Secure Instant Messaging
(Threema, TextSecure, etc.)

CS 470
Internet Security Protocols
Ali Aydın Selçuk
(with Burak Kocuroğlu)

Instant Messaging Security
• IM is used more than SMS worldwide
• We wouldn’t want anyone to read/modify our private conversations
• Possible threats:
  – Government intelligence
  – Curious family/friends/neighbors
  – Analyzing messages for targeted ads etc.
• HTTPS?
  – Encrypts only between client and server
  – Current CA model limitations (CA compromise, etc.)

Non-Secure (Not E2E) IM Examples
• Skype
• Viber
• Line
• WhatsApp (recently started deploying E2E)
• Telegram (“normal chat”s)
• ...

E2E Encrypted IM Examples
• TextSecure
• Threema
• CryptoCat
• Telegram (“secret chat”s only)
• WhatsApp (recently, with TextSecure’s protocol)
WhatsApp

- De-facto instant messaging application
- Facebook bought it for $19 billion (2014)
- Closed source and its security design is not documented
- Many vulnerabilities have been discovered and fixed over time

WhatsApp History of Vulnerabilities

- Password from IMEI number or MAC address
- Same RC4 key stream was used twice
- Chat logs were saved to SD card (other apps could read it!)
- Constant disk encryption key (databases were encrypted but one could easily access the encryption key)
  - [https://blog.thijisalkema.de/me/blog/2013/10/08/piercing-through-whatsapp-s-encryption](https://blog.thijisalkema.de/me/blog/2013/10/08/piercing-through-whatsapp-s-encryption)
  - [http://scholarworks.uno.edu/cgi/viewcontent.cgi?article=2736&context=td](http://scholarworks.uno.edu/cgi/viewcontent.cgi?article=2736&context=td)

WhatsApp History of Vuls (cont.)

Incorrect use of SSL:
- No certificate pinning
- Weak cipher support
- Null cipher support
- SSLv2 support

http://www.praetorian.com/blog/whats-up-with-whatsapps-security-facebook-ssl-vulnerabilities

Whatsapp’s EFF Scorecard (2014)

[https://www.eff.org/secure-messaging-scorecard](https://www.eff.org/secure-messaging-scorecard)
Whatsapp – Final Remarks

• Started with many major security bugs; now they seem to figure out most of them.
• Bringing E2E encryption support (as of 2015)
  – They have been working with Moxie to bring it in.
  – Considered a great success for the sec. community.
• Just beware: Whatsapp is able to switch you to the old protocol (for backward compatibility) and do MitM.

Threema

• Swiss proprietary application (for $1.99).
• Gained popularity after WhatsApp’s acquisition by Facebook
• Platforms: iOS and Android
• Initial Release: 11 December 2012

Threema – Registration

• Generates random Long Term Keypairs (LTK)
• User input (moving finger on screen) is requested to increase randomness
• Server generates a username (8 byte) and associates it with LTK
• Verifies phone number and e-mail (optional)

Threema – Authentication

• Chat server uses a custom protocol over TCP; directory and media servers use HTTPS.
• All servers support certificate pinning.
• Clients are authenticated by Long Term Keypairs
• Users authenticate each others’ public keys by:
  – reading a hash of it
  – scanning QR code of their public keys
Threema – Key Management

- Client identities are tied to Long Term DH Keypairs (LTK) and an 8 byte generated username.
- When a client is communicating with the server, a Short Term Key (STK) is generated using Diffie-Hellman.
- Server STKs are refreshed at each application startup or 7 days.
- No PFS between clients. (DH keys remain the same between a given pair of clients.)
- Encryption: Salsa20 stream cipher (with Poly1305 for MAC)

Threema - Message Encryption

- Threema - Encryption Validation

- Since Threema is closed source, you cannot be sure if it works as the way described. Because of that, they introduce a debug logging system.
- Logging mechanism is enabled within application, a few messages are sent/received.
- Logs are extracted from device.
- Private key is extracted from Threema ID backup.
- Then you can check if the security is the same as documented (Of course we cannot be sure if logged messages and sent messages are the same without analyzing network).

Threema – EFF Scorecard
Threema – Final Remarks

• One of the few paid secure messaging applications

• Quite user friendly

• Not open source; but has good security documentation

• General security design looks good

Cryptocat

• Chat application used as browser extensions and iPhone application

• Uses Off-the-Record Messaging (OTR) protocol for message key management

• Shared keys are generated with Diffie-Hellman key exchange using keypairs (256 bits for encryption, 256 bits for MAC)

• Its certificate is pinned at Google Chrome (Chromium)

Off-the-Record (OTR) Communication

• Perfect Forward Secrecy (PFS)
  – Uses long-term keys to sign short-term DH pub keys.
  – Uses a per-message DH key exchange.

• Deniability: Only MAC but no signature on messages.

• MAC is derived from the short-lived key, so anyone who can verify can also forge.

• Old MAC keys are also published. (optional)

Cryptocat – Registration

• At each login session, a new public / private key pair is generated.
  (This is subject to change. (Github, 2015))

• Implements OTR messaging:
  – Uses long-term keys to sign short-term DH pub keys.
  – Uses a per-message DH key exchange.

• Server doesn’t verify phone number or e-mail
Cryptocat – Key Management

- Uses certificate pinning to authenticate servers (over XMPP)
- Server does C-R with the client to verify and distribute the key.
- Clients compare their public key fingerprints over a secure (out-of-band) channel to authenticate each other.
- Must be done at the beginning of each chat session
- Browser extension creates a new keypair at each session
  - Difficult to use (at the beginning of every chat you need to verify the other party)
  - Subject to change

Cryptocat – EFF Scorecard

<table>
<thead>
<tr>
<th>Encryption in transit?</th>
<th>Encrypted so the provider can’t read it?</th>
<th>Can you verify contacts’ identities?</th>
<th>Are past communications secure if your keys are stolen?</th>
<th>Is the code open to independent review?</th>
<th>Is security design properly documented?</th>
<th>Has there been any recent code audit?</th>
</tr>
</thead>
<tbody>
<tr>
<td>✅</td>
<td>✅</td>
<td>✅</td>
<td>✅</td>
<td>✅</td>
<td>✅</td>
<td>✅</td>
</tr>
</tbody>
</table>

Telegram

- Founded in 2013 by the brothers Nikolai and Pavel Durov, the founders of VK, Russia’s largest social network
- Telegram Messenger LLP is an independent nonprofit company
- Clients (iOS, Android) are open source, server is proprietary

Telegram

- Uses a custom protocol (MTProto) which has been criticized heavily in the community.
- Client-to-server communication is encrypted by MTProto, and uses HTTP (without SSL).
- Client-to-client communication doesn’t have end-to-end encryption by default.
- Secret chat session have to be started manually for end-to-end encryption.
Telegram: Registration & Authentication

- Servers are authenticated by checking their public keys (RSA public key of each server) in the app code (like certificate pinning).
- Clients are authenticated (by server) using a (symmetric) authorization key, auth_key.
- The authorization key is generated during registration by a 2048-bit DH exchange, and is shared by the client and server.

Telegram – Key Management

- Server has shared symmetric keys with every client device (auth_key)
- Each communication with the server is encrypted with auth_key using MTProtocol.
- For E2E encryption, both parties have to be online.
- For each session, a fresh 256-bit AES session key is generated by a 2048-bit DH exchange.

Telegram – Authentication (cont.)

- Clients can verify each other and the session key using a 128-bit white-blue QRcode-like picture.
  - Generated from the session key

Telegram’s Crypto Contests

- First one: “$200,000 to the hacker who can break Telegram.”
- Someone found a bug and received $100,000. In the below formula nonce is created by the server to increase randomness. But server can send different nonces for clients and carry out a MitM attack:

  \[ \text{key} = (g^a)^b \mod \text{dh.prime} \oplus \text{nonce} \]

- Second one offered $300,000 – no winners.
Telegram – EFF Scorecard

<table>
<thead>
<tr>
<th>Encrypted in transit?</th>
<th>Encrypted so the provider can't read it?</th>
<th>Can you verify contacts' identities?</th>
<th>Are past communications secure if your keys are stolen?</th>
<th>Is the code open to independent review?</th>
<th>Is security design properly documented?</th>
<th>Has there been any recent code audit?</th>
</tr>
</thead>
<tbody>
<tr>
<td>✅</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>✅</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
</tbody>
</table>

Telegram – Final Remarks

- Uses a custom encryption protocol (almost never a good thing).
- Default chat mode is not end-to-end encrypted; to start a E2E encrypted chat, the other party must be online.
- Has lots of users (50M-100M install on Google Play).

TextSecure

- Developed by Whisper Systems (Moxie Marlinspike)
- Then acquired by Twitter (2011)
  - Undisclosed amount
- Became open source as Open Whisper Systems

TextSecure – Registration

- Device generates 18 byte password which will be used to authenticate the device.
- Verify phone number (send a code via SMS, like Whatsapp, Threema, Telegram).
- 52 byte signalingKey is generated. This key is used to encrypt messages sent via Google Cloud Messaging (GCM).
TextSecure – Authentication

- HTTPS is used, server is verified by certificate pinning.
- Client is authenticated by HTTP Auth (phone number:password) for all requests.
- Clients can authenticate each other using hash or QR codes of their public keys.

TextSecure – Key Management

- Each user has a unique identity key (public - private EC-DH keypair).
- Chats starts with shared key generation which uses identity keypairs (and random key parts).
- For every message, message encryption key is changed!
- Feature: Preserves PFS while allowing sending a message to an offline user.

TextSecure – Key Management (cont.)

- Device generates 100 PreKeys and one ‘last resort key’ then these keys are sent to the server.
- These keys are used for asynchronous messages (they allow sender to send messages even if receiver is offline), obtained from the server.

TextSecure – Encryption

```
if (user A) {
    ECDHE(b, A)
    ECDHE(B, a)
    ECDHE(b, a)
} else {
    ECDHE(A, b)
    ECDHE(a, B)
    ECDHE(a, b)
}
```

A: Long-term identity key of user A
B: Long-term identity key of user B
a: Ephemeral key of A
b: Ephemeral key of B

These three shared secrets are combined as one and used as seed of an HKDF to create Root Key (RK) and Chain Key (CK)
TextSecure – Encryption

Key generation protocol: “Axolotl ratchet”

Axolotl: Ratchet:

RootKey (RK): used to derive CK, updated at every ratchet

ChainKey (CK): used to derive MK when one party is sending continuous messages

MessageKey (MK): used to encrypt message
  - Each DH ratchet is combined with the existing root key (RK) to derive a new RK as well as a “chain key” for that DH pair.
  - Each “chain key” is hash iterated for each message sent/received under that chain.

TextSecure – EFF Scorecard

- Supports asynchronous messaging without compromising PFS
- Lots of good reviews from the security community
- Fully open source, mostly OK security documentation
- WhatsApp chose TextSecure’s “axolotl ratchet” protocol for its E2E encryption (may be experimental)
References

- https://github.com/trevp/axolotl/wiki
- http://nacl.cr.yp.to/
- http://news.ycombinator.com/item?id=6913456
- http://unhandledexpression.com/2013/12/17/telegram-stand-back-we-know-maths/
- http://thoughtcrime.org/blog/telegram-crypto-challenge/
- https://telegram.org/blog/crowdsourcing-a-more-secure-future
- https://whispersystems.org/blog/private-groups/
- https://whispersystems.org/blog/asynchronous-security/