Hoot

Dustin Bachrach Christopher Nunu
Department of Computer Science
Rice University
Houston, Texas
{ahdustin,canunu}@gmail.com

Dan Wallach
Department of Computer Science
Rice University
Houston, Texas
dwallach@rice.edu

Matthew Wright
Department of Computer Science and Engineering
University of Texas at Arlington
Arlington, Texas
mwright@cse.uta.edu

ABSTRACT

FiXme: *TODO: This is an example of the hlfixme command. When we get down to the last 2-3 days, we'll set the fixme status from "draft" to "final," which will prevent compilation to pdf without fixing or removing all the fixmes.*

1. INTRODUCTION

- Problem: Twitter like semantics w/ encrypted messages - Follow a Hash tag - Take hash tag and create something with crypto strength - Something derived from tags you can search on - But also deliberate collisions (cover traffic)

- Like to have thing that feels like twitter but anonymity properties:

- Twitter/Facebook relevant in Tunisia, (Social media playing big role in revolution across many countries. govt deliberately shut down)

- While we cannot keep them from filtering out service altogether, want to have private communication in plain sight (not stenographic)

- Strong crypto usable by people whispering to each other in streets

- Only trusted channel is not electronic (spoken word), to exchange key.

- complimentary to Tor, solve problems Tor+Twitter does not

- What we are doing – Define a protocol for users to communicate over an insecure public network like twitter with message confidentiality and subscriber anonymity.

- Punting on plain tag distribution

- Vocabulary

2. BACKGROUND

1) History of government censorship, man in the middle - Tunisia code injection - Chinese firewall - Crypto keys for important services to iranian source (Komodo) - Person providing netwrok (even over ssl) might be evil

2) Tor - Trying to work against government censorship

- Group Crypto Keys

2.1 Definitions

Privacy for groups and individuals in those groups has been investigated in a variety of contexts. In this section, we describe how the aims of Hoot relate to those of previously studied systems.

Anonymity.

Systems for online anonymity, like Tor [3] and Mixminion [1], aim to protect users from being linked with their traffic.

In this context, we can consider a variety of privacy attributes, including:

- *sender anonymity*: The sender of a message is not identifiable from among a set of possible senders.
- *recipient anonymity*: The recipient of a message is not identifiable from among a set of possible recipient.
- *unlinkability*: Any of the items of interest (senders, recipients, or messages) cannot be linked with other items of interest.
- *pseudonymity*: The use of pseudonyms as identifiers, either for a sender or a recipient.

Pfitzmann and Hansen have compiled a rich discussion of the meaning of these and other terms [2]. Unlinkability is a rather broad term. Two useful terms that can be derived from it are *recipient unlinkability* — the recipient cannot be linked with the sender(s) and messages — and *relationship anonymity* — the sender and recipient cannot be linked with each other.

Hoot does not seek to provide sender anonymity: the adversary can observe the fact that a given sender sent a particular message. Against weaker adversaries who cannot eavesdrop on the sender, Hoot can provide pseudonymity. More importantly, Hoot does aim to provide recipient

It does

Anonymity: - sender anonymity - Unlinkability - Receiver anonymity - subscriber anonymity - Relationship anonymity - Group anonymity

Social network privacy - Community privacy - Adversarial community discovery

Data and database privacy

Anti-censorship mechanisms: - Eternity, Freehaven - Freenet, Gnunet - Publius, Tangler, Dagster - Perng - Serjantov

Metrics - k-anonymity - l-diversity - differential privacy - plausible deniability -

Techniques: - Cover traffic - Link cover - Path cover - RBC - PIR/OT

3. DESIGN

The Hoot protocol consists of a message header and a message body. The header contains the identifier, session key, digest key, and integrity digest. Messages are indirect in that they do not reference recipients. Anyone who knows the shared secret must be able to read the message, but should also be able to find the message easily. Rather than attempting to treat every message posted to Twitter as a potential group message, and thus decrypting the entire Twitter stream, the protocol places an identifier into every Hoot so a fellow group member can simply search for the identifier to see all potential messages. With a constant group identifier, readers can also publicly follow that identifier like any other Hash Tag on Twitter.

To create an identifier, we must find a short set of bits that are derivable from the shared secret, but give an attacker little information about the shared secret itself. Hash functions provide a great way to get a set of bits from a shared secret without divulging much information about the original shared secret. In our protocol, the shared secret is called a *Plain Tag*, which is comparable to a Twitter Hash Tag. The result of hashing the Plain Tag with a given hash function, *H*, is referred to as the *Long Tag*. Thus,

$$LongTag \leftarrow H(PlainTag) \tag{1}$$

The protocol could simply use the Long Tag as an identifier, but this choice leads to several problems. First, to uphold our design goal of being concise and to fit within Twitter's 140 character limit, it is infeasible to use the full output of a hash function. Secondly, strong hash functions do not produce many collisions. If a group is communicating in secret, not only do we want to protect the content of the communication, but we would like to conceal subscription to a particular group. For example, a rebellion group wishes to communicate over Twitter using Hoots, but it can be dangerous for a supporter of the rebellion to listen and subscribe to the identifier of the group. If, however, the identifier of the group can collide with the identifier of a popular Internet topic, like Charlie Sheen, group followers can shadow their rebellious activities with other innocent topics.

To generate a collision, we need to shorten the long tag, generating a *Short Tag*, which will induce more collisions. The shorter the Short Tag, the higher the collision rate will be and the less sure an observer can be of what topic a Hoot reader is listening to. With this greater anonymity comes more computational work, however. Since more group messages will now be belong to the same identifier, a follower must decrypt more messages to find relevant signal. Depending on the required degree of subscriber anonymity, more collisions might be worth the computational overhead. Also, even given a constant Short Tag length, a group can choose a tag that will collide with extremely popular tags to generate even more noise. Given a Long tag of byte-length *N*, we produce a Short Tag by taking the first *K* bytes of it:

$$ShortTag \leftarrow LongTag_{[0:K]} : K < N \tag{2}$$

The header also contains an set of encrypted keys, *(Session Key, Digest Key)*. To combat replay attacks, the protocol uses a random session key to encrypt the plain text. This session key is then included in the set of keys in the header. The protocol ensures integrity the standard way by including an HMAC of the cipher text (http://www.daemonology.net/blog/2009-06-24-encrypt-then-mac.html). The key to the HMAC function is randomly generated and is included in these set of keys:

$$SessionKey \leftarrow randomBytes() \tag{3}$$

$$DigestKey \leftarrow randomBytes() \tag{4}$$

To encrypt this set of keys, we exploit the entropy of the Long Tag. Since the identifier for a message is only the first *K* bytes in the Long Tag, we can use the latter set of bytes elsewhere. By encrypting the keys with part of the Long Tag, a reader can easily know the key to decrypt with since it is simply the hash of the Plain Tag, and we also do not require knowledge of two different secrets. Thus, given an encryption function and corresponding key, E_{key}, we have the encrypted keys as:

$$KeyPair \leftarrow E_{LongTag_{[J:N]}}(SessionKey.DigestKey) \quad \text{where } J > K \wedge J < N \tag{5}$$

The body of the Hoot is the cypher text, *C*, of a message, *M*, encrypted with the Session Key:

$$C \leftarrow E_{SessionKey}(M) \tag{6}$$

The header also contains the integrity digest, *D*, of the cypher text, using the Digest Key in the header. By including an integrity digest, the protocol allows for quick verification that a message is for a specific group without having to decrypt the message. It also verifies that the Hoot has not been tampered with.

$$D \leftarrow HMAC_{DigestKey}(C) \tag{7}$$

Therefore, a complete Hoot appears as:

$$Hoot \leftarrow ShortTag.KeyPair.D.C \tag{8}$$

4. IMPLEMENTATION

In this section we describe an implementation of the Hoot protocol we create for our prototype.

4.1 Generating a Hoot

The Hoot protocol can be implemented in a variety of ways using different encryption and hashing algorithms. We created a Python prototype that fully implements the protocol.

A Long Tag is generated by running a SHA256 hash over the Plain Tag. The first 128 bits of the Long Tag are dedicated to identification. We then take the first m bytes of the Long Tag to get a Short Tag, where m is the desired Short Tag length.

The session key is 128 random bits, and the integrity key is 160 random bits. These random numbers are concatenated together and then encrypted using AES with the last 128 bits of the Long Tag as the key.

The plain text is encrypted using AES with the session key as the key. An integrity check is created by performing an HMAC-SHA1 of the cipher text and the integrity key as the key.

With all components created, a Hoot can be generated by printing out a # symbol, the short tag, a space, the encrypted keys, the HMAC digest, and the cipher text.

4.2 Message Length

Given our goal to conform to Twitter, the final output of a message must be under the 140 character limit. Another goal is to be concise and use as little overhead as possible for the entire encryption process. Twitter has a very broad definition of a character. It is not simply ASCII characters, but UTF-8 as well. Based on the encoding, we can squeeze the encryption into even fewer characters. However, this manipulation has issues since many clients do not conform to UTF-8, and there will be little gain if the plain text itself is written in UTF-8. Our prototype can generate Base64 encoded messages or Unicode messages.

Base64 encoding is simple, universal, yet yields longer message sizes. Given a short tag of length m and a plain text message of length n, a Base64 encoded message has

$$m + 20 * floor(\frac{n}{16}) + 118 \tag{9}$$

characters. Assuming a Short Tag of 2 characters, to fit a Base64 encoded message into 140 characters, the plain text message can be at most 31 characters.

A unicode message on the other hand can be much shorter. Given m and n, a Unicode encoded message has

$$m + 12 * floor(\frac{n}{16}) + 64 \tag{10}$$

characters. Assuming again a Short Tag of length 2, a Unicode encoded message can fit within 140 characters when the plain text is at most 111 characters. By using Unicode, the Hoot protocol only uses 29 characters of overhead and can handle reasonable length messages. Obviously, if longer messages are needed, multiple Hoots can be posted, but by having such a low overhead, we are getting 79% utility out of the 140 characters.

5. DISCUSSION

In this section, we discuss a variety of issues and future extensions of the Hoot design.

5.1 Incremental Rollout

1. Can you incrementally roll it out - Service provided by twitter or yourself 2. 140 character limit - Twitter doesn't have to go too far to have all the metadata for encryption

- How hard is it for twitter to do this. - Reference message fitting into 140 char using unicode - Reference peak hps, and how our system holds up

5.2 Adoption

The next question to ask after we have shown that rolling out the Hoot service is feasible would be whether Twitter would actually implement such a feature. Based on the nature of Twitter, we believe that Twitter would not add a secure messaging framework like Hoot. Twitter as a company needs to know what people are talking about, so it can provide relevant advertisement. Adding the Hoot infrastructure to Twitter would prevent Twitter from knowing the content of the messages, and so we believe the Hoot service will never be adopted. However, this need not prevent individuals from using the Hoot protocol over Twitter. As long as Twitter faithfully delivers tweets, users are free to run the Hoot encryption on their own machines over their messages and then post the output to Twitter. In fact, this method is more secure in that a user only has to trust their machine. If Twitter is responsible for encrypting a message, nothing is stopping them from keeping a copy of the plain text message. Paranoid users will always want to encrypt messages themselves, so Twitter adopting the Hoot protocol is unnecessary.

5.3 Usability

As described in the Cover Traffic section, a group can deliberately collide with a popular tag by concatenating an easily rememberable string of text with random letters or numbers. As Miller noted in (The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information), people can remember 7 +/- 2 unique packets of data, so as long as these collisions can be generated with a suffix in that range a user is likely to be able to remember it. We could further improve rememberability by restricting suffix values to only digits, and then generate a suffix of 7 or 10 digits, emulating a phone number. The main requirement is that a group should be able to have a unique shared secret that can deliberately collide with other tags while still being easy to remember and more importantly easy to transfer. Since our proposal does not deal with key transfer, we assume our key communication is done through whisper channels and thus must be short and memorizable for simple transfer. We have found that a collision can be found with only appending 3-6 characters to a prefix, so deliberate collisions can both be found and transferred without much effort.

5.4 Alternate Backends

Even though our protocol was designed with Twitter in mind, it is extensible to other systems and platforms. The Hoot protocol describes a secure way to transfer short messages (with short encryption overhead) across a publicly available network completely openly. Twitter is a great example of this environment but not the only one. Hoot could interact, for example, with a Distributed Hash Table system (DHT) like CHORD or Pastry. Importantly, Hoot can interface with both centralized systems like Twitter or Buzz or decentralized systems. As long as the platform's content is publicly searchable, the Hoot protocol allows secure transmission to anonymous group followers.

6. CONCLUSIONS

FiXme: *TODO:*

7. REFERENCES

[1] G. Danezis, R. Dingledine, and N. Mathewson. Mixminion: Design of a type III anonymous remailer protocol. In *Proc. 2003 IEEE Symposium on Security and Privacy*, May 2003.

[2] A. Pfitzmann and M. Hansen. A terminology for talking about privacy by data minimization: Anonymity, unlinkability, undetectability, unobservability, pseudonymity, and identity management. Draft, Aug. 2010.

[3] P. S. R. Dingledine, N. Mathewson. Tor: The next-generation onion router. In *Proc. 13th USENIX Security Symposium*, Aug. 2004.