Filipe Beato*, Kimmo Halunen*, and Bart Mennink*

Tweety: Tweet Secrets Efficiently

Abstract: Online social networks such as Twitter have users in the hundreds of millions, providing easy and reliable channels for dissemination of information. Twitter plays a fundamental role in quick dissemination of information as well as coordinating and amplifying grassroots movements, as demonstrated by recent Arab spring events. At the same time, those networks store treasure troves of information to support their economical business model through targeted advertisements, and have become prime targets of censorship and surveillance based on the content shared. In this paper, we propose Tweety, a novel and secure system for privacy in Twitter, tailored for usage in social media websites. Tweety guarantees semantic security while complying with the commonly imposed space restrictions and allowing limited targeted advertisement. After formalizing Tweety we present three distinct designs based on different primitives, tweakable blockciphers and Sponge functions, that offer higher levels of security than existing alternatives. Furthermore, we implemented a prototype demonstrating the improved efficiency and low overhead of our constructions, mainly related to the increased key sizes for better security.

Keywords: Privacy, social media, tweakable blockciphers, Sponge.

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) 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, providers end up centralizing and storing large amounts of information about the users and their communications, thus leading to several privacy threats. Most popular OSNs, such as Facebook, Google+, and Twitter, provide users with certain privacy controls over their content and with secured direct messaging. However, this process relies not only on the diligence of the users but also on the trustworthiness of the providers in protecting stored content from possible adversaries. Nevertheless, providers require access to the content in order to comply with their economical model, such as targeted advertisement [21, 23], consequently exposing users to several privacy issues. This has lead to several reports of mass breaches on the information shared on OSNs [11], as is evident from the recent accounts of surveillance programs like Prism [22], and the recent iCloud mega leak [18], amplifying the privacy issues.

These issues motivate the need to implement more reliable user-centric privacy protection mechanisms, such as end-to-end encryption. One possible solution is to encrypt all the data shared and transferred, for instance using a standard encryption modes of operation, such as AES-CTR or AES-CBC. However, those constructions have several shortcomings. For instance, sharing encrypted information breaks the business model 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. Also, those general constructions do not take into account the limitations and specific properties of OSN, such as the space constraints (140 characters, about 1120 *bits*) and the use of hashtags for filtering and marking specific events, in the case of Twitter and Instagram. For instance, an adaption of the CTR or CBC mode of operation with AES-128[1] to those properties, achieves 128-bit security *at most*, and even allows for distinguishability attacks in complexity about 2^{64} (cf. Bellare et al. [5]), while requiring at least 8 AES evaluations. Although direct messaging services apply similar mechanisms to protect the communication among users,

***Corresponding Author: Filipe Beato:** ESAT/COSIC, KU Leuven and iMinds, Leuven, Belgium, E-mail: filipe.beato@kuleuven.be
***Corresponding Author: Kimmo Halunen:** VTT Technical Research Center of Finland, Oulu, Finland, E-mail: kimmo.halunen@vtt.fi
***Corresponding Author: Bart Mennink:** ESAT/COSIC, KU Leuven and iMinds, Leuven, Belgium, E-mail: bart.mennink@kuleuven.be

1 http://codecereal.blogspot.ch/2011/06/encrypted-tweets.html

these are generally inefficient and often require keys to be managed by providers.

In this paper we propose *Tweety*, a novel privacy protection mechanism tailored for Twitter-like OSNs, that selectively reveals parts of your shared content while keeping the majority of the data available only to selected recipients. It additionally allows for authenticity, ensuring integrity of data. Tweety employs recent advances in tweakable blockciphers and Sponge functions 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. Furthermore, our system helps to save some computational resources at the recipient side of the communication as it is possible to decide whether or not to decrypt the message based on the hashtags. For instance, one could choose not to decrypt messages with the hashtag "#football" from a certain (or every) user. This could provide energy saving benefits with some resource constrained devices, while enabling providers to use this information to support their economical business model.

The paper is organized as follows: Section 2 introduces the required notation, and formally describes Tweety and the assumed threat model. After describing in Section 3 the basic approach of Tweety based on Threefish [15], Section 4 extends to allow expanded tweak space. Section 5 presents another version of Tweety based on Sponge functions [10]. Then, we describe our proof-of-concept implementation in Section 6, while Section 7 reviews related work. Finally, Section 8 discusses the results and concludes the paper.

2 Model

This section introduces the required notation, Tweety, and the respective threat model. Without loss of generality, we consider users to be registered, use, and share private information on Twitter. We also assume that users share a symmetric key using auxiliary out-of-band channels.

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 $M\|N$, and if M and N are of the same size, $M \oplus N$ 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 $N = \lceil N \rceil_m \| \lfloor N \rfloor_{n-m}$.

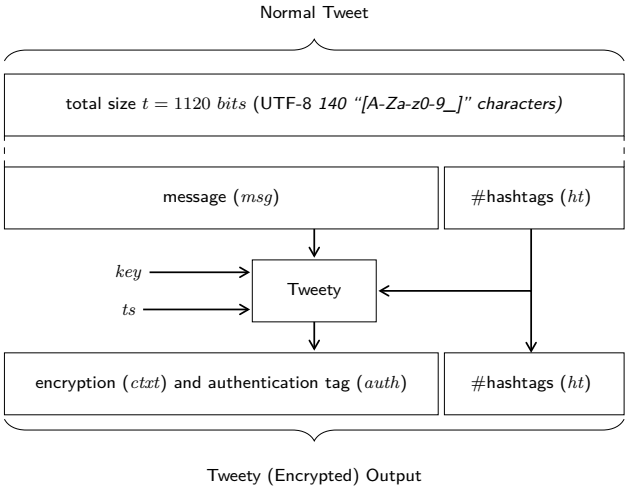

Fig. 1. Tweety design overview

2.2 Tweety

Tweety allows users to encrypt and authenticate content of the tweets while leaving (some) hashtags public. These public hashtags serve a twofold purpose: information for targeted advertisements, and to allow users to enforce selective control of tweets. Figure 1 illustrates the general design of Tweety. The key principle of Tweety is inspired by tweakable blockciphers, where the public hashtags and also the time stamp of the tweet act as the tweak. Conversely, Tweety operates by parsing a 1120-bit tweet into a message/hashtag-tuple $(msg, ht) \in \{0,1\}^{\leq \mu} \times \{0,1\}^{\leq \tau}$, behaving then like a symmetric encryption scheme: using a secret key $key \in \{0,1\}^{\kappa}$, the message msg is encrypted to a ciphertext $ctxt \in \{0,1\}^{\leq \nu}$, where ht and time stamp $ts \in \{0,1\}^{\sigma}$ function as the tweak. It additionally outputs an authentication 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). We assume that

two tweets under the same secret key are never made with the same time stamp.

Note that we allow for a small amount of ciphertext expansion (from μ to ν bits), as long as $\mu + \tau \leq \nu + \alpha + \tau \leq 1120$. Inspired by the Unix time stamping, the size of the time stamp, σ, is considered to be between 32 and 64 bits. In more detail, the Unix time stamp is conventionally written in 32 bits. This is enough until January 19, 2038, before which all systems should be migrated to 64-bit stamps.

More formally, we consider Tweety to be composed of three algorithms: KeyGen, PTweet, and PRead. KeyGen is a randomized algorithm that gets as input $\kappa \in \mathbb{N}$ and outputs a random key $key \leftarrow \{0,1\}^\kappa$, whereas the PTweet and PRead algorithms are defined as follows:

PTweet:

Input: $(key, msg, ht, ts) \in$
$\{0,1\}^\kappa \times \{0,1\}^{\leq\mu} \times \{0,1\}^{\leq\tau} \times \{0,1\}^\sigma$,

Output: $(ctxt, auth, ht) \in$
$\{0,1\}^{\leq\nu} \times \{0,1\}^\alpha \times \{0,1\}^{\leq\tau}$,

PRead:

Input: $(key, ctxt, auth, ht, ts) \in$
$\{0,1\}^\kappa \times \{0,1\}^{\leq\nu} \times \{0,1\}^\alpha \times \{0,1\}^{\leq\tau} \times \{0,1\}^\sigma$,

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

PRead outputs the unique msg that satisfies $\texttt{PTweet}(key, msg, ht, ts) = (ctxt, auth, ht)$, or it returns $\perp$ if no such message exists. The output of ht by PTweet will sometimes be left out and considered to be implicit.

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 encrypted tweets under a secret and unknown key itself. In this case, $\mathcal{A}$ should not learn the encrypted content, beyond that revealed in the hashtags.

More technically, we consider adversary $\mathcal{A}$ that has query access to the encryption functionality PTweet 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 time stamp respecting, meaning that every query must be made under a different time stamp (see also Section 8).[2] 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 Tweety is defined as follows:

$$\mathbf{Adv}^{\text{cpa}}_{\text{Tweety}}(\mathcal{A}) = \left| \Pr\left(key \xleftarrow{\$} \texttt{KeyGen}(\kappa) \ : \ \mathcal{A}^{\texttt{PTweet}_{key}} = 1\right) - \Pr\left(\$ \xleftarrow{\$} \mathsf{Func}(\texttt{PTweet}_{key}) \ : \ \mathcal{A}^{\$} = 1\right) \right| .$$

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

For the authenticity of Tweety, we consider $\mathcal{A}$ to have access to the encryption functionality PTweet under a secret key key, and we say that $\mathcal{A}$ *forges* an encrypted tweet if it manages to output a tuple $(ctxt, auth, ht, ts) \in \{0,1\}^{\leq\nu} \times \{0,1\}^\alpha \times \{0,1\}^{\leq\tau} \times \{0,1\}^\sigma$ such that $\texttt{PRead}(key, ctxt, auth, ht, ts) = msg \neq \perp$ and (msg, ht, ts) was never queried to PTweet before. Note that the forgery attempt may be made under a time stamp ts that has appeared before. The advantage of $\mathcal{A}$ in breaking the authenticity of Tweety is defined as follows:

$$\mathbf{Adv}^{\text{auth}}_{\text{Tweety}}(\mathcal{A}) = \Pr\left(key \xleftarrow{\$} \texttt{KeyGen}(\kappa) \ : \ \mathcal{A}^{\texttt{PTweet}_{key}} \text{ forges}\right) .$$

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

3 Tweety: 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 Tweety, we suggest using *Threefish*, a tweakable blockcipher by Ferguson et al. used for the Skein hash function family [15]. 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

2 More formally, we assume that two tweets are never posted under the same key at the exact same time. This is reasonable if the key is used by a relatively small set of users.

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 to encrypt tweets directly, a construction which we dub Tweety$^{3\text{fish}}$. 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 time stamp and hashtag 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. Note that if $\sigma = 32$, these parameters satisfy $\mu + \tau \leq \nu + \alpha + \tau \leq 1120$. At a high level, the encryption consists of putting $m = \mathsf{pad}_{1024}(msg)$ and $t = \mathsf{pad}_{128}(ts\|ht)$, and the ciphertext and authentication tag are derived as $ctxt\|auth = c$. Formally, the encryption and decryption of Tweety$^{3\text{fish}}$ are defined as in Algorithms 1 and 2.

Algorithm 1 $\mathtt{PTweet}^{3\text{fish}}$

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

Algorithm 2 $\mathtt{PRead}^{3\text{fish}}$

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

Security. In this section we formally derive the security under the assumption that 3fish is a secure tweakable blockcipher, so that the security of Tweety$^{3\text{fish}}$ 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

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

By $\mathbf{Adv}^{\widetilde{\text{sprp}}}_{\widetilde{E}}(Q,T)$ 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*

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

where $T' \approx T$.

Proof. We start with the secrecy of Tweety$^{3\text{fish}}$. Let $\mathcal{A}$ be an adversary that makes Q queries and runs in time T. It has access to either $\mathtt{PTweet}_{key}$ or \$. Note that Q evaluations of Tweety$^{3\text{fish}}$ 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 $\mathtt{PTweet}^{\widetilde{\pi}}(key, msg, ht, ts)$ is responded with

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

As $\mathcal{A}$ is required to be time stamp respecting, every query is made under a new time stamp, which means that every query initiates a new instance of $\widetilde{\pi}$, and $ctxt\|auth$ is a random 1024-bit value. This means that $\mathtt{PTweet}^{\widetilde{\pi}}_{key}$ 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 Tweety$^{3\text{fish}}$ induce $Q + R$ evaluations of 3fish. It remains to consider the probability to forge an authentication tag for Tweety$^{\widetilde{\pi}}$. By [**?**], it suffices to consider any attempt and sum over all R attempts. Consider any forgery attempt $(ctxt, auth, ht, ts)$. Note that, as $\mathcal{A}$ is required to be time stamp respecting, there has been at most one encryption query under hashtag ht and time stamp ts. Therefore, the value

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

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 [**?**].

4 Tweety: Expanded Tweak Space

The Tweety basic construction presents a rather small tweak space, and the usage of a larger time stamp results in a limitation on the size of the hashtag. A way to resolve this is to employ a random oracle that maps the time stamp and (larger) hashtag 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. [19] 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:[3]

LRW[3fish]:
Input: $(k, t, t', m) \in \{0,1\}^{1024} \times \{0,1\}^{1024} \times \{0,1\}^{128} \times \{0,1\}^{1024}$,
Output: $3\text{fish}(k, t', 3\text{fish}(k, t', m) \oplus t) \in \{0,1\}^{1024}$.

This construction can be used to realize $\text{Tweety}^{\text{LRW[3fish]}}$ as illustrated in Figure 2 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 the hashtags should now be of size at most $\tau \leq 1023$, a condition clearly satisfied as $1120 \geq \nu + \alpha + \tau = 1024 + \tau$.

Algorithm 3 $\text{PTweet}^{\text{LRW[3fish]}}$

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

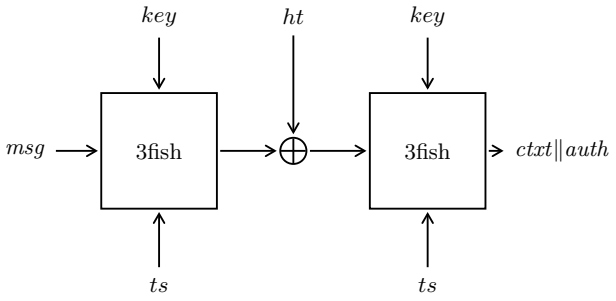

Fig. 2. Tweety based on LRW[3fish]. Padding of data is excluded from the figure

Algorithm 4 $\text{PRead}^{\text{LRW[3fish]}}$

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

Security. The security of $\text{Tweety}^{\text{LRW[3fish]}}$ in fact follows from Theorem 1 and a result from [19].

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

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

$$\mathbf{Adv}^{\text{auth}}_{\text{Tweety}^{\text{LRW[3fish]}}}(Q,R,T) \leq \Theta\left(\frac{(Q+R)^2}{2^n}\right) + \mathbf{Adv}^{\widetilde{\text{sprp}}}_{\text{3fish}}(2(Q+R),T') + \frac{R2^{n-\alpha}}{2^n-1},$$

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

$$\mathbf{Adv}^{\text{cpa}}_{\text{Tweety}^{\text{LRW[3fish]}}}(Q,T) \leq \mathbf{Adv}^{\widetilde{\text{sprp}}}_{\text{LRW[3fish]}}(Q,T''),$$

$$\mathbf{Adv}^{\text{auth}}_{\text{Tweety}^{\text{LRW[3fish]}}}(Q,R,T) \leq \mathbf{Adv}^{\widetilde{\text{sprp}}}_{\text{LRW[3fish]}}(Q+R,T'') + \frac{R2^{n-\alpha}}{2^n-1},$$

where $T'' \approx T$. In [19] it is proven that

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

where $T' \approx T''$. □

3 The other construction is less relevant as it requires an additional key and as it needs a universal hash function with a 1024-bit range (or smaller, in which case the security of the construction degrades).

5 Tweety: Sponge Construction

The Sponge functions were originally introduced by Bertoni et al. [10] for cryptographic hashing, but can also be used in a broad spectrum of keyed applications, including message authentication [1, 7**?**] and stream encryption [8**?**]. They are also particularly useful for tweet encryption given the tweet size and the flexibility in the state size of the Sponge. In more detail, we suggest the following function realization, which resembles ideas of the full-state duplex mode [**?**], transformed to the tweakable setting, as depicted in Figure 3. We stress, however, that the keyed Sponges are merely stream-based encryption, and a unique time stamp is required for every encryption.

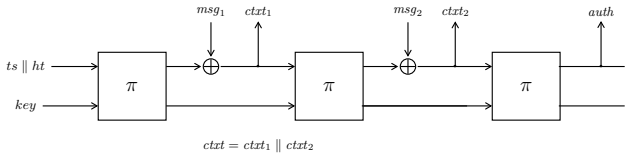

Fig. 3. Tweety based on a Sponge. Padding of data is excluded from the figure

The realization of Tweety$^{\pi,\ell,n}$ 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), and the sizes of the time stamp and hashtag 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 $\mu + \alpha + \tau \leq 1120$. The formal encryption and decryption functionalities are given in Algorithms 5 and 6.

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

Input: $(key, msg, ht, ts) \in \{0,1\}^{\kappa} \times \{0,1\}^{\leq \mu} \times \{0,1\}^{\leq \tau} \times \{0,1\}^{\sigma}$
Output: $(ctxt, auth) \in \{0,1\}^{\leq \mu} \times \{0,1\}^{\alpha}$

$\ell' \leftarrow \lceil (|msg| + 1)/n \rceil$
$msg_1 \parallel \cdots \parallel msg_{\ell'} \xleftarrow{n\text{-blocks}} \mathsf{pad}_{\ell' \cdot n}(msg)$
$s_0 \leftarrow \mathsf{pad}_n(ts \| ht) \parallel 0^{b-n-|key|} \parallel key$
for $i = 1, \ldots, \ell'$ **do**
  $s_i \leftarrow \pi(s_{i-1})$
  $s_i \leftarrow s_i \oplus \left(msg_i \parallel 0^*\right)$
  $ctxt_i \leftarrow \lceil s_i \rceil_n$
$s_{\ell'+1} \leftarrow \pi(s_{\ell'})$
return $(\lceil ctxt_1 \parallel \cdots \parallel ctxt_{\ell'} \rceil_{|msg|}, \lceil s_{\ell'+1} \rceil_{\alpha})$

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

Input: $(key, ctxt, auth, ht, ts) \in \{0,1\}^{\kappa} \times \{0,1\}^{\leq \mu} \times \{0,1\}^{\alpha} \times \{0,1\}^{\leq \tau} \times \{0,1\}^{\sigma}$
Output: $msg \in \{0,1\}^{\leq \mu}$ or $\perp$

$\ell' \leftarrow \lceil (|ctxt| + 1)/n \rceil$
$ctxt_1 \parallel \cdots \parallel ctxt_{\ell'} \xleftarrow{n\text{-blocks}} \mathsf{pad}_{\ell' \cdot n}(ctxt)$
$s_0 \leftarrow \mathsf{pad}_n(ts \| ht) \parallel 0^{b-n-|key|} \parallel key$
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. Tweety$^{\pi,\ell,n}$ is in fact a full-state duplex construction [**?**], 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. [1]. 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 $\mathtt{PTweet}^{\pi,\ell,n}$ and $\mathtt{PRead}^{\pi,\ell,n}$ 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*

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

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

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

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

where, abusing notation, $|msg| \bmod n \in \{1, \ldots, n\}$. In other words, any evaluation of $\mathtt{PTweet}_{key}$ 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 $\mathtt{PTweet}_{key}$, 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 time stamp 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 $\mathtt{PTweet}_{key}$ are made under a different *ts* as the adversary is time stamp respecting. Consequently, every query to $\mathtt{PTweet}_{key}$ is responded with a uniformly randomly generated $|msg|$-bit value, and thus,

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

Next, for authenticity, it suffices to only focus on the value *auth*. Consider any forgery attempt $(ctxt, auth, ht, ts)$. Let *msg* be the message that is derived by $\mathtt{PRead}^{\pi,\ell,n}$. As the forgery is required to be non-trivial, $\mathcal{RO}$ has never been queries on

$$\mathsf{pad}_n(ts\|ht)\|msg_1 \cdots msg_\ell$$

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

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

Now, in [1] it is proven that[4]

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

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 $\mathrm{Tweety}^{\pi_{\mathrm{keccak}},\ell,n}$ increases linearly in ℓ, we suggest to use $\ell \le 2$. If $\ell = 1$ and $n = 1024$, the messages can be of size at most 1024 bits, and $ts\|ht$ 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, $ts\|ht$ should be of size at most 575, and 512-bit security is achieved.

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

6 Implementation

To demonstrate the viability of our proposal, we implemented a proof-of-concept prototype Tweety-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 `PTweet` and `PRead` operations are as follows:

`PTweet`: The user selects Twitter text 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.

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

Tweety. The cryptographic module of Tweety-App comprises the Javascript implementation of the Tweety cipher designs based on Threefish, allowing easy portability to other browser implementations, e.g., Chrome. However, in order to increase the performance the Tweety cryptographic module can also be used, interacting with Firefox through a local socket connection In particular, the different instantiations of Tweety were implemented using the available C libraries for Threefish and Keccak: Skein3Fish[6] and KeccakCodePackage,[7] respectively. The performance of the three different instantiations comply with the values depicted in Table 1. Hence, Tweety-App demonstrates that the efficiency of the different Tweety approaches is much higher than other constructions, such as AES-CBC.

In addition, while it only supports desktop browsers at the moment, Tweety-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 mentioned earlier, Twitter uses UTF-8 to encode the messages. Thus, the bitlength of a 140 character message is not a constant as the representation of a UTF-8 encoded character can be anything 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 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.

7 Related Work

Along with the increased popularity of OSNs several privacy concerns start to arise which have prompted a large interest within the research community. As a result, several privacy-preserving solutions have been proposed to deliver OSN content confidentiality by means of existing cryptographic mechanisms. However, there is a lack of tailored designs that take into account the space limitation and the general design of OSNs. For instance, systems like FaceCloak [20] use blockciphers under different modes of operation such as AES-CBC to protect the content. Scramble [4] and FSEO [3] use a broadcast

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

Table 1. Comparison of the different versions of Tweety 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

encryption mechanism that encrypts the content using a blockcipher under a mode of operation. It attaches a header with the public-key encryptions of the key used by the blockcipher, representing the access control list of the content. Similarly, Persona [2] and EaSiER [17] use attribute-based encryption mechanisms to protect confidentiality while allowing access control definition by attributes.

Tweetcipher [16] was presented by a group of cryptographers as a compact authenticated encryption algorithm initiated through Twitter discussions. Although Tweetcipher is based on Sponge and Salsa20 [6] constructions, it aims at delivering authentication and requires six tweets, whereas Tweety requires only one tweet with easy re-keying and public hashtags.

Moreover, other privacy-friendly architectures have been suggested to replace existing platforms. Hummingbird [13] 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 [12] to match the authorized followers to the private tweets, and uses blockcipher to protect content.

In general, such constructions require large entropy which may not be allowed by OSNs like Twitter. Also those constructions demand at least 8 AES evaluations achieving only 64-bit indistinguishability security. Aligned with the fact that general tweets are of a relatively short (but larger than 128 bits) length, the dedicated constructions presented by the different Tweety approaches achieve higher efficiency results and a much higher level of security. This can be seen from Table 1. In addition, Tweety can be used together with the broadcast and attribute-based encryptions solutions as a tailored symmetric encryption, and with Hummingbird system for protecting tweets.

8 Discussion and Conclusions

This paper presented Tweety, a customized system for selectively revealing parts of your tweets publicly, while keeping the majority of information secret. We described different options for realizing this and compared their efficiency. The constructions employ tweakable blockciphers and Sponge permutations achieving provable security that is greater than existing systems or naive constructions from normal blockciphers.

Although encryption is possible with existing general blockciphers, such as AES-128, these solutions do not provide similar level of security and efficiency as Tweety when applied to the Twitter setting. This is also demonstrated in Table 1: security of the Tweety system goes up to approximately 512 bits, while encrypting more efficiently than, for instance, AES-CBC.[10] Furthermore, in order to fit more information than the common 1120 bits (i.e., 140 characters using Latin UTF-8 alphabet) into single tweets, one could employ bit packing schemes. For instance, by using larger character sets, such as emoji characters and Chinese characters, one can accommodate extra information as these characters are larger than 8 bits.

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 [9, 14, 15]. These are included in the table for comparison.

Tweety is proven secure against time stamp respecting adversaries. In the case of protection against time stamp misuse, for example by the OSN itself, the Tweety can accommodate a client generated time stamp. This would add some overhead (32 or 64 bits), but could protect against this type of attack. For the 3fish constructions the security for time stamp misusing adversaries is reduced and for the π_{keccak} construction the proof does not hold as the adversary can learn the XOR of two ciphertexts. Adding this client generated explicit time stamp would make the use of plain 3fish construction harder due to the limited size of the tweak space, but the other two options would still be viable. In any case, the time stamp misusing adversary will not directly learn the plaintext or the key used in encryption even if the proof of security does not hold. In addition, there are many incentives for the platform such as Twitter to remain honest such as bad publicity.

References

[1] Andreeva, E., Daemen, J., Mennink, B., Van Assche, G.: Security of keyed Sponge constructions using a modular proof approach. In: FSE 2015. LNCS (To appear), Springer (2015)

[2] Baden, R., Bender, A., Spring, N., Bhattacharjee, B., Starin, D.: Persona: an online social network with user-defined privacy. In: ACM SIGCOMM 2009. pp. 135–146. ACM (2009)

[3] Beato, F., Ion, I., Čapkun, S., Preneel, B., Langheinrich, M.: For some eyes only: protecting online information sharing. In: ACM CODASPY 2013. pp. 1–12. ACM (2013)

[4] Beato, F., Kohlweiss, M., Wouters, K.: Scramble! your social network data. In: PETS 2011. LNCS, vol. 6794, pp. 211–225. Springer (2011)

[5] 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)

[6] Bernstein, D.J.: Salsa20. eSTREAM, ECRYPT Stream Cipher Project, Report 25, 2005 (2005)

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

[8] 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)

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

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

[11] Bosker, B.: How Facebook Explains User Data Bug That Leaked 6 Million People's Information. In: The Huffington Post (Feb 2015), http://huff.to/1A4Y5NG, Accessed: Feb. 18, 2015

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

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

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

[15] 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

[16] Green, M., Barreto, P.S.L.M., Aumasson, J.P., Neves, S.: Tweetcipher! (Jun 2013), http://cybermashup.com/2013/06/12/tweetcipher-crypto-challenge/. Accessed: Jul 10, 2015

[17] Jahid, S., Mittal, P., Borisov, N.: EASiER: encryption-based access control in social networks with efficient revocation. In: ACM ASIACCS 2011. pp. 411–415. ACM (2011)

[18] Lewis, D.: iCloud Data Breach: Hacking And Celebrity Photos. In: Forbes Online (Sept 2014), http://onforb.es/1Cmngvl, Accessed: Jan. 6, 2015

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

[20] Luo, W., Xie, Q., Hengartner, U.: FaceCloak: An architecture for user privacy on social networking sites. In: IEEE CSE 2009. pp. 26–33. IEEE Computer Society (2009)

[21] Oremus, W.: Facebook sued for "reading" your private messages. In: Slate (Jan 2014), http://slate.me/1evQXN1, Accessed: Oct. 27, 2014

[22] Post, W.: NSA slides explain the PRISM data-collection program. In: Washington Post (Jun 2013), http://wapo.st/J2gkLY. Accessed: Dec 3, 2014

[23] Riederer, C., Erramilli, V., Chaintreau, A., Krishnamurthy, B., Rodriguez, P.: For sale : your data, By : you. In: HOTNETS 2011. p. 13. ACM (Nov 2011)