End-To-End Message Encryption for Tinder

Katie Ho, Akhil Nistala, Kevin Tu
MIT CSAIL, {ksho, anistala, kevintu}@mit.edu
Professor Ron Rivest
May 11, 2016

ABSTRACT

Users of dating apps often share sensitive personal information on the app's messaging system. Because this information is often personally identifying, message security and therefore user identity protection is of utmost importance. For Tinder, a rapidly growing dating start-up, this message security is largely absent. Therefore, we propose an end-to-end messaging encryption scheme that provides message security and gives Tinder the ""backdoor" ability to access and read any messages if necessary. At a high level, the suggested implementation puts the onus on Tinder to facilitate the generation and distribution of keys to all users, but then maintains a hands-off third party role in the end-to-end messaging unless legally or otherwise required to investigate messages.

1. INTRODUCTION

Tinder is a rapidly growing location-based dating startup that aims to bring people–often strangers–together via a mutual opt-in system. Users create an account on Tinder linked to their personal Facebook. A Tinder profile contains self-selected images from Facebook, a short written bio, location data relative to the viewing party, and a list of mutual Facebook friends. A user is shown other Tinder profiles sequentially. At each profile the user is forced to make a decision, "LIKE", indicating they'd like to connect with the person, or "NOPE," indicating the opposite, before they can view another profile. If two users "LIKE" each other, they can begin a private conversation with each other on the app.

Tinder has seen incredible growth since its founding in 2012. As of late 2015 Tinder was approaching 50 million users [1], with well over 10 million daily active users [2]. Tinder self-reported a daily average of 26 million matches per day, totaling 10 billion matches overall throughout its 4 years [3].

The app provides this service to a significant number of users, which requires it to maintain nontrivial amounts of messaging information. Because the application encourages meetups, people often share sensitive personal information via the app, including but not limited to their phone numbers and addresses. For this reason, Tinder requested that our team develop and propose a more secure messaging encryption scheme. We had multiple objectives: (1) to increase the user protection provided by the app, (2) to provide Tinder with a backdoor to access and read any messages if necessary, and (3) to prevent users who have unmatched from accessing their previous conversation.

2. CURRENT IMPLEMENTATION

Tinder currently uses a fairly minimal Data at Rest encryption scheme for their users' messages. This encryption exists only at Tinder's servers, where all the messaging data is stored. While the identity of the parties and users of the message exchange is authenticated by Transport Layer Security (TLS), the messages transmitted are sent in plaintext. This makes the TLS records vulnerable to various attacks.

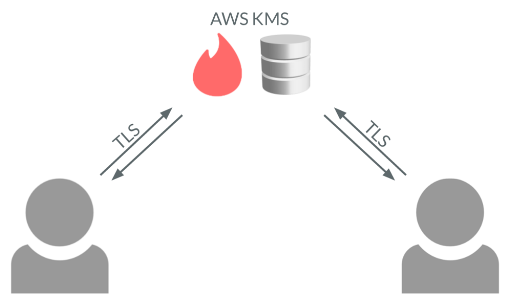

Fig. 1. Messages sent between users pass through Tinder's servers, where they are stored using KMS. Currently, only TLS guarantees authentication - messages are otherwise unencrypted.

In Tinder's messaging model, Tinder acts as a middleman who listens in on all communications–all messages between users are first sent to Tinder, where the information is stored at the servers. The messages are then sent from the Tinder server to the receiving user. Encrypted versions of the messages are stored in Tinder's databases. Tinder then uses Amazon Web Services (AWS) Key Management Service (KMS) to encrypt and store the keys. The encrypted messages and the encrypted key are stored together so that Tinder can decrypt the messages on demand.

The messaging between Tinder and its users is secured via TLS. In the current implementation, this only ensures that the two communicating parties are authentic-the user is communicating with the real Tinder and vice versa. Following the successful TLS handshake, the users can communicate with Tinder under the TLS record protocol. However, in Tinder's implementation, protocol messages are unencrypted plaintext messages and are therefore susceptible to attack by adversaries.

3. Design Goals & Considerations

At a high level, our project was to design an end-to-end message encryption scheme that allows for a backdoor. We broke this down into five design goals. These goals summarize our main considerations when designing the encryption scheme.

a) Messages between "matched" users A and B should be encrypted while in transit and on their devices: This design goal is in line with the basic definition of an end-to-end encryption scheme. Encrypting messages on users' devices and also while in transit ensures that only the people communicating (matched Tinder users) can read the messages.

b) When users A and B "unmatch," neither should be able to decrypt their conversation, even locally cached copies: This design goal ensures that users cannot see conversations from matches that no longer exist. The idea behind this is that it will prevent or at least dissuade users from using old messages to blackmail or threaten each other. We ignored many edge-cases when considering this design goal, since the threat model is simply too vast. For example, malicious users can simply screenshot conversations before a match becomes nonexistent, or use a jailbroken device to access locally cached copies.

c) Tinder should always be able to decrypt messages, even after two users have unmatched: This design goal falls in line with the "backdoor" of our encryption scheme. Being able to decrypt messages even after users unmatch is critical when responding to user complaints or requests from law enforcement.

d) The encryption scheme should be implementable on a node.js platform running on AWS: This design goal simply ensures that the encryption scheme is compatible with Tinder's current framework and guarantees backward compatibility.

e) The encryption scheme should be fast and scalable: This design goal ensures that the encryption scheme doesn't lead to a decrease in quality of Tinder's main product. There should be a zero or negligible increase in message latency. The scheme must be scalable to accommodate the millions of current Tinder users.

4. PROPOSED ENCRYPTION SCHEME

Our proposed encryption scheme can be broken down into several subcategories which make it easier to describe - key generation, key distribution, behavior on match and on unmatch, message passing, and message storage. These aspects of our encryption scheme are described in detail below, as well as some miscellaneous points that should be taken into consideration.

Some terminology used in this section may be confusing – following are some definitions to act as clarifications.

- *User* - A person using the Tinder application
- *Client* - The Tinder application code on a users device. A user interacts with the Tinder application, whereas the client interacts with the Tinder server.

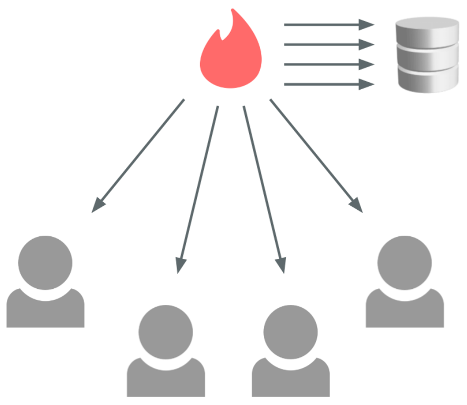

Fig. 2. Tinder will generate a random secret and public key for all users, which it is responsible for distributing. It also maintains a copy of both keys along with a user ID in its own secure database.

4.1 Key Generation

The main Tinder server generates a unique (public key, secret key) pair for each user. These keys will be generated on account creation for new users. For existing users, these keys will be generated on demand as the new encryption scheme is rolled out. Tinder stores each users secret key (SK) using Amazons

Key Management Service (KMS). This process can be summarized through the steps given below.

1) The Tinder server generates a K_X = (PK, SK) pair for user X
2) The server encrypts K_X with a special Tinder encryption key K_T to produce K = E(K_X, K_T)
3) Amazon KMS essentially stores an encrypted copy of each users (PK, SK) pair, as well as an encrypted copy of the key that was used to encrypt the (PK, SK) pair.

4.2 Key Distribution

Each user acquires his/her (public key, secret key) pair from the Tinder server using Shamir's Three-Pass Protocol. We assume that communicating over TLS and using an appropriate certificate authority will help the Tinder server and user authenticate each others identity. Each user will acquire his/her key through the following sequence of events:

1) Existing user updates Tinder app OR new user installs Tinder app.
2) User authenticates him/herself by logging into Tinder with Facebook.
3) Users client sends request to Tinder server to generate unique (public key, secret key) pair.
4) Client and Tinder both create random keys to communicate with each other.
5) Users client pulls this (public key, secret key) pair from the server using Shamir's Three-Pass Protocol.

Using Shamir's Three Pass Protocol ensures that the key exchange is secure against eavesdroppers without having to previously agree on any shared key. Due to the computational and communication requirements required, we chose not to use this encryption for messaging. However, because each party can communicate securely without overhead, this protocol is appropriate for exchanging keys. This allows the users to transfer to symmetric encryption, which performs significantly better than asymmetric encryption.

Tinder will tentatively roll out this encryption scheme as an app update. Future users will download the updated version of the app and will be all set to go with the new encryption scheme. However, there may be many existing users who either don't update the app or take a considerable time to do so. One way to approach this problem is to give existing users some leeway time, perhaps a few weeks to a few months, during which they can continue to use the app as is. However, users who fail to update their app within this time will be removed from the Tinder recommendation system until they use the new encryption system.

4.3 Behavior on Match

For each new Tinder match, a Diffie-Hellman key exchange is initiated between the two users' clients to establish a shared secret key. The users learn each other's public keys when they swipe right on each other – e.g. user A's client stores user B's public key when A swipes right on B. A's client then stores this public key along with a timestamp in a local cache. Public keys that are more than two weeks old (indicating that no match has occurred in that time) will be cleared.

Each user then uses his/her own secret key along with with the other user's public key to generate the shared secret key. This results in a unique secret key for this match, which Tinder can learn on-demand when necessitated, since Tinder maintains a record of all secret keys in their user-key database.

We decided to use a Diffie-Hellman key exchange for our encryption scheme as it is secure, simple to follow, and already has a JavaScript implementation that is compatible with Tinder's existing framework.

4.4 Message Passing

Once two users match and their clients have established a shared secret key, they can exchange messages using the Advanced Encryption Standard (AES). We have decided to use AES for message encryption for several reasons; the most pertinent ones are listed below:

1) AES handles variable message length
2) AES is highly performant
3) AES already has an existing JavaScript implementation

Handling variable message length is obviously a key consideration as messages in Tinder conversations definitely fall into this category. Furthermore, high performance is critical to the design goal of having a fast and scalable encryption scheme. Finally, the existing JavaScript implementation of AES is compatible with Tinders node.js framework, which is a huge bonus.

4.5 Message Storage

Message storage is handled in a similar manner to key storage. A special Tinder encryption key KT (different than the one used for key storage) is used to encrypt all messages passed between users. The encrypted message is then stored using Amazon KMS along with an encrypted copy of the encryption key. This aspect of the message encryption scheme mirrors the current Data at Rest encryption used by Tinder.

4.6 Behavior on Unmatch

This part of the design can be best explained through an example. Suppose that, for whatever reason, user A unmatches user B. A's client will first delete all messages with B from the Tinder app. A's client will then tell the Tinder server (through an API call) of this unmatch. The Tinder server then makes an API call of its own, signalling B's client of this unmatch. The next time B uses the app, B's client will delete all messages with A from the app. Both A's and B's clients will also delete any locally cached copies of their conversation.

The Tinder server then blocks any attempts at further communication from either A or B – this isn't done through a key 'revocation' but rather just by preventing any sent messages from reaching their destination. From a UI standpoint, neither A nor B can see a conversation window from this recent unmatch, which also prevents any further messages from being sent.

4.7 Miscellaneous Points

In addition to the encryption scheme discussed so far, we are presenting Tinder with a few additional recommendations that are in line with our design goals.

The Tinder client app currently caches all conversations. This results in good user experience, but at the slight expense of security. For example, if user A unmatches user B and B's phone is in airplane mode, then B will be able to see the entire conversation until reconnecting to the network.

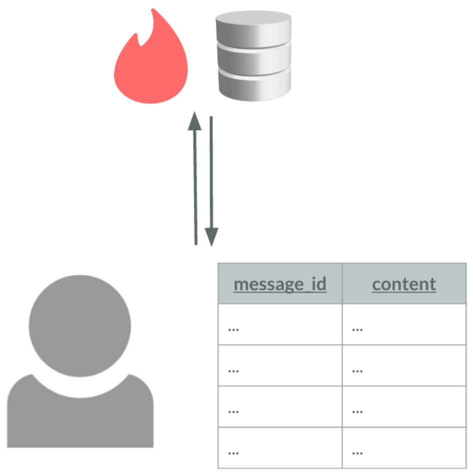

Fig. 3. One suggestion to accomplish pseudo-ephemeral message access is to reduce the number of messages cached on the phone. Instead, require users to ping the server in order to access previous messages.

A solution to this is to only locally store the previous 10 messages or so messages per conversation on each client. This solution gives the benefit that adversaries will only be able to access very few messages on an unmatched conversation. However, a drawback to this solution is that there will be increased latency to load the rest of any conversation, as each load will require an API call to the Tinder server.

We give this recommendation with the caveat that 10 is an arbitrary number, and thorough testing should be done to determine the optimal number.

Another feature that we recommend is allowing users to opt-in to an additional passcode or touch-ID verification for messages. This augments security in the case that a user's phone is stolen by an adversary who knows the phone's main passcode.

5. ANALYSIS

In this section we use several different metrics to evaluate our proposed encryption scheme and use calculations to justify our design decisions. We address concerns regarding both the scalability of the system and the speed of the system.

5.1 Scalability

While most of this paper has been focused on messages sent between two individual users, it is important to remember that Tinder is a service used widely across the world. Our proposed framework will need to scale seamlessly to connect users who may be located at very different geographies and many users all in close proximity.

Based on Tinders current and projected usage, we have determined that on average between 0 and 10,000 messages are being sent every second (range purposefully widened to mask sensitive information). At this load, Tinders servers that store messages and doubly encrypt the keys using Amazons Key Management Service are sufficient for processing and storing every message that is sent. Now considering this load within our proposed implementation, Tinder would not need to exercise any additional computation or processing power on the server side to store messages. All encryption and decryption is done on the client side, distributing the work and helping ensure a bottleneck does not occur on the servers. Instead of storing plaintext messages, Tinder servers will now simply store encrypted messages. The only additional storage required would be for tracking the secret keys of each user, something that could be run on a separate machine with minimal impact on scalability. In summary, due to the nature of our encryption system, Tinder will remain scalable as it continues to grow its user base.

5.2 Timing Analysis

One of the primary design goals of our encryption system was to ensure that it was fast and high performing. To validate our proposed framework we conducted a careful analysis of the impact this additional encryption and decryption would have compared to Tinders pre-existing framework. For the purposes of this timing analysis, we ignore the one-time setup costs of generating and distributing public and secret keys. This procedure will be implemented to take place when a user first signs up or installs the update, making the additional time required to complete these tasks negligible. Furthermore, our system will have no additional impact on the unmatch process or the server-side message storage process (as described in Section 6.1), since the underlying processes are not affected by the end-to-end encryption. To asses the impact of encryption and decryption has on the speed of our system we consider two scenarios:

5.2.1 On Match Timing Analysis: When a match occurs, the two parties engage in a Diffie-Hellman key exchange. The way we designed the app to work, when a user is looking at recommended matches, they would have direct access to each persons public key, which would be embedded in each individuals profile. On a match, it is trivial for a user to calculate the shared secret key for that pairing by making one exponentiation with their private key and the public key of the matched user. This operation takes a negligible amount of time, and therefore the additional step in the matching process does not impact the overall performance of Tinder.

5.2.2 Message Passing Timing Analysis: The only portion of our system that has a notable impact on performance is the actual message passing process. By using AES as our encryption protocol, a well tested industry standard, we are able to estimate the approximate effect on speed. Assuming that a normal message consists of a couple hundred characters, our research and calculations indicate that the encryption and decryption process would take at most a few milliseconds. Furthermore, the decryption process can be run in parallel due to the structure of AES, allowing for even faster speeds. From that point, the encrypted message is passed over TLS to the Tinder servers in the same way that it currently is. Therefore, the net difference in overall performance due to our changes is on the order of a few milliseconds, a worthwhile tradeoff to achieve true message security.

6. FINDINGS

We had several insights and findings from designing this message encryption scheme for Tinder, which we've summarized below.

We don't believe we can prevent the visibility of messages by an active adversary who has previously had access to the messages. As a result, we focused our efforts on an end-to-end encryption scheme with a backdoor, rather than on client side revocation (which prevents unmatched users from seeing previous conversations).

We came to this conclusion based off the vast threat model – adversaries can screenshot or even write down entire conversations; they can keep the ciphertext and their decryption key (which allows for reading conversations even after an unmatch); etc. Given this threat model, we firmly believe that it is not worth it to invest in a client/server model that prioritizes client side revocation over system performance.

However, this being said, it is still a good idea to have an encryption scheme for locally stored messages that satisfies some basic criteria. The encryption scheme should be simple to implement (not many developer hours) and shouldn't affect system performance (message latency should be about the same as before).

7. FUTURE WORK

We have demonstrated that our proposed end-to-end message encryption scheme satisfies the original design considerations and is highly performant in standard use cases. There are additional components to the project that our team plans to explore further, including (1) empirical validation of performance thresholds and (2) filtering and searching encrypted messages.

As mentioned in Section 4.7, our design balances security and latency by minimizing the number of cached messages on a clients phone. This decision makes it infeasible for a client to, upon an unmatch, manipulate or retain a substantial number of cached messages from the old conversation. The corresponding tradeoff in performance is that retrieving old messages would require additional server calls, increasing latency. A next step for this project includes determining what the optimal number of messages to cache on a clients phone is, based on this tradeoff. This would require running extensive tests with varying thresholds of messages and examining at the resulting latency that users see. Based on the results of these tests we would be able to determine an exact threshold that best aligns with our design goals.

Our proposed message passing scheme stores the encrypted messages on Tinder servers in the case that the messages need to be revisited by legal authorities. However since the messages are encrypted, it becomes extremely difficult to filter, sort, or quickly search through the data. Tinder may want to read and classify messages that are flagged by users to improve their bot recognition and spam prevention features. Our team plans to investigate encryption schemes that may allow for searching and sorting without fully decrypting the entire database, which would compromise the security of the overall system. Research in this field conducted by Microsoft Research and Boston University (insert citation) explores potential abstractions that could meet this requirement. While this specific consideration was outside the original scope of the project, we hope to continue to develop this idea and empower Tinder to be able to interact with messages in this manner.

8. CONCLUSION

This paper demonstrated an encryption scheme that can greatly improve the current implementation that Tinder employs. The combination of key distribution and added layers of encryption and decryption ensure data integrity for messages while still allowing Tinder to have a means to read messages if necessary. Through a rigorous design process we were able to meet all requirements and considerations that the system needed to follow, without significant impacts to scalability and speed. After analyzing the viability of the framework under different scenarios, we strongly recommend that Tinder adapt this end-to-end encryption scheme as a compelling improvement to their current messaging system.

9. ACKNOWLEDGMENTS

This paper was made possible by the support and guidance of the Tinder Security Team. In particular, we would like to thank Anthony Trummer, Shiv Patel, and Alex Segal for their technical insights and time spent working alongside us. Additionally, Professor Ron Rivest and Conner Fromknecht were integral to the development of this project and we thank them for their design recommendations.

REFERENCES

[1] Giuliano, Karissa. ”Tinder Is Swiping Right on Monetization.” CNBC. 02 Mar. 2015. Web. 18 Mar. 2016. ¡http://www.cnbc.com/2015/03/02/-monetization.html¿.

[2] Margalit, Liraz. ”Tinder And Evolutionary Psychology.” TechCrunch. 27 Sept. 2014. Web. 18 Mar. 2016. ¡http://techcrunch.com/2014/09/27/tinder-and-evolutionary-psychology/?ncid=rss¿.

[3] Tinder. ”Tinder.” Tinder. 2016. Web. 18 Mar. 2016. ¡https://www.gotinder.com/press¿.

[4] Baldimtsi, F. & Ohrimenko, O. (2015). Sorting and Searching Behind the Curtain.

[5] Chatterjee, A. & Sengupta, I. (2015). Searching and Sorting of Fully Homomorphic Encrypted Data on Cloud.