Anonymous

AITO: Compact Authenticated Encryption for Private and Secret Messaging

Abstract: Compact messaging services, such as microblogging and direct messaging, provide users with easy and reliable channels for dissemination of information. At the same time, networks offering those services store treasure troves of information to support their business model through targeted advertisements, and have become prime targets of censorship and surveillance based on the shared content. Current approaches that deliver privacy in such systems use traditional encryption techniques requiring large entropy and lack flexibility to selectively provide data utility, and thus, becoming unappealing for the service providers and unsuitable to compact messaging. We present AITO, a novel and secure authenticated encryption scheme tailored for usage in any compact messaging service. AITO guarantees semantic security while complying with the commonly imposed space restrictions in several services and allowing limited user defined data utility for providers. It consists of three distinct instantiations based on different primitives, tweakable blockciphers and Sponge functions, and offers higher levels of security than existing alternatives. In addition, we describe how AITO can be efficiently applied to different deployed systems, such as OTR, Sphinx, and Twitter. Finally, we discuss an implementation of a prototype, demonstrating the improved efficiency and low overhead of our constructions.

Keywords: Privacy, compact authenticated encryption, social media, microblogging, Sphinx.

DOI Editor to enter DOI
Received ..; revised ..; accepted ...

1 Introduction

With the large growth of Internet services, modern users rely more on digital and ubiquitous communications. Services such as Online Social Networks (OSNs), microblogging, and mobile direct messaging have become prominent communication channels by providing efficient as well as reliable sharing tools for dissemination and exchange of information. At the same time, given their prominent design and current business models, providers end up centralizing and storing large amounts of information about the users and their communications, thus leading to several privacy threats. For example, it is important for the users to be able to protect the content posted on OSN as well as have some mechanism for anonymous sending and receiving of messages over otherwise public or non-anonymous OSNs.

Encrypting the content delivered over such services is one way of protection. While many messaging services offer the possibility to encrypt messages, in OSNs this is usually not the case, and in many cases the business model of the OSN requires that it can collect and analyze the posted content and other information. Thus, allowing and facilitating the sharing of encrypted content, possibly even anonymously, is against their interests. Finding a good balance between privacy of the content and (some) ability to use some user-specified information about that content would be a win-win situation for privacy and the economic viability of the majority of today's OSNs.

One important tool in modern cryptography for achieving these goals is *authenticated encryption* (AE), which means that the content of a message is not only encrypted to some recipient, but also the integrity of that content is protected (and possibly the sender is also authenticated to the recipient). Many AE schemes have been introduced in the past, and the ongoing CAESAR competition [17] for the design of new AE schemes has induced renewed attention to the field.

1.1 Existing Solutions

The classical approach to AE design is blockcipher based, where a generic mode of operation is built on top of a blockcipher in order to process data blocks iteratively (see, for instance, [4, 9, 22, 28, 31, 34, 37, 38]). The usage of a mode of operation of this form entails overhead, e.g. in the form of padding, and the security level is in the end dominated by what the underlying primitive offers. For instance, by plugging the AES blockcipher into the above constructions, we achieve 128-bit security *at most*, and sometimes there is a possibility for distinguishability attacks in complexity of

Anonymous: (Anonymous Submission)

about 2^{64} (cf. Bellare et al. [7]). A more novel approach is to build authentication on top of a public permutation. This line of research has been popularized by the SpongeWrap construction [11], and various Sponge based CAESAR submissions follow this idea, such as [2, 6, 14, 24, 27, 35, 39]. These constructions use a permutation much larger in size than 128 bits, which in some cases allows for an increased level of security. However, all of above constructions offer 256-bit security at most [29]. In addition, recent advances on keyed Sponges have shown that a potentially significant speed-up is possible by compressing in full-state mode [33].

In general, such constructions require large entropy which may not be allowed by OSNs like Twitter and short messaging services, with the space constraints (140 characters, about 1120 *bits*). Also those constructions demand at least 8 AES evaluations achieving only 64-bit indistinguishability security. Aligned with the fact that general tweets (or text messages) are of a relatively short (but larger than 128 bits) length, the dedicated constructions presented by the different AITO approaches achieve higher efficiency results and a much higher level of security. Also, sharing encrypted information breaks business model of several systems relying on the mining of the shared content to provide targeted advertisements, which could cause providers to restrict and censor the distribution of the encrypted content. In fact, it has been shown that is actually possible to keep data utility and privacy by allowing providers to access the *"gist"* of information, i.e., a set limited information released that does not affect user's privacy [15].

1.2 Contributions and Outline

In this paper we propose AITO, a novel AE scheme that is tailored for compact messages (Section 2). Besides providing access control to the content by means of encryption, it additionally allows for authenticity, ensuring integrity of data. In particular, AITO employs recent advances in tweakable blockciphers (Sections 3 and 4) and keyed Sponge functions (Section 5) to achieve efficient authenticated encryption and high levels of security. We show that these constructions achieve provable security that is better than existing possibilities for this type of authenticated encryption with a tolerable cost in key size. We also give three practical use cases for AITO in protecting privacy in OSN settings.

AITO can be applied and used in many different applications, allowing security, integrity, and efficiency while enabling providers to prevail some data utility. For instance, it can be plugged into several messaging protocols, such as OTR [16] by replacing the symmetric key operation allowing higher security, as well as increasing efficiency for refreshing the session keys (Section 6.1). Furthermore, AITO can be used to increase the 128-bit security and efficiency of anonymized messages in a mix-network, such as Sphinx [21]. We propose that using AITO in the Sphinx mix-network scenario instead of the original LIONESS cipher from 1996 will provide greater security and also better performance with smaller headers (Section 6.2). In addition, AITO could be applied to Twitter, the widely popular microblogging service (Section 6.3), to enable secret messaging and access control by means of encryption, while enabling providers to keep the data utility of some information to support their business model, allowing users to selectively select the *"gist"* of information shared. Consequently also helping in saving some computational resources at the recipient side of the communication as it is possible to filter messages based on the hashtags, and thus limit the number of decryption operations. For instance, one could choose not to decrypt messages with the hashtag "#football" from a certain (or every) user.

In the final sections, we discuss our three constructions and present our conclusions (Section 8).

2 Model

This section introduces the required notation, AITO, and the respective threat model.

2.1 Notation

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 $N = \lceil N \rceil_m \| \lfloor N \rfloor_{n-m}$.

2.2 AITO

AITO is an authenticated encryption scheme for short data. Let $\lambda_{\max} \in \mathbb{N}$ be an integer that specifies the maximal size of an authenticated ciphertext (for instance $\lambda_{\max} = 1024$ or 1120). Let $\mu, \nu, \alpha, \tau, \sigma \in \mathbb{N}$ be size values that satisfy $\mu \le \nu$ and $\nu + \alpha + \tau + \sigma \le \lambda_{\max}$. Here, μ will denote the size of the message, ν the size of the ciphertext, τ the size of the meta data, and σ the size of the nonce. The value α determines the size of the authentication tag. If no authentication is needed, we simply have $\alpha = 0$.

AITO 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 $\mathit{key} \leftarrow \{0,1\}^{\kappa}$, whereas the `Enc` and `Dec` algorithms are defined as follows:

`Enc`:

Input: $(\mathit{key}, \mathit{msg}, \mathit{meta}, \mathit{nonce}) \in$
$\{0,1\}^{\kappa} \times \{0,1\}^{\le\mu} \times \{0,1\}^{\le\tau} \times \{0,1\}^{\sigma}$,

Output: $(\mathit{ctxt}, \mathit{auth}) \in$
$\{0,1\}^{\le\nu} \times \{0,1\}^{\alpha}$,

`Dec`:

Input: $(\mathit{key}, \mathit{ctxt}, \mathit{auth}, \mathit{meta}, \mathit{nonce}) \in$
$\{0,1\}^{\kappa} \times \{0,1\}^{\le\nu} \times \{0,1\}^{\alpha} \times \{0,1\}^{\le\tau} \times \{0,1\}^{\sigma}$,

Output: $\mathit{msg} \in \{0,1\}^{\le\mu}$ or $\perp$.

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

Various approaches of authenticated encryption schemes exist; see for instance the CAESAR competition [17] for the design of a new authenticated encryption scheme. These schemes are often generic modes of operation that process data blocks iteratively. AITO, however, is designed for small data, and orthogonal design approaches turn out to be more suitable. Particularly, the key principle of the design of AITO is inspired by tweakable blockciphers, where the meta data and nonce function as the tweak. Nevertheless, a similar threat model applies, and we discuss it next.

2.3 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, it is allowed to generate encryptions under a secret and unknown key itself. In this case, $\mathcal{A}$ should not learn the encrypted content, beyond that revealed in the meta data.

More technically, adversary $\mathcal{A}$ has query access to the encryption functionality `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. Here, $\mathcal{A}$ is required to be nonce respecting, meaning that every query must be made under a different nonce (see also Section 8). 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 AITO is defined as follows:

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

We define by $\mathbf{Adv}^{\mathrm{cpa}}_{\mathrm{AITO}}(Q, T)$ the maximum advantage over all adversaries that make at most Q encryption queries and operate in time T.

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

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

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

3 AITO: Basic Construction

The first 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 $t \in \mathcal{T}$, 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 AITO, we suggest using *Threefish*, a tweakable blockcipher by Ferguson et al. used for the Skein hash function family [25]. 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:

3fish:
 Input: $(k, t, m) \in \{0,1\}^{1024} \times \{0,1\}^{128} \times \{0,1\}^{1024}$,
 Output: $c \in \{0,1\}^{1024}$.

3fish can be used for authenticated encryption directly, a construction which we dub AITO$^{\text{3fish}}$. It operates on keys of size $\kappa = 1024$ bits, messages can be of arbitrary length but of size at most $\mu = 1023 - \alpha$, and the sizes of the meta data and nonce should satisfy $\sigma + \tau \leq 127$. 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. At a high level, the encryption consists of putting $m = \mathsf{pad}_{1024}(\mathit{msg})$ and $t = \mathsf{pad}_{128}(\mathit{nonce}\|\mathit{meta})$, and the ciphertext and authentication tag are derived as $\mathit{ctxt}\|\mathit{auth} = c$. Formally, the encryption and decryption of AITO$^{\text{3fish}}$ are defined as in Algorithms 1 and 2.

Algorithm 1 Enc$^{\text{3fish}}$

```
Input: $(\mathit{key}, \mathit{msg}, \mathit{meta}, \mathit{nonce}) \in$
    $\{0,1\}^{\kappa} \times \{0,1\}^{\leq\mu} \times \{0,1\}^{\leq\tau} \times \{0,1\}^{\sigma}$
Output: $(\mathit{ctxt}, \mathit{auth}) \in \{0,1\}^{\nu} \times \{0,1\}^{\alpha}$
  $c \leftarrow \text{3fish}(\mathit{key}, \mathsf{pad}_{128}(\mathit{nonce}\|\mathit{meta}), \mathsf{pad}_{1024}(\mathit{msg}))$
  return $(\lceil c \rceil_{\nu}, \lfloor c \rfloor_{\alpha})$
```

Algorithm 2 Dec$^{\text{3fish}}$

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

Security. In this section we formally derive the security under the assumption that 3fish is a secure tweakable blockcipher, so that the security of AITO$^{\text{3fish}}$ directly follows. 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 $\widetilde{\mathsf{Perm}}(\mathcal{T}, \mathcal{M})$ the set of tweakable permutations. We define the strong PRP security of $\widetilde{E}$ as

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

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

Theorem 1. *Let* $n = 1024$ *be the state size of* 3fish*. We have*

$$\mathbf{Adv}^{\mathrm{cpa}}_{\mathrm{AITO}^{\text{3fish}}}(Q, T) \leq \mathbf{Adv}^{\widetilde{\mathrm{sprp}}}_{\text{3fish}}(Q, T') ,$$
$$\mathbf{Adv}^{\mathrm{auth}}_{\mathrm{AITO}^{\text{3fish}}}(Q, R, T) \leq \mathbf{Adv}^{\widetilde{\mathrm{sprp}}}_{\text{3fish}}(Q+R, T') + \frac{R 2^{n-\alpha}}{2^n - 1} ,$$

where $T' \approx T$*.*

Proof. We start with the secrecy of AITO$^{\text{3fish}}$. Let $\mathcal{A}$ be an adversary that makes Q queries and runs in time T. It has access to either $\mathtt{Enc}_{\mathit{key}}$ or \$. Note that Q evaluations of AITO$^{\text{3fish}}$ induce Q evaluations of 3fish. We replace 3fish by an ideal tweakable permutation $\widetilde{\pi} \xleftarrow{\$} \widetilde{\mathsf{Perm}}(\{0,1\}^{128}, \{0,1\}^{1024})$. Now, any query $\mathtt{Enc}^{\widetilde{\pi}}(\mathit{key}, \mathit{msg}, \mathit{meta}, \mathit{nonce})$ is responded with

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

As $\mathcal{A}$ is required to be nonce respecting, every query is made under a new nonce, which means that every query initiates a new instance of $\widetilde{\pi}$, and $\mathit{ctxt}\|\mathit{auth}$ is a random 1024-bit value. This means that $\mathtt{Enc}^{\widetilde{\pi}}_{\mathit{key}}$ is perfectly indistinguishable from \$.

For authenticity, the first part of the proof is identical: we replace 3fish by ideal tweakable permutation $\widetilde{\pi} \xleftarrow{\$} \widetilde{\mathsf{Perm}}(\{0,1\}^{128}, \{0,1\}^{1024})$, where now the $Q + R$ evaluations of AITO$^{\text{3fish}}$ induce $Q + R$ evaluations of 3fish. It remains to consider the probability to forge an authentication tag for AITO$^{\widetilde{\pi}}$. By [8], it suffices to consider any attempt and sum over all R attempts. Consider any forgery attempt $(\mathit{ctxt}, \mathit{auth}, \mathit{meta}, \mathit{nonce})$. Note that, as $\mathcal{A}$ is required to be nonce respecting, there has been at most one encryption query under meta data *meta* and nonce *nonce*. Therefore, the value

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

is randomly drawn from a set of size at least $2^n - 1$, and satisfies $\lfloor m \rfloor_{\alpha} = 0$ with probability at most $2^{n-\alpha}/(2^n - 1)$. □

We briefly remark that the construction is even secure under *release of unverified plaintext*, where *msg* is disclosed before tag verification is done [3].

4 AITO: Expanded Tweak Space

The AITO basic construction presents a rather small tweak space, limiting the sizes of the meta data and nonce. In other words, the usage of a larger nonce results in a limitation on the size of the meta data. A way to resolve this is to employ a random oracle that maps the (larger) meta data and nonce to a string of size 128 bits, but this would significantly degrade the security of the construction to 64 bits. 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. [32] 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:[1]

$$
\begin{aligned}
&\mathrm{LRW}[\mathrm{3fish}]:\\
&\quad \textbf{Input: } (k,t,t',m) \in \{0,1\}^{1024} \times\\
&\qquad\qquad \{0,1\}^{1024} \times \{0,1\}^{128} \times \{0,1\}^{1024}\,,\\
&\quad \textbf{Output: } \mathrm{3fish}(k,t',\mathrm{3fish}(k,t',m)\oplus t) \in \{0,1\}^{1024}\,.
\end{aligned}
$$

This construction can be used to realize $\mathrm{AITO}^{\mathrm{LRW}[\mathrm{3fish}]}$ as illustrated in Figure 1 and described as in Algorithms 3 and 4. The conditions on the sizes of the inputs and outputs carry over from Section 3, with the difference that meta data should now be of size at most $\tau \leq 1023$.

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

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

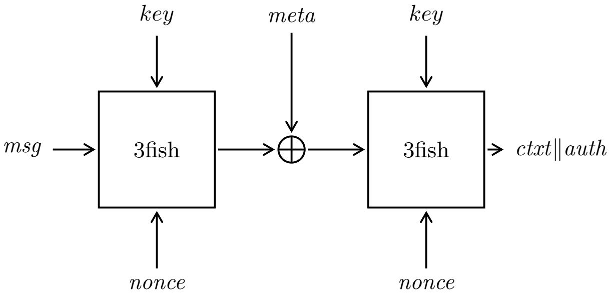

Fig. 1. AITO based on LRW[3fish]. Padding of data is excluded from the figure

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

```
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 $\bot$
  $m \leftarrow \mathrm{LRW}[\mathrm{3fish}]^{-1}(key,$
        $\mathsf{pad}_{1024}(meta), \mathsf{pad}_{128}(nonce), ctxt\|auth)$
  $msg \leftarrow \mathsf{unpad}_{\mu+1}(\lceil m\rceil_{\mu+1})$
  return $\lfloor m\rfloor_{\alpha} = 0$ ? $msg$ : $\bot$
```

Security. The security of $\mathrm{AITO}^{\mathrm{LRW}[\mathrm{3fish}]}$ in fact follows from Theorem 1 and a result from [32].

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

$$
\begin{aligned}
&\mathbf{Adv}^{\mathrm{cpa}}_{\mathrm{AITO}^{\mathrm{LRW}[\mathrm{3fish}]}}(Q,T) \leq \Theta\left(\frac{Q^2}{2^n}\right) + \mathbf{Adv}^{\widetilde{\mathrm{sprp}}}_{\mathrm{3fish}}(2Q,T')\,,\\
&\mathbf{Adv}^{\mathrm{auth}}_{\mathrm{AITO}^{\mathrm{LRW}[\mathrm{3fish}]}}(Q,R,T) \leq\\
&\qquad \Theta\left(\frac{(Q+R)^2}{2^n}\right) + \mathbf{Adv}^{\widetilde{\mathrm{sprp}}}_{\mathrm{3fish}}(2(Q+R),T') + \frac{R2^{n-\alpha}}{2^n-1}\,,
\end{aligned}
$$

where $T' \approx T$.

Proof. Note that the derivation in Theorem 1 not only applies to 3fish, but to any tweakable blockcipher. Applied to LRW[3fish] we get

$$
\begin{aligned}
&\mathbf{Adv}^{\mathrm{cpa}}_{\mathrm{AITO}^{\mathrm{LRW}[\mathrm{3fish}]}}(Q,T) \leq \mathbf{Adv}^{\widetilde{\mathrm{sprp}}}_{\mathrm{LRW}[\mathrm{3fish}]}(Q,T'')\,,\\
&\mathbf{Adv}^{\mathrm{auth}}_{\mathrm{AITO}^{\mathrm{LRW}[\mathrm{3fish}]}}(Q,R,T) \leq\\
&\qquad\qquad \mathbf{Adv}^{\widetilde{\mathrm{sprp}}}_{\mathrm{LRW}[\mathrm{3fish}]}(Q+R,T'') + \frac{R2^{n-\alpha}}{2^n-1}\,,
\end{aligned}
$$

where $T'' \approx T$. In [32] it is proven that

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

where $T' \approx T''$. □

[1] 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).

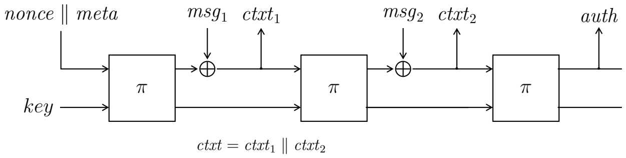

Fig. 2. AITO based on a Sponge. Padding of data is excluded from the figure.

5 AITO: Sponge Construction

The Sponge functions were originally introduced by Bertoni et al. [13] for cryptographic hashing, but can also be used in a broad spectrum of keyed applications, including message authentication [5, 10, 36] and stream encryption [11, 33]. They are also particularly useful for compact authenticated encryption, and hence to instantiate AITO. In more detail, we suggest the following function realization, which resembles ideas of the full-state duplex mode [33], transformed to the tweakable setting, as depicted in Figure 2. We stress, however, that the keyed Sponges are merely stream based encryption, and a unique nonce is required for every encryption.

The realization of $\mathrm{AITO}^{\pi,\ell,n}$ is indexed by a permutation π of width b and two parameters ℓ and $n \leq b$ which specify the way it parses the message blocks: it considers at most ℓ message blocks of size n bits. It operates on keys of size $\kappa \leq b - n$ bits, messages and ciphertexts can be of arbitrary 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 meta data and nonce should satisfy $\sigma + \tau \leq n - 1$. The size of the authentication tag is $\alpha \leq n$ (this bound is merely for simplicity, the scheme easily generalizes to $\alpha > n$). We additionally still require $\mu + \alpha + \tau + \sigma \leq \lambda_{\max}$. The formal encryption and decryption functionalities are given in Algorithms 5 and 6.

Algorithm 5 $\mathtt{Enc}^{\pi,\ell,n}$

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

Algorithm 6 $\mathtt{Dec}^{\pi,\ell,n}$

```
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 $\perp$
$\ell' \leftarrow \lceil(|ctxt| + 1)/n\rceil$
$ctxt_1 \parallel \cdots \parallel ctxt_{\ell'} \xleftarrow{n\text{-blocks}} \mathsf{pad}_{\ell' \cdot n}(ctxt)$
$s_0 \leftarrow \mathsf{pad}_n(nonce \| meta) \parallel 0^{b-n-|key|} \parallel key$
for $i = 1, \ldots, \ell' - 1$ do
    $s_i \leftarrow \pi(s_{i-1})$
    $msg_i \leftarrow \lceil s_i \rceil_n \oplus ctxt_i$
    if $i < \ell'$ then
        $s_i \leftarrow ctxt_i \parallel \lfloor s_i \rfloor_{b-n}$
    else
        $s_i \leftarrow \lceil ctxt_i \rceil_{|ctxt| \bmod n} \parallel \lfloor s_i \rfloor_{b-(|ctxt| \bmod n)}$
$msg \leftarrow \lceil msg_1 \parallel \cdots \parallel msg_{\ell'} \rceil_{|ctxt|}$
$s_{\ell'+1} \leftarrow \pi(s_{\ell'})$
return $\lceil s_{\ell'+1} \rceil_\alpha = auth$ ? $msg$ : $\perp$
```

Security. $\mathrm{AITO}^{\pi,\ell,n}$ is in fact a full-state duplex construction [33], but for the sake of presentation, it is easier to explain the security of the construction in terms of the Inner-Keyed Sponge (IKS) of Andreeva et al. [5]. 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:

$$\mathrm{IKS}^\pi(k, m, \rho) = z \in \{0,1\}^\rho\,.$$

It is defined as the classical Sponge with an outer 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 2.2; it differs in two aspects: 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 also 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 $(\mathrm{IKS}, \pi^\pm)$ 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 $\mathbf{Adv}^{\mathrm{cpa}}_{\mathrm{IKS}}(\mathcal{A})$. We define by $\mathbf{Adv}^{\mathrm{cpa}}_{\mathrm{IKS}}(Q, S)$ the maximum advantage over all adversaries with total complexity Q, and that make at most

S primitive queries to $\pi^{\pm}$. Here, 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 $\mathtt{Enc}^{\pi,\ell,n}$ and $\mathtt{Dec}^{\pi,\ell,n}$ 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 follows.

$$\ell_\alpha = \begin{cases} \ell\,, \text{ if } \alpha = 0\,, \\ \ell+1\,, \text{ if } \alpha > 0\,. \end{cases}$$

Theorem 3. *Assume $\pi \xleftarrow{\$} \mathsf{Perm}(\{0,1\}^b)$ is an ideal permutation. We have*

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

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

where S is the maximal number of evaluations of π that can be made in time T.

Proof. Let $\mathcal{A}$ be an adversary that makes Q queries and runs in time T. It has access to either $\mathtt{Enc}_{key}$ or \$. Consider any evaluation $\mathtt{Enc}_{key}$ on input of $(msg, meta, nonce)$. If we define $k = 0^{b-n-|key|}\|key$, then its output is as follows,

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

Where abusing notation, $|msg| \bmod n \in \{1, \ldots, n\}$. In other words, any evaluation of $\mathtt{Enc}_{key}$ 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 $\mathtt{Enc}_{key}$, IKS is evaluated with a total complexity $\ell_a Q$. We replace IKS by a random oracle $\mathcal{RO}$. This step costs us $\mathbf{Adv}^{\mathrm{cpa}}_{\mathrm{IKS}}(\ell_a Q, S)$, 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 $\mathtt{Enc}_{key}$ are made under a different *nonce* as the adversary is nonce respecting. Consequently, every query to $\mathtt{Enc}_{key}$ is responded with a uniformly randomly generated $|msg|$-bit value, and thus,

$$\mathbf{Adv}^{\mathrm{cpa}}_{\mathrm{AITO}^{\pi,\ell,n}}(Q,T) \leq \mathbf{Adv}^{\mathrm{cpa}}_{\mathrm{IKS}}(\ell_a Q, S)\,.$$

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 $\mathtt{Dec}^{\pi,\ell,n}$. As the forgery is required to be non-trivial, $\mathcal{RO}$ has never been queries on

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

before. Its response *auth* is thus a randomly generated value and the forgery is successful with probability $1/2^\alpha$. Again using [8],

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

Now, in [5] it is proven that[2]

$$\mathbf{Adv}^{\mathrm{cpa}}_{\mathrm{IKS}}(Q', S) \leq \frac{(Q')^2}{2^{b-n}} + \frac{Q'S}{2^\kappa}\,,$$

which completes the proof of both secrecy and authenticity. □

Instantiation. For π, we suggest to use the Keccak permutation $\pi_{\mathrm{keccak}} : \{0,1\}^{1600} \to \{0,1\}^{1600}$, and the following specific choices of (ℓ, n). As the complexity of the $\mathrm{AITO}^{\pi_{\mathrm{keccak}},\ell,n}$ increases linearly in ℓ, we suggest to use $\ell \leq 2$. If $\ell = 1$ and $n = 1024$, the messages can be of size at most 1024 bits, and $nonce\|meta$ is of size at most 1023 bits. Security up to approximately $\frac{1600-n}{2} = 288$ is achieved. In contrast, taking $\ell = 2$ and $n = 576$ gives flexible message lengths, $nonce\|meta$ should be of size at most 575, and 512-bit security is achieved.

6 Applications of AITO

In this section we discuss the practical application of the AITO constructions on different scenarios and use cases: low-latency messaging, secure mix-formatting and microblogging. However, we stress that AITO is suitable to any application requiring a compact output with high efficiency and security benefits.

6.1 Secret Messaging

The popularity of secret message services has been increasing since Edward Snowden's whistleblowing events.

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

Secret message protocols such as OTR [16], TextSecure[3], and Wickr[4] have been deployed into widely used mobile application. Most use common symmetric encryption schemes under an authenticated mode of operation, such as AES-CCM. In addition, each scheme requires a re-keying protocol for providing a different session key for each communication to provide forward secrecy. The usage of AITO in such protocols allows for greater security and efficiency (for short messages) and provides the possibility to use these protocols through the common text messaging channel, which is also limited to 140 characters for each message.

One recent proposal that would be especially suitable to be instantiated with AITO is Vuvuzela [42]. The anonymity provided by Vuvuzela is better than that of Tor, but the proposed implementation is geared towards short messages. Thus, AITO could be used in Vuvuzela to provide greater security for the message contents together with the extreme anonymity of messaging.

6.2 Secure Mix Formatting

Mix networks rely on (cryptography based) mix message formats that provide nice efficiency and security properties. Sphinx [21] 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.

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

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 $m_l \| m_r \leftarrow m$. These two blocks are then transformed using a four-round Feistel structure:

$$\begin{aligned} m_r &\leftarrow m_r \oplus S_{k_1}(m_l)\,, \\ m_l &\leftarrow m_l \oplus H_{k_2}(m_r)\,, \\ m_r &\leftarrow m_r \oplus S_{k_3}(m_l)\,, \\ m_l &\leftarrow m_l \oplus H_{k_4}(m_r)\,. \end{aligned}$$

The updated $m_l \| m_r$ constitutes the ciphertext c.

LIONESS enjoys a security proof if the underlying hash function SHA-1 and stream cipher SEAL are 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. In more detail, attacks on SEAL [26] and SHA-1 [23, 30, 40, 41, 43], 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.

A naive solution to this situation is 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 AITO Sponge construction of Section 5 is particularly suited for this purpose, as it is an authenticated encryption scheme based on the SHA-3 permutation. In this way, it seems logical to also replace the other cryptographic functionalities in Sphinx by SHA-3 or SHA-3 based alternatives. Alternatively, one can use the Threefish based version of AITO (Section 3 or Section 4), and use the Skein hash function family [25] to serve for hashing, as it already uses Threefish natively.

Either approach makes the encryption functionality of Sphinx more secure and more efficient. Figure 3 depicts a generic depiction of using AITO on Sphinx, such that, the Sphinx header on each Mix n is composed by the tuples (α, β, γ) and δ the payload message. In addition, AITO facilitates the twofold advantages: it supports tweaks an input which could be used for the processing of the header, and it 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 AITO.

[3] https://whispersystems.org/

[4] https://wickr.com/

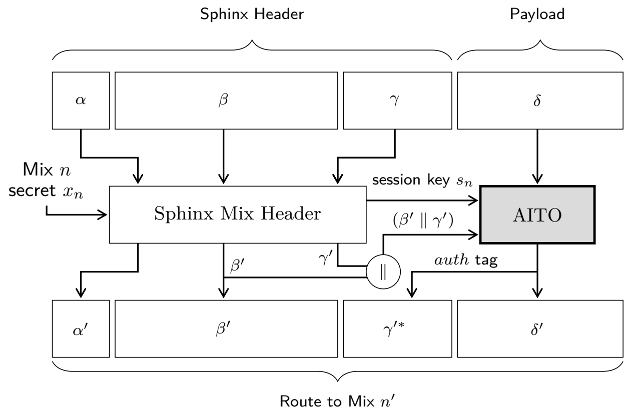

Fig. 3. Using AITO on Sphinx to process a Sphinx message $((\alpha, \beta, \gamma), \delta)$ into $((\alpha', \beta', \gamma'^{*}), \delta)$ at Mix n with the secret x_n. Note that the key derivation from $s_n = \mathsf{keyderive}(\alpha, x_n)$, blinding $\alpha_n = \mathsf{blind}(\alpha, s_n)$, and $(\beta, \gamma) \rightarrow (\beta', \gamma')$ computations occur as in Sphinx original version (see. Figure 3 from [21]). The session key s_n is used as the key, whereas the header (β', γ') is used as metadata on the input of AITO.

As a bonus, Sphinx using AITO *additionally* authenticates the payload for free, a feature that was missing in the original Sphinx.

6.3 Microblogging

The text input size limitation on microblogging systems, such as Twitter or Instagram, makes AITO a particularly suitable solution for sharing secret messages with a high level of security (significantly higher than ordinary encryption modes). We acknowledge that sharing secret messages on these systems may not be ideal for the broadcast messaging setting, however it is useful for sharing private messages among a segregated audience.

The high level idea of applying AITO to microblogging is as follows:

- The time stamp of a tweet functions as the nonce *nonce*;
- The hashtags (or part of them) function as a public meta data *meta*;
- The tweet itself (and some hashtags, if needed) are authenticated and encrypted.

We denote the time stamp by *ts* and the hashtags by *ht*. Due to the Twitter limitation on 140 characters, we take $\lambda_{\max} = 1120$ bits, but we can abuse the fact that the time stamp is implicit in the tweet. In other words, we require that $\nu + \alpha + \tau \leq 1120$. We remark that the Unix time stamp is conventionally written in 32 bits. This makes all three constructions of AITO suitable for the application.

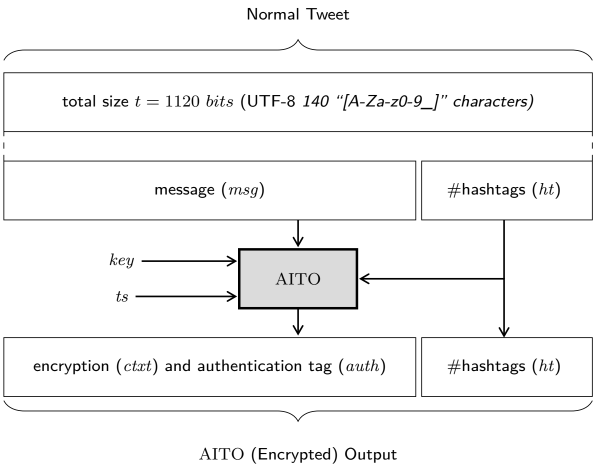
Normal Tweet
total size $t = 1120$ $bits$ (UTF-8 *140 "[A-Za-z0-9_]" characters)*
message (msg)
#hashtags (ht)
key
ts
AITO
encryption ($ctxt$) and authentication tag ($auth$)
#hashtags (ht)
AITO (Encrypted) Output

Fig. 4. Using AITO for microblogging to share private messages, while allowing filtering and providing limited data utility through the hashtags.

Without loss of generality, we consider users to be registered, use, and share private information on any Twitter-like systems. We also assume that users share a symmetric key using auxiliary out-of-band channels. For example, by performing an authenticated Diffie-Hellman or by simply encrypting the secret with the public keys of the intended recipients. However, this is beyond the scope of AITO mechanism.

For a specific instantiation, we suggest the framework of Figure 4. AITO operates by parsing a 1120-bit tweet into a message/hashtag-tuple $(msg, ht) \in \{0,1\}^{\leq \mu} \times \{0,1\}^{\leq \tau}$. Using the time stamp $ts \in \{0,1\}^{\sigma}$, it is then encrypted to a ciphertext $ctxt \in \{0,1\}^{\leq \nu}$ and authenticated using tag $auth \in \{0,1\}^{\alpha}$ (absent if no authentication is needed, in which case we have $\alpha = 0$). Thus, the encrypted tweet is of the form $(ctxt, auth, ht) \in \{0,1\}^{\leq \nu} \times \{0,1\}^{\alpha} \times \{0,1\}^{\leq \tau}$ (the time stamp *ts* is implicit from the tweet). In this configuration, nonce respecting behavior of users corresponds to them being "time stamp respecting," and we assume that two tweets under the same secret key are never encrypted with the same time stamp.

Moreover, other privacy-friendly architectures have been suggested to replace existing platforms. Hummingbird [19] presents a variant of Twitter that provably guarantees confidentiality of tweet contents, hashtags, and follower interests. Hummingbird bases its design on private set intersection methods [18] to match the authorized followers to the private tweets, and uses blockcipher to protect content.

7 Practical Analysis

In this section we discuss the practical result of AITO, with respect to its performance and possible implementation. In order to demonstrate the compact design, effieciency, and filtering of content through hashtags, we have implemented a proof-of-concept of the microblogging use case on Twitter.

7.1 Performance

Table 1 gives a summary of the performances of the three instantiations of AITO in the context of microblogging. In all cases, we take $\lambda_{\max} = 1120$ and $\sigma = |ts| \leq 64$, and for simplicity, this summary omits authentication (hence $\alpha = |auth| = 0$). We see that the three options for AITO are mutually competitive, but all of them significantly outperform a classic mode like AES-CBC. This is caused by that fact that AES only has an 128-bit state, and hence a mode of operation has to be placed on top. Consequently, such a scheme on the one hand requires no less than 8 blockcipher invocations, and on the other hand achieves only 128-bit security (and distinguishing attacks can be performed in a complexity of about 2^{64} [7]).

7.2 Implementation

To demonstrate the viability of our proposal, we implemented a proof-of-concept prototype AITO-App as a Firefox extension.[5] The current prototype is compatible with Firefox 14+, but it could be easily ported to other browsers extensions, e.g., to Chrome, as it is written in simple Javascript. Specifically, the `Enc` and `Dec` operations are as follows:

`Enc`: The user selects the Twitter text input area, and the extension launches a dialog where the user inserts the secret message and the public hashtags. The extension encrypts using the message, the hashtag, with the server time stamp, and the key as input and publishes the result into the Twitter text area.

`Dec`: The Firefox extension parses the messages on the Twitter-feed, and for each message transparently decrypts and replaces the result with the secret message.

AITO. The cryptographic module of AITO-App comprises the Javascript implementation of the AITO cipher designs based on Threefish, allowing easy portability to other browser implementations, e.g., Chrome. However, in order to increase performance, the AITO cryptographic module can also be used, interacting with Firefox through a local socket connection. In particular, the different instantiations of AITO were implemented using the available C libraries for Threefish and Keccak: Skein3Fish[6] and KeccakCodePackage,[7] respectively. The performances comply with the data from Table 1.

For the sake of simplicity, the AITO implementation uses a passphrase as input to a key derivation function to generate the secret key, whereas the exchange of the key is assumed to be performed using a secure and authenticated offline channel. However, the secret key distribution could be made by posting a QR code image containing the encryption of the secret key using the public key of the intended recipients. In addition, while it only supports desktop browsers at the moment, AITO-App is perfectly suitable for resource-constrained devices, such as smartphones. This is crucial considering that a significant portion of users access Twitter via their mobile devices. Even the somewhat increased size of the key will not be a problem in these settings.

Encoding of Messages. As aforementioned, Internet services like Twitter use UTF-8 to encode shared messages. Thus, the bit length of a 140 character message is not constant, as the representation of a UTF-8 encoded character can be range between 1–4 bytes.[8] However, if only ASCII characters are used, then UTF-8 encoding will only take one byte per character, and we get the $8 \times 140 = 1120$ bits that is used as a starting point of our constructions.

One problem with UTF-8 encoding is that after encryption, when we have an arbitrary sequence of bits, it is likely that it is no longer valid UTF-8 and thus we need to do some encoding for the ciphertext. In our implementation we have chosen to encode two bytes (16 bits) of the ciphertext into a single UTF-8 character and thus get more than enough space for any expansion

[5] Source of our implementations is freely available upon request.

[6] https://github.com/wernerd/Skein3Fish

[7] https://github.com/gvanas/KeccakCodePackage

[8] See for example unicode.org/faq/utf_bom.html

Table 1. Comparison of the different versions of AITO without authentication, compared with AES-CBC. A size is "flex" if it can take any value up to the trivial upper bounds ($|ctxt| + |ht| \leq 1120$ bits and $|ts| \leq 64$ bits). The size of the key is omitted, as it is always possible to take a key of size at least the security bound

	Size of		Efficiency		Security
	$ctxt$	$ht \| ts$	primitive	c/b	bits
AES-CBC	flex	flex	8·AES	16.0	64
3fish	1024	<128	1·3fish	6.5	512
LRW[3fish]	1024	flex	2·3fish	13	512
$\pi_{\text{keccak}}, \ell, n$	$\leq \ell \cdot n$	$<n$	$\ell \cdot \pi_{\text{keccak}}$	ℓ·12.5	$\frac{1600-n}{2}$
$1, 1024$	≤ 1024	<1024	$1 \cdot \pi_{\text{keccak}}$	12.5	288
$1, 800$	≤ 800	<800	$1 \cdot \pi_{\text{keccak}}$	12.5	400
$2, 576$	flex	<576	$2 \cdot \pi_{\text{keccak}}$	25	512

and/or overhead that the encryption might induce. The encoding works by simply mapping the two bytes into the Unicode character table[9] and encoding this character as valid UTF-8.

Traditional encoding schemes for encrypted data are base64 and base56, but these schemes' overhead will not allow encrypted and encoded tweets to be represented in only single tweets as the textual output is much longer than 140 characters if we have 1120 bits (or more) of ciphertext. With our encoding, it is possible to have the ciphertext in a single tweet, although the actual binary length of our encoding can add more overhead than the traditional schemes in some cases. This is of course specific to Twitter and UTF-8 encoding and in other possible applications the encoding can be done in other ways that are most suitable to that OSN.

8 Discussion and Conclusions

This paper presented AITO, a novel family of compact authenticated encryption (AE) schemes, that employs tweakable blockciphers and the Sponge permutation to achieve provable security that is greater than with existing mechanisms and systems, and naive constructions from normal blockciphers.

We also gave three use cases for this new scheme. First, we discussed the use of AITO in direct messaging applications and protocols such as OTR, where it can be used to offer AE for short messages. Second, AITO can be utilized in secure mix networks, especially in the Sphinx mix network, to provide better security and efficiency for the cryptographic headers required for the mixing. Finally, AITO can be used to offer AE for short messages on OSNs and microblogging services, such as Twitter and Instagram. To this means, we also demonstrated through a proof-of-concept prototype for the final use case on Twitter the practicality of AITO.

Although AE is possible with existing general blockciphers, such as AES-128, these solutions do not provide similar level of security and efficiency as AITO when applied to compact data. This is also demonstrated in Table 1: security of the AITO system goes up to approximately 512 bits, while encrypting more efficiently than, for instance, AES-CBC.[10]

It is worth noting that AITO is proven secure against nonce respecting adversaries. For our application to microblogging, this translates to time stamp respecting adversaries. In the case of protection against time stamp misuse, for example by the OSN itself, the AITO can accommodate a client generated time stamp. This would add some overhead (of 32 bits), but could protect against this type of attack. For the two Threefish based constructions the security of the schemes against time stamp misusing adversaries is achieved with a slight security degradation, the proof of which is similar to the ones in Section 3 and Section 4. For the Sponge based construction, time stamp misuse resistance cannot be achieved as the adversary can learn the XOR of two ciphertexts. Adding a client generated explicit time stamp would make the use of the plain Threefish construction harder due to the limited size of the tweak

[9] See for example http://unicode-table.com/en/

[10] The c/b's in the table are in fact derived from the speed of the Keccak permutation, AES-CBC, and 3fish on an Intel Core 2 for long messages [12, 20, 25]. These are included in the table for comparison.

space, but the other two options would still be viable. In any case, there are many incentives for a platform such as Twitter to remain honest, such as bad publicity.

As our work demonstrates, it is possible to achieve increased security for AE in compact messages with novel designs such as AITO. Topics for future research would be to find more applications for this type of AE schemes as well as potentially new AE schemes for compact messages besides AITO.

References

[1] Anderson, R. J., and Biham, E. Two practical and provably secure block ciphers: BEARS and LION. In *FSE '96* (1996), vol. 1039 of *LNCS*, Springer, pp. 113–120.

[2] Andreeva, E., Bilgin, B., Bogdanov, A., Luykx, A., Mendel, F., Mennink, B., Mouha, N., Wang, Q., and Yasuda, K. PRIMATEs v1, 2014. Submission to CAESAR competition.

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

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

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

[6] Aumasson, J., Jovanovic, P., and Neves, S. NORX v1, 2014. Submission to CAESAR competition.

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

[8] Bellare, M., Goldreich, O., and Mityagin, A. The power of verification queries in message authentication and authenticated encryption. Cryptology ePrint Archive, Report 2004/309, 2004.

[9] Bellare, M., Rogaway, P., and Wagner, D. The EAX mode of operation. In *FSE 2004* (2004), vol. 3017 of *LNCS*, Springer, pp. 389–407.

[10] Bertoni, G., Daemen, J., Peeters, M., and Van Assche, G. On the security of the keyed Sponge construction. Symmetric Key Encryption Workshop (SKEW 2011), 2011.

[11] Bertoni, G., Daemen, J., Peeters, M., and Van Assche, G. Duplexing the Sponge: Single-pass authenticated encryption and other applications. In *SAC 2011* (2012), vol. 7118 of *LNCS*, Springer, pp. 320–337.

[12] Bertoni, G., Daemen, J., Peeters, M., and Van Assche, G. Keccak implementation overview, May 2012. http://keccak.noekeon.org/Keccak-implementation-3.2.pdf.

[13] Bertoni, G., Daemen, J., Peeters, M., and Van Assche, G. Sponge functions, ECRYPT Hash Function Workshop 2007. http://sponge.noekeon.org/SpongeFunctions.pdf.

[14] Bertoni, G., Daemen, J., Peeters, M., Van Assche, G., and Van Keer, R. Keyak v1, 2014. Submission to CAESAR competition.

[15] Bilogrevic, I., Freudiger, J., Cristofaro, E. D., and Uzun, E. What's the gist? privacy-preserving aggregation of user profiles. In *ESORICS 2014* (2014), vol. 8713 of *LNCS*, Springer, pp. 128–145.

[16] Borisov, N., Goldberg, I., and Brewer, E. A. Off-the-record communication, or, why not to use PGP. In *WPES 2004* (2004), ACM, pp. 77–84.

[17] CAESAR: Competition for Authenticated Encryption: Security, Applicability, and Robustness, March 2014. http://competitions.cr.yp.to/caesar.html.

[18] Cristofaro, E. D., Kim, J., and Tsudik, G. Linear-complexity private set intersection protocols secure in malicious model. In *ASIACRYPT 2010* (2010), vol. 6477 of *LNCS*, Springer, pp. 213–231.

[19] Cristofaro, E. D., Soriente, C., Tsudik, G., and Williams, A. Hummingbird: Privacy at the time of Twitter. In *IEEE SP 2012* (2012), IEEE Computer Society, pp. 285–299.

[20] Crypto++ 5.6.0 Benchmarks, May 2015. http://www.cryptopp.com/benchmarks.html.

[21] Danezis, G., and Goldberg, I. Sphinx: A compact and provably secure mix format. In *IEEE S 2009* (2009), IEEE Computer Society, pp. 269–282.

[22] Datta, N., and Nandi, M. ELmE: A misuse resistant parallel authenticated encryption. In *ACISP 2014* (2014), vol. 8544 of *LNCS*, Springer, pp. 306–321.

[23] De Cannière, C., and Rechberger, C. Finding SHA-1 characteristics: General results and applications. In *ASIACRYPT 2006* (2006), vol. 4284 of *LNCS*, Springer, pp. 1–20.

[24] Dobraunig, C., Eichlseder, M., Mendel, F., and Schläffer, M. Ascon v1, 2014. Submission to CAESAR competition.

[25] Ferguson, N., Lucks, S., Schneier, B., Whiting, D., Bellare, M., Kohno, T., Callas, J., and Walker, J. The Skein hash function family, 2010. Submission to NIST's SHA-3 competition.

[26] Fluhrer, S. R. Cryptanalysis of the SEAL 3.0 pseudorandom function family. In *FSE 2001* (2002), vol. 2355 of *LNCS*, Springer, pp. 135–143.

[27] Gligoroski, D., Mihajloska, H., Samardjiska, S., Jacobsen, H., El-Hadedy, M., and Jensen, R. π-Cipher v1, 2014. Submission to CAESAR competition.

[28] Hoang, V., Krovetz, T., and Rogaway, P. Robust authenticated-encryption AEZ and the problem that it solves. In *EUROCRYPT 2015, Part I* (2015), vol. 9056 of *LNCS*, Springer, pp. 15–44.

[29] Jovanovic, P., Luykx, A., and Mennink, B. Beyond $2^{c/2}$ security in sponge-based authenticated encryption modes. In *ASIACRYPT 2014, Part I* (2014), vol. 8873 of *LNCS*, Springer, pp. 85–104.

[30] Karpman, P., Peyrin, T., and Stevens, M. Practical free-start collision attacks on 76-step SHA-1. In *CRYPTO 2015, Part I* (2015), vol. 9215 of *LNCS*, Springer, pp. 623–642.

[31] Krovetz, T., and Rogaway, P. The software performance of authenticated-encryption modes. In *FSE 2011* (2011), vol. 6733 of *LNCS*, Springer, pp. 306–327.

[32] Liskov, M., Rivest, R. L., and Wagner, D. Tweakable block ciphers. In *CRYPTO 2002* (2002), vol. 2442 of *LNCS*, Springer, pp. 31–46.

[33] Mennink, B., Reyhanitabar, R., and Vizár, D. Security of full-state keyed and duplex sponge: Applications to authenticated encryption. In *ASIACRYPT 2015, Part II* (2015), vol. 9453 of *LNCS*, Springer, pp. 465–489.

[34] Minematsu, K. Parallelizable rate-1 authenticated encryption from pseudorandom functions. In *EUROCRYPT 2014* (2014), vol. 8441 of *LNCS*, Springer, pp. 275–292.

[35] Morawiecki, P., Gaj, K., Homsirikamol, E., Matusiewicz, K., Pieprzyk, J., Rogawski, M., Srebrny, M., and Wójcik, M. ICEPOLE v1, 2014. Submission to CAESAR competition.

[36] Peter Gaži, Pietrzak, K., and Tessaro, S. The exact PRF security of truncation: Tight bounds for keyed sponges and truncated CBC. In *CRYPTO 2015, Part I* (2015), vol. 9215 of *LNCS*, Springer, pp. 368–387.

[37] Rogaway, P. Efficient instantiations of tweakable blockciphers and refinements to modes OCB and PMAC. In *ASIACRYPT 2004* (2004), vol. 3329 of *LNCS*, Springer, pp. 16–31.

[38] Rogaway, P., Bellare, M., Black, J., and Krovetz, T. OCB: a block-cipher mode of operation for efficient authenticated encryption. In *ACM Conference on Computer and Communications Security* (2001), ACM, pp. 196–205.

[39] Saarinen, M. STRIBOB r1, 2014. Submission to CAESAR competition.

[40] Stevens, M. New collision attacks on SHA-1 based on optimal joint local-collision analysis. In *EUROCRYPT 2013* (2013), vol. 7881 of *LNCS*, Springer, pp. 245–261.

[41] Stevens, M., Karpman, P., and Peyrin, T. Freestart collision on full SHA-1. Cryptology ePrint Archive, Report 2015/967, 2015.

[42] Van Den Hooff, J., Lazar, D., Zaharia, M., and Zeldovich, N. Vuvuzela: Scalable private messaging resistant to traffic analysis. In *Proceedings of the 25th Symposium on Operating Systems Principles* (2015), ACM, pp. 137–152.

[43] Wang, X., Yin, Y., and Yu, H. Finding collisions in the full SHA-1. In *CRYPTO 2005* (2005), vol. 3621 of *LNCS*, Springer, pp. 17–36.