. Springer (2014)
47. Morawiecki, P., Gaj, K., Homsirikamol, E., Matusiewicz, K., Pieprzyk, J., Rogawski, M., Srebrny, M., Wójcik, M.: ICEPOLE v2 (2015), submission to CAESAR competition
48. Naito, Y., Yasuda, K.: New bounds for keyed Sponges with extendable output: Independence between capacity and message length. In: FSE 2016. LNCS, Springer (2016), to appear
49. Peter Gaži, Pietrzak, K., Tessaro, S.: The exact PRF security of truncation: Tight bounds for keyed sponges and truncated CBC. In: CRYPTO 2015, Part I. LNCS, vol. 9215, pp. 368–387. Springer (2015)
50. Rogaway, P.: Efficient instantiations of tweakable blockciphers and refinements to modes OCB and PMAC. In: ASIACRYPT 2004. LNCS, vol. 3329, pp. 16–31. Springer (2004)
51. Rogaway, P., Bellare, M., Black, J., Krovetz, T.: OCB: a block-cipher mode of operation for efficient authenticated encryption. In: ACM Conference on Computer and Communications Security. pp. 196–205. ACM (2001)
52. Rogaway, P., Coppersmith, D.: A software-optimised encryption algorithm. In: FSE '93. LNCS, vol. 809, pp. 56–63. Springer (1993)
53. Saarinen, M., Brumley, B.: STRIBOB v2 (2015), submission to CAESAR competition
54. Stevens, M.: New collision attacks on SHA-1 based on optimal joint local-collision analysis. In: EUROCRYPT 2013. LNCS, vol. 7881, pp. 245–261. Springer (2013)
55. Stevens, M., Karpman, P., Peyrin, T.: Freestart collision for full SHA-1. In: EUROCRYPT 2016, Part I. LNCS, vol. 9665, pp. 459–483. Springer (2016)
56. Wang, X., Yin, Y., Yu, H.: Finding collisions in the full SHA-1. In: CRYPTO 2005. LNCS, vol. 3621, pp. 17–36. Springer (2005)
57. Yu, H., Chen, J., Wang, X.: The boomerang attacks on the round-reduced Skein-512. In: SAC 2012. LNCS, vol. 7707, pp. 287–303. Springer (2013)