End-to-end Encrypted Communication Between Multi-device Users

Halil Kemal Taşkın, Murat Demircioğlu and Salim Sarımurat

***Abstract*—The rapid growth of the mobile device industry has led people to start to use more than one device (smart phones, tablets, laptops etc.) and this results in a multi-device environment. People often tend to use more than one device in their daily lives by switching between them or using them simultaneously. From the security point of view, end-to-end encryption is a problem in multi-device environment. Known applications are lack of multi-device end-to-end encryption schemes and have some known issues [4].**

In this paper, we present a new protocol that allows end-to-end encrypted communication, multi-device user synchronization with offline key exchange capability. Our protocol can be deployed on cloud services without compromising the security requirements.

***Index Terms*—Multi-device Users, Mobile Devices, Private Communication, End-to-end Encryption, Cloud Computing.**

I. Introduction

INTERNET became the state of art communication tool between people. In its early stages, instant messaging (ICQ,IRC, MSN etc.) was the first major communicator and after that e-mail became more popular. But, with the growth of mobile devices, instant messaging, again, became the major communication way for people.

Instant messaging using mobile devices was a game changer since it allowed people to communicate in realtime at anywhere and anytime. Also, mobile technologies brought the cloud technologies more popular so every data could be reached from anywhere. On the other hand, from security point of view, instant messaging over cloud networks had privacy issues since your data could be easily sniffed or intercepted over the cloud network. There are several cryptographic solutions for end-to-end encrypted communication. OTR based protocols [1], [2] and ZRTP [3] are some of the popular protocols used for this purpose. Using end-to-end encryption, relaying servers or other third parties that have access to network between the communicating parties can not intercept or sniff the messages sent through the network.

However, people start to use more than one device (smart phones, tablets etc.) and this resulted in a multi-device environment. In a multi-device environment, users expect that any message sent to any device of a user should be synchronized instantly to all devices of the user. People use their devices in their daily lives by switching between them or using them simultaneously. So synchronization between these devices by preserving end-to-end encryption capabilities is an important problem. Cloud based services are easiest and cheapest solutions for network infrastructures. Thus, preserving these security properties using a cloud based service is also an important problem.

Conventional end-to-end encryption protocols don't support synchronization of messages when one party on the communication is a multi-device user. Instant messaging applications such as Whatsapp, Viber and Facebook Messenger are widely used but their protocols are not known clearly and they are mostly lacking end-to-end encryption schemes. Applications like TextSecure, Threema and Telegram claims that they use end-to-end encryption. But, they all are lack of multi-device environment capabilities. Apple's iMessage service is the only one which handled both multi-device user environment and end-to-end encryption capabilities. But recent studies [4] showed that the protocol is vulnerable to a man-in-the-middle attack that could be executed by the relaying servers which totally destructs the end-to-end encryption flavor.

In this work, we present a new end-to-end encryption protocol which allows multi-device user synchronization with offline key exchange capability and is resistant against man-in-the-middle attacks. End-to-end encryption is achieved in such a way the protocol could easily be deployed on a cloud based service without compromising the security requirements.

We give the abstract protocol details. Some parameter and format selection is up to implementation. In Section II, we state the security requirements for the concerns mentioned above. Our protocol details are given in Section III. Finally, Section IV concludes our work.

II. Security Requirements

In our protocol we have some assumptions that enables security features. Below, these security requirements and concerns are discussed. Also, details of how we use them in our protocol is given.

- **End-to-end Encryption Flavor**: End-to-end encryption is the major aim of our work. All data leaving a client must be encrypted in such a way that the cloud service operators have no knowledge of the message content.Thus, our protocol makes use of Public Key Cryptography (PKC) for exchanging keys before the communication starts.

H.K. Taşkın and M. Demircioğlu are with the Department of Cryptography, Institute of Applied Mathematics, Middle East Technical University.

H.K. Taşkın, M. Demircioğlu and S. Sarımurat are with Oran Information Technologies.

S. Sarımurat is with the Department of Computer Science and Engineering, Sabancı University.

This research is supported by the Scientific and Technological Research Council of Turkey (TÜBİTAK) under the project number TEYDEB-1130063 titled "Mobil Terminaller Üzerinde Kriptografik Yöntemler Kullanılarak Uçtan Uca Güvenli Bir İletişim Sisteminin Tasarlanması ve Geliştirilmesi".

- **Private Key Distribution**: In a multi-device environment with PKC, it is essential to have the long term private key on all devices. The easiest way to distribute data including private keys is to store them on the cloud in an encrypted way that server should not be able to decrypt data.
- **Server-side Security**: The data stored on the server should be resistant against brute force and dictionary attacks. In our implementation we use double scrypt [10] key derivation function with salting to store user password. Server is also responsible for the message delivery. For every user there should be a storage quota on the server, higher quotas mean more messages and key parts users can store.
- **Offline Key Exchange**: To establish a key exchange in a conventional end-to-end encryption scheme, both parties have to be online. But in a mobile environment this could not be always possible because of its asynchronous nature. Thus, we introduce a proper way of key exchange while two parties don't require to be online at the same time. We store signed Diffie-Hellman key parts on the cloud and make use of them when needed. Size of the short term key list is optional. It can be changed due to the increasing load of messages. In our implementation it is fixed to 1024.
- **Re-keying**: To achieve forward secrecy, session keys should be updated in a regular interval. In our protocol re-keying is handled in a similar way as it is done in [1]. Also, when the long term private key compromised, it should not be possible to recover previous session keys. To achieve this, we use long term keys only for signing key exchange process.
- **New Device Addition**: In a multi-device environment new devices should be easily added and removed from the user's account. But authentication is also an important issue. We require two-factor authentication when a device is registered. There is no maximum limit for devices but it can be limited up to implementation.
- **Certificate Pinning**: For client-server communication we use certificate pinning for TLS. This approach provides two major advantages. First, there is no need for a trusted third party such as CAs. Secondly, even legitimate authorities (with a valid certificate) can not intercept the network traffic between client and server. For client-client verification we use offline verification settings. Offline verification could be done in several ways such as using QR codes, mnemonics and text-based verification. Details of offline verification techniques are out of scope of this paper.
- **Two-factor Authentication**: The server only requests username and password. All user data in the cloud is stored by encrypting with user password. Thus, even we use state-of-art scrypt key derivation function, security depends on the user password meaning a strong password is required for better protection against the issues in Section II. Also, two-factor authentication is required for clients to register. If a user registers first time, an SMS OTP is used for authentication. If a user is adding a new client to its account, OTP data is sent to its previous clients over the application itself.
- **Local Disk Encryption**: Any data that is stored on the client should be encrypted. This includes private keys. Methods used for this purpose is out of scope of this paper. But we assume all clients use local disk encryption.

III. The Protocol

In this chapter, we give the details of our protocol. In our assumption, two parties which want to communicate with each other have multiple devices and they are all connected to a centralized server infrastructure. All communication between devices are relayed over this infrastructure.

From the cryptographic point of view, restrictions of the mobile devices are important due to cryptographic operations are the major performance bottlenecks of the systems. Therefore, protocols and algorithms that have small footprint and fast implementations are needed for mobile environments. Specifically, asymmetric operations require most cpu power. Thus, we acted carefully while selecting the algorithms. Our cipher suite is as follows:

- Signing algorithm: Ed25519 [5]
- Key exchange algorithm: Curve25519 [6]
- Symmetric encryption algorithm: ChaCha20 [7]
- Message authentication code: Poly1305 [8]
- Hash function: Keccak [9]
- Key derivation function: scrypt [10]
- Operating system's Cryptographically Secure Random Number Generator (CSRNG) is used for random number generation.

We use the following notations throughout the paper.

- A and B are the public pseudonyms (usernames) for the parties (users).
- A client is the mobile device (and application in it) that the user is interacting with.
- A has m devices: $A_0, A_1, \ldots, A_{m-1}$
- B has n devices: $B_0, B_1, \ldots, B_{n-1}$
- S is the centralized server.
- $||$ is concatenation.
- MSB n-bit of m is the most significant n-bit of m.
- LSB n-bit of m is the least significant n-bit of m.
- e_X and d_X are Ed25519 public and private keys used for signing/verification operations of part X, respectively.
- $[m]_{d_X}$ is the signature of message m signed by part X.
- $\{s_m\}_{e_X}$ is the verification operation of signature s for message m with part X's public key.
- $H(m)$ is the KECCAK-256 hash function of message m.
- $KDF(pwd, salt)$ is the scrypt 512-bit key derivation function with user password pwd and 128-bit nonce $salt$.
- $EncMac_{K,IV}(m)$ and $DecMac_{K,IV}(m)$ are ChaCha20-Poly1305 authenticated encryption and decrypt-and-verify algorithms with message m, 256-bit key K and 64-bit initialization vector IV, respectively. The generated authentication tag is 128-bit and appended to the ciphertext.

A. Server Details

We assume that the centralized server can be deployed using cloud based services. As message communication backend we use XMPP [11]. XMPP handles most of the server-side deployment difficulties including distributed server infrastructure. We handle client registration, authentication and messaging. Our messaging format is converted to XMPP payloads and messages are delivered through XMPP. Communication between servers and clients are all TLS [12] secured with certificate pinning. Using certificate pinning avoids the need for a trusted third party such as CAs and ensures that nobody (even legitimate authorities) is doing a man-in-the-middle attack against the communication channel. A database stores informations about users. On the server side, we are storing the following information for every user:

- **Username** (username): A valid mobile phone number is used for identification of the user.
- **One-time Password State** (otp-state): Used for two-factor authentication.
- **Device List** (device-list): List of user's devices. Elements of the list contain three values: *device-id*, *device-data* and *session-data*.
- **Database Salt** (db-salt): Contains the salt value used in password-data
- **Hashed and Salted User Password** (password-data): Contains salted and hashed (with KDF) value of the local password data.
- **Encrypted Encryption Key** (eek-data): Contains the encrypted encryption key and IV used to encrypt long term and short term private keys.
- **Encrypted Long Term Private Key** (L_{priv}): Contains the encrypted Ed25519 long term private key. The data is encrypted with a key derived from user password.
- **Self-signed Long Term Public Key** (L_{pub}): Contains the self-signed Ed25519 public key used for signature verification. This data is public to requests.
- **Encrypted Short Term Private Keys** (S_{priv}): Contains an encrypted list of Curve25519 short term DH key exchange private keys. Elements of the list contain three values: *curve25519-private-key*, *key-uid* and *timestamp*. The list is encrypted with a key derived from user password.
- **Signed Short Term Public Keys** (S_{pub}): Contains a list of Curve25519 short term DH key exchange public keys. Elements of the list contain 4 values: *curve25519-public-key*, *key-signature*, *key-uid*, *timestamp*. This data is public to requests. Public and private key pairs are matched with the *key-uid* identifier which is the hash of *curve25519-public-key*.

Usage details about the information stored on the server will be given in the following subsections.

B. Client Registration

1) New Client Registration: When a new user A wants to register first time with the system, the protocol is as follows:

1) Using client, user A types its username and a password (pwd). The password itself is never sent to server. Instead, the fingerprint of the password is sent to server. To compute the fingerprint *local-pwd-data*, the client generates nonce *local-salt* and computes the password data and sends registration request to the server.

$$\textit{local-salt} = \text{LSB 128-bit of } H(\text{"314159265359"}||username)$$

$$K = KDF(pwd, \textit{local-salt})$$

$$\textit{local-pwd-data} = \text{MSB 256-bit of } K$$

$$A \xrightarrow{\text{"Register"},username,\textit{local-pwd-data}} S$$

2) Server checks if the user exists or not. If it exists than the request is rejected. If not, the server creates a new user in the database, generates a random salt value and fills *username*, *db-salt* and *password-data*.

$$\textit{password-data} = KDF(\textit{local-pwd-data}, \textit{db-salt})$$

Then, server randomly generates an *otp-state* and sends it to the user's username using an SMS gateway. Server also sends acknowledgment to the client and requests the verification code which is sent by the server.

3) When client gets the verification SMS, user writes the code into client application and sends the verification request to the server as follows:

$$A \xrightarrow{\text{"VerifyOTP"},username,\textit{otp-data}} S$$

4) Server checks if the *otp-data* matches with the *otp-state* in the database for the corresponding *username*. If it is, server generates *session-data* and *device-id* and sends an acknowledgment to the client, requests device information and long term and short term private and public key data.

$$A \xleftarrow{\text{"Welcome"},\textit{username,device-id,session-data}} S$$

5) Client saves the device-id and session-data in an encrypted format. They are used to authenticate the client later.
6) Client randomly generates random encryption keys and IVs K_1, K_2, IV_1, IV_2 for both long term and short term private keys. $K1, IV_1$ is used to encrypt long term private key data and $K2, IV_2$ is for short term private key data. Client randomly generates IV_E and encrypts the encryption keys using a form the key K which was derived from user password in step 1 as follows:

$$K_E = \text{LSB 256-bit of } K$$

$$\textit{eek-data} = EncMac_{K_E, IV_E}(K_1||IV_1||K_2||IV_2)||IV_E$$

Now, client generates long term Ed25519 public private key pair (e_A, d_A), encrypts the private one and signs the public one. Client also generates 1024 short term Curve25519 public private key pairs ($\textit{st-pub}_n$,$\textit{st-priv}_n$), encrypts the private ones and signs the public ones. All data is formed in a proper way to be stored on a

database. The storage format is up to implementation but using base64 and JSON would be a good choice.

$$L_{priv} = EncMac_{K_1,IV_1}(d_A)||IV_1$$

$$L_{pub} = e_A||timestamp||[H(e_A||timestamp)]_{d_A}$$

$$S_{priv} = \{EncMac_{K_1,(IV_1+n)}(\textit{st-priv}_n) \mid 0 \leq n < 1024\}$$

$$S_{pub} = \{\textit{st-pub}_n||timestamp|| [H(\textit{st-pub}_n||timestamp)]_{d_A} \mid 0 \leq n < 1024\}$$

7) Client generates *device-data* which is a unique identifier for this particular client. Then, sends device-data and encrypted datas to the server.

$$A \xrightarrow[\textit{eek-data},L_{priv},L_{pub},S_{priv},S_{pub}]{\textit{"RegisterData",username,device-data,session-data}} S$$

8) Server registers data into the database and a new user is created. Server returns final welcome acknowledgment.

2) Existing Client Registration: When an existing user wants to add a new client to its account, the protocol is as follows:

1) Assume user A has i clients, namely $A_0, A_1, \ldots, A_{i-i}$ and wants to add a new client A_i.
2) Using the new client, user writes its username and password (pwd). *username* and *password-data* (shown in Section III-B1) is sent to server with tag "AddClient".
3) Server checks if *username* is valid and corresponding *password-data* matches. If it is, a new *otp-state* is randomly generated and sent to user's previous clients $A_0, A_1, \ldots, A_{i-i}$.
4) When user gets OTP data from one of its previous clients, writes the code into new client's application. Client sends the verification data to the server.
5) Server checks if the *otp-data* matches with the *otp-state* in the database for the corresponding *username*. If it is, server generates a new device with *session-data* and *device-id* and sends an acknowledgment to the client.
6) Once new client gets the acknowledgment and session details, it requests the data set $\textit{eek-data}, L_{priv}, L_{pub}, S_{priv}, S_{pub}$ with session authentication.
7) Server sends the data set to the client and client generates the decryption keys from user password as mentioned in Section III-B1.
8) After verifying the data integrity, client stores the public and private key data, generates *device-data* and sends to the server.
9) Server stores the new client details and returns final welcome acknowledgment.

3) Client Authentication (Login/Logout): When a user logs out on a client, that client application sends a de-registration request to the server for that particular *device-data*. Or when user deletes the client application from the client and reinstalls it, session authentication fails and in both cases server deletes the corresponding device from the database. If user wants to add that client again to its account, the protocol is same as existing client registration mentioned in Section III-B2.

4) Client-to-client Communication: Assume that users A and B are registered with the server and A has m devices: $A_0, A_1, \ldots, A_{m-1}$, B has n devices: $B_0, B_1, \ldots, B_{n-1}$. A wants to start a secure chat session with B from its client A_i. All communication between clients are relayed over the server. When a message is sent to any of the user's client, it is relayed by the server to all clients that are associated with that user. We assume that A and B are friends and shared each other's long term public keys over the server and done the offline verification step as mentioned in Section II. Communication is established as follows:

1) A sends a chat request to server.

$$A \xrightarrow[\textit{from:A,session-data}]{\textit{"InitChatWith"},B} S$$

2) Server returns the next short term public key of B with its id.

$$A \xleftarrow{\textit{st-pub}_B,[\textit{st-pub}_B]_{d_B},\textit{key-uid}_B} S$$

Queuing of the DH key parts are handled by the server. When all key parts are used, server randomly returns key parts.

3) A verifies the signature of B's short term public key, randomly generates an IV_{AB} and selects its next short term private/public key pair $\textit{st-priv}_A$ and $\textit{st-pub}_A$. Then, computes the shared secret K_{AB} and sends the key exchange details to the server.

$$K_{AB} = KDF\big(Curve25519(\textit{st-priv}_A, \textit{st-pub}_B), IV_{AB}\big)$$

$$SKID = H(\textit{st-pub}_A, \textit{st-pub}_B, IV_{AB})$$

$SKID$ is used as session key ID and included within every message encrypted with K_{AB}.

$$A \xrightarrow[\textit{"KeyExchange",from:A,to:B,session-data}]{\textit{key-uid}_A,\textit{key-uid}_B,IV_{AB},EncMac_{K_{AB},IV_{AB}+1}(SKID)} S$$

From now on, A can send message m to B using the shared secret key K_{AB} as follows:

$$A \xrightarrow[\textit{"Message",from:A,to:B,session-data},timestamp,SKID]{EncMac_{K_{AB},IV_{Rnd}}(m||timestamp||SKID),IV_{Rnd}} S$$

The authentication tag generated by the $EncMac$ operation is also used for the message id (MID) to track messages by server. It does not matter if user B is online or not. All messages (including key exchange messages) sent by user A can be stored on the cloud and delivered when B becomes online.

4) Server gets the key exchange details and sends a push notification to the user B.
5) When B gets online, server first sends the key exchange details sent by the user A. Using these data (shown in step 3), B requests A's corresponding short term public key from the server, verifies the signature of the public key, generates session key, verifies the SKID and sends acknowledgment to the server, then starts to accept

messages. If verification fails, key exchange request is rejected.

$$K_{AB} = KDF\big(Curve25519(\textit{st-priv}_B, \textit{st-pub}_A), IV_{AB}\big)$$

6) B first decrypts than verifies the message m.

$$m = DecMac_{K_{AB}, IV_{Rnd}}(m||timestamp||SKID)$$

B does the same procedure to send a message to party A.

IV. Conclusion

We presented an end-to-end encryption scheme which allows multi-device synchronization and offline key exchange. Multi-device synchronization is achieved by backing up all private data on the cloud in a secure way therefore any new client for a user can easily fetch all private keys from the cloud. The private keys' security on the cloud is based on the user password and the password is never known by the server. Offline key exchange is achieved by storing precomputed key exchange parts on the cloud, thus, even if the user is not online, its key exchange data can be requested from the cloud. User authentication is done using two-factor authantication where user should write its password and OTP data. Our design has two protection methods against man-in-the-middle attacks. First one is public key pinning which secures the communication between the client and server. Secondly, client to client communication is secured by one-time offline verification. Also, using timestamp based message id prevents possible reply attacks.

Our protocol allows two-parties with multiple devices to communicate in a secure way but is lack of group chat. Thus, adopting group chat functionality without compromising security requirements into our design is the major future work for our research.

Acknowledgment

The authors would like to thank Ali Aydın Selçuk and Mutlu Çiçek for their support and assistance with this project.

References

[1] H.K. Taşkın, M. Demircioğlu, *Off-the-Record Communication with Location Hiding*, Information Security and Cryptology, Ankara, Turkey, Sep 2013.
[2] N. Borisov, I. Goldberg, E. A. Brewer, *Off-the-record communication, or, why not to use PGP*, WPES 2004: 77-84.
[3] P. Zimmermann, A. Johnston, Ed., J. Callas, *ZRTP: Media Path Key Agreement for Unicast Secure RTP*, Apr 2011, http://tools.ietf.org/html/rfc6189.
[4] Quarkslab, *iMessage Privacy*, HITBSecConf2013, http://blog.quarkslab.com/imessage-privacy.html.
[5] D. J. Bernstein, N. Duif, T. Lange, P. Schwabe, B. Yang, *High-speed high-security signatures*, Journal of Cryptographic Engineering 2 (2012), 77–89.
[6] D. J. Bernstein, *Curve25519: new Diffie-Hellman speed records*, Proceedings of PKC 2006.
[7] D. J. Bernstein, *ChaCha, a variant of Salsa20*, Workshop Record of SASC 2008: The State of the Art of Stream Ciphers.
[8] D. J. Bernstein, *The Poly1305-AES message-authentication code*, Proceedings of Fast Software Encryption, 2005.
[9] G. Bertoni, J. Daemen, M. Peeters, G. V. Assche, *The Keccak sponge function family*, http://keccak.noekeon.org/.
[10] C. Percival, *Stronger Key Derivation via Sequential Memory-Hard Functions*, BSDCan'09, May 2009, https://www.tarsnap.com/scrypt.html.
[11] *The Extensible Messaging and Presence Protocol*, http://xmpp.org/.
[12] T. Dierks, E. Rescorla *The Transport Layer Security (TLS) Protocol Version 1.2*, August 2008, http://tools.ietf.org/html/rfc5246.

Halil Kemal Taşkın Halil Kemal Taşkın (S'14) graduated from Department of Mathematics Teaching at Gazi University, Turkey in 2009. He then received his M.Sc. degree in Cryptography from Institute of Applied Mathematics at Middle East Technical University in 2011 and he has been continuing his Ph.D. studies at the same department since 2011. During his studies, he has worked on Block Cipher Cryptanalysis and involved in a project about Differential Cryptanalysis funded by TÜBİTAK. Since 2013, he has been working as Cyber Security Specialist in Oran Teknoloji, Turkey.

Murat Demircioğlu Murat Demircioğlu (S'14) was born in Istanbul, Turkey, in 1986. He graduated from Department of Mathematics at Middle East Technical University, Turkey, in 2009. He then received his M.Sc. degree in Cryptography from Institute of Applied Mathematics at Middle East Technical University, Turkey in 2011 and he has been continuing his Ph.D. studies at the same department since 2011. During his studies, he has worked on Quantum cryptography and also involved in a project about Differential Cryptanalysis funded by TÜBİTAK. He worked as a Research Assistant in Institute of Applied Mathematics of Middle East Technical University, Turkey, from 2012 to 2013. Since 2013, he has been working as Cyber Security Specialist in Oran Teknoloji, Turkey.

Salim Sarımurat Salim Sarımurat graduated from Computer Engineering Department of Bilkent University, Turkey, in 2011. He then received his M.S. degree from Computer Science and Engineering Department of Sabanci University, Turkey, 2013. During his M.S. studies, he was working on a project, funded by Scientific and Technological Research Council of Turkey (TÜBİTAK), to develop secure key predistribution methods for mobile and multiphase Wireless Sensor Networks that improves resiliency performance against node capture attacks. Salim currently works as an Information Security Consultant.