COMP 527 Final Project
char[140] twitter_project;

Dustin Bachrach and Christopher Nunu

December 6th, 2010

Introduction

Typical encryption schemes are devised for bi-directional communication channels where both parties are aware of the conversation. These conversations also usually take place in "private," where a bystander must actively watch the network traffic to learn the details of the conversation. New social discussion websites like Twitter and Google Buzz are creating a new type of communication where every message or tweet is posted publicly and forever archived. Directed conversations, that is conversations directed to a certain recipient, are represented on Twitter via @mentions. For instance, a user might tweet: "@John, how are you today?". Although the format of this tweet gives the impression that the message is solely between the author and John, anyone can actually read this tweet.

Many interesting security considerations come into play when we attempt to design a secure protocol for having private conversations on Twitter using the public channel. For instance, we may be interested in hiding the recipient of a message, so @mentions cannot be used. In this paper, we consider many design decisions of a secure protocol for private group communication on Twitter. We also provide an implementation for our chosen security protocol.

Threat Model

Our goal is to allow a user to send a secure message to a private group of individuals allowing only the group members to read the plain-text message. We wish to guarantee as much privacy as possible via an open public timeline like that of Twitter. We now discuss the threat model involved in evaluating the design decisions for the security protocol.

There are essentially two entities which can attack this system. One is the service provider for the communication, which can either be an active adversary, or just a spy. The other is some third party which is either trying to gather information about who you are talking to and what you are saying.

Imagine an evil Twitter that can maliciously tamper with any of the tweets posted to Twitter. Since Twitter has full control of the service, they act as an empowered man-in-the-middle between a secure message sender and the recipient group members. Our most important goal is to prevent Twitter or a third party from reading, creating, or altering a secure message.

Clearly, we cannot directly protect the identity of the secure message sender. A message must be posted to Twitter from some account. By definition, Twitter must know who that user is. We do not consider this limitation a security flaw, since it is fundamental to the problem area. However, if this extra privacy was required, a third party service could offer to post a user's message for them, which would prevent Twitter from knowing who posted the message (although the third party service would know).

Secondly, Twitter can simply refuse to post a tweet, which is a simple Denial of Service attack. Like any platform, protecting against DOS is nearly impossible. A slight modification to this, is the DDOS, which can be performed by other malicious Twitter users. Earlier we mentioned that we would like to protect the recipient identity, so instead of directly tweeting the recipient's name, we will include an identifier that the recipients will search for. If a group of malicious users found this identifier, they could tweet hundreds of thousands of messages with this identifier, which would overload a user's search for the identifier with noise. Later we will describe how we prevent against this action.

An attack that can be done either by Twitter or a third party is a replay attack. Since Twitter is a public forum, anyone can see the encrypted texts posted, and nothing stops anyone from simply copying such a message and posting it again. It will still be a perfectly valid message. This attack is something we want to protect against.

Another thing a malicious Twitter can do is take a valid post and change the associated author or time that it displays with it. Our system needs to be able to handle time altering attacks so that those reading the message have some sense of when the message was sent. On the other hand, who sent the particular message is of a lesser concern. As we mentioned previously, the focus of the project is of protecting the recipients of the message and the message contents, not the author. If this additional security is needed, the protocol would need to be modified as we will discuss.

Finally, we want to prevent Twitter or any third party from gathering trending data on the encrypted posts. Even if they cannot read the messages, if they can observe that someone is posting to the same group of recipients following certain patterns (such as daily at X time, or after major events), this repetition could comprise the groups identity by essentially creating a profile. This type of information is less of a concern to users of Twitter that are simply chatting, but one can imagine that this information can end up being used to identify spies. For example, by observing a pattern between company secrets being leaked to some competitor and a particular employee's secret tweets to a group just hours prior to each incident, the leaker could be identified.

Identification

We want to keep the recipients of a message private, but these recipients must still be able to find messages addressed to them. Therefore, each message must be marked with an identifier, so a recipient can find it. As stated previously, we wish to send messages to groups, so each group will have some identifier, which its members will know. They can then search the Twitter timeline for this identifier and decrypt the secure messages.

Twitter has a concept of groups via #hashtags. For example, a group might be named #lost-fans. Let us assume that only members of this group are aware of its name. We can then perform a hash of this #hashtag, and receive an identifier that only members of #lost-fans will know. The problem arises that if each private message sent to #lost-fans is marked with a single identifier (the hash of #lost-fans), then a bystander could learn that all these messages are sent to the same group (even though they would not know the real name of the group). In our model, we attempt to protect against a bystander recognizing that multiple secure messages are to the same group.

To prevent this, we need a way to get multiple identifiers from the same private group name. That way, all group members can find any of the identifiers, but a bystander would have no idea which secure messages belonged to the same group. We have devised several ways to solve for this, each with tradeoffs and benefits, which we will discus.

If we used multiple hash-routines (like SHA128, SHA256, MD5, etc), we would generate many different hashes for the same value. A message would then be tagged with one of the output hash values, but this method still suffers from repetition because we have a finite number of hash functions, and any number of messages to encrypt.

The other methods all use a MAC to generate multiple hashes with each alternative using a different way to seed the MAC. If the seed to the MAC was included in the cipher-text for the previous message, then it would be guaranteed to be impossible to see any repetitions, but if Twitter fails to post one message, the chain between messages is broken, and therefore no more messages will be identified. We could also seed the MAC function with a number representing the epoch. This method would mean that as long as two messages are not sent within the same epoch, they will have different identifiers. This solution requires a more intensive search process, since searches would have to be performed for every epoch. Finally, we could seed the MAC function with a number that we post in the plain-text tweet. This solution guarantees different identifiers, but suffers from the same intensive search requirements. The methods are a balance between ease of search and maximum privacy.

Once we have an identifier, we want to introduce collisions. Collisions are good because the more groups that use the same identifier, the more uncertainty there is. Collisions with non-secure tweets is also ideal because we want a large set of tweets for the secure messages to blend into. By causing collisions, we give away less information about the underlying group name, which will be very important when we discuss the encryption scheme next. We introduce

identification collisions by truncating the identifier to a short identifier, simply by choosing the first N characters of the hashed output. By removing data from the identifier, we introduce uncertainty and partially prevent reverse lookup hash tables.

Encryption

To ensure that only members of the group can send and read secure messages, we employ symmetric encryption. By establishing a shared secret for the group, all members can write to and read from the group using a fast encryption method. Initially, we decided to use offline distribution for the shared secret, but this idea requires that all the members of a group be in contact via some other secure service to transmit the key. With very large groups, this method becomes infeasible. Instead, we found that the private group name could be used both for identification and for encryption. The encryption scheme that follows demonstrates that our method prevents the threats that Twitter or a bystander can create, alter, or read secure messages.

```
identifier = base64(hash_alg(group_secret))
short_identifier = truncate(identifier)

msg = plain_text + timestamp()
cipher = base64(AES(group_secret, msg))

output = short_identifier + cipher
```

We begin by hashing the group secret and base64 encoding it. As discussed in the identification section, we then truncate the identifier, so we have a short identifier. Next, we append the current time stamp to the input plain-text, which forms the message we wish to encrypt. The cipher-text is formed by AES encrypting the message with the shared secret being the plain-text of the private group name. The final output to Twitter is the short identifier concatenated with the cipher-text.

For example, imagine a user who belongs to the group named "Security-Profesionals" who wants to send this secure message to everyone in the group: "Welcome to the sec-prof group!". The identifier would then be the hash of the group name, that is "84b19n00yhr12dnb997ab". Truncated, the short identifier might become "84b1". The plain-text message is then encrypted with the shared secret being "Security-Profesionals". The final output would then be "84b1" followed by the cipher-text of the message.

Integrity

When a recipient finds a message with the correct identifier, it means that the message might be for him. The recipient then attempts to decrypt the message using the shared secret. If a message decrypts unsuccessfully, then either the

message was not meant for that user (collisions in identifiers), or Twitter has tampered with the message. In either case the user dismisses the message. In the case of successful decryption, the message's timestamp is checked. If the internal encrypted timestamp matches the timestamp of the message as displayed on Twitter, then we can confirm the integrity of the message and be assured that no replay attack occurred. Guaranteeing authorship integrity requires messages to be digitally signed. This method requires a separate public-key store, which is beyond the scope of our problem area. In our system, messages are not signed, so authorship is not trusted. If this sort of security is needed, then digital signatures can be added to the protocol.

140 Characters

The arbitrary limit that Twitter imposes on messages to be under 140 characters makes the encryption process very difficult. When this research project was originally conceived, we planned on primarily focusing on this issue. After researching further, we found the security aspects to be far more interesting than the workarounds to Twitter's character limitation, so we explored the security protocol more thoroughly and left the 140 character limitation as an implementation detail. We propose a few methods for addressing the character limitation.

First, our research has more to do with encryption for messages in any public timeline rather than specifically Twitter. We can simply choose Google Buzz as our service, which has no character limit.

Another option would be to take advantage of Twitter's Unicode support, which would allow us to squeeze a few more bits into each character. That is, instead of base-64 encoding the cipher-text, we could apply an algorithm that would map into the Unicode alphabet. This method would save extra characters, but it did not lead to any significant improvement in writing longer secure messages.

The obvious solution is to perform the encryption process and then split the cipher-text into chunks, where each chunk is the length of the identifier subtracted from 140. Using this method, we can post arbitrarily long cipher-texts to Twitter. We would introduce a way of marking the last message as the final message, and we would need to ensure that Twitter was unable to validly post partial messages.

Finally, to avoid Twitter's character limitation, we considered encrypting the plaintext and publishing the cipher-text to a webpage. This web page's URL would then be shortened using a URL shortener like bit.ly. The shortened URL would be posted to Twitter along with the identifier. A group member would then need to find tweets using the identifier, follow the bit.ly URL, and decrypt the contents of that URL.

Implementation

The encryption and decryption services are implemented in a Python script hosted on GitHub: https://github.com/dbachrach/char140. The script can generate a secure tweet from a plain-text message and private group name. It will also decrypt a secure tweet, which has been received via Twitter to show the original plain-text message.

Future Work

As mentioned previously, collisions are helpful in our process, but certain identifiers will more likely collide than others. For instance, an identifier beginning with "the" will collide with thousands of tweets, whereas an identifier like "9X3" will have very few collisions. If we used variable length identifiers, then we could use less characters from identifiers that are very unique. We would try to find a balance for each tweet, which maximizes collisions without creating too many collisions where search becomes infeasible. This approach requires us to search the Twitter timeline and query for variable length identifiers until we find the proper length.