O²TR: Offline OTR messaging system under network disruption

Mahdi Daghmehchi Firoozjaei\textsuperscript{a,b}, MinChang Kim\textsuperscript{a}, JaeSeong Song\textsuperscript{b}, Hyoungshick Kim\textsuperscript{a,1,*}

\textsuperscript{a}Department of Electrical and Computer Engineering, College of Information and Communication Engineering, Sungkyunkwan University, Suwon, Republic of Korea
\textsuperscript{b}Department of Computer and Information Security, College of Software & Convergence technology, Sejong University, Seoul, Republic of Korea

\textbf{Abstract}

Providing a secure and efficient communication system under network disruption without a trusted third party remains a challenging issue. To develop a secure and efficient system in such situations, we extend the conventional Off-The-Record (OTR) protocol into a new protocol named offline OTR (O²TR). O²TR provides end-to-end security between users without requiring the assumption that they are persistently connected to each other. To show the feasibility of the proposed protocol, we implemented a prototype to support O²TR based on the Gajim XMMP instant messaging platform. Our experiments showed that O²TR can be used reliably even when the corresponding network party is temporarily broken down. Moreover, O²TR provides an efficient way to resume private sessions which is about 34% faster than the original OTR. We also proved the secrecy of O²TR using an automated verification tool called AVISPA.

© 2018 Elsevier Ltd. All rights reserved.

1. Introduction

When a disaster such as an earthquake, fire, and terrorist attack happens, secure and efficient communication systems are needed to manage the crisis. In such situations, constructing a reliable and efficient communication system for emergency requests and responses is a major challenge. Due to the high priority of rescue missions, the privacy of some victims may not necessarily be protected well. In general, people are concerned about their privacy associated with their physical and mental health, which include: stress, disability, and confusion, in a disaster. Moreover, the prevalence of global surveillance, including an untrusted third party or government monitoring, has caused much concern to many users (Channa and Ahmed, 2010; Daghmehchi Firoozjaei et al., 2017b; Rastogi and Hendler, 2017), and communication service providers generally make money from mining personal data (Lustgarten, 2016). Based on these facts, a generic emergency communication system is expected to provide confidentiality and integrity services in any condition.

Due to the rapid growth of smartphone-based applications and global trends toward online services, traditional emergency facilities face challenges handling the types of communication that consumers expect today. In comparison to

\textsuperscript{*} This research was supported by the MSIT (Ministry of Science, ICT), Korea, under the ITRC (Information Technology Research Center) support program (IITP-2018-2015-0-00403) supervised by the IITP (Institute for Information & communications Technology Promotion). This research was also supported by the ICT R&D program (No.2018-0-00532).

\textsuperscript{1} Corresponding author.
E-mail address: hyoung@skku.edu (H. Kim).
https://seclab.skku.edu/people/hyoungshick-kim/
0167-4048/© 2018 Elsevier Ltd. All rights reserved.
the traditional communication systems, online services, such as instant messaging (IM), provide different features that include: texting, picture/video sharing, location-based services, and most importantly, real-time delivery. The next generation 9-1-1 (NG9-1-1) project, for instance, will allow callers to take advantage of mobile device features, such as IM systems in an emergency situation. Since IMs have no security features by default, some security mechanisms such as client-to-server encryption via TLS or end-to-end confidentiality and integrity via SCIMP (Moscaritolo et al., 2012), TextSecure (Frosh et al., 2016) or off-the-record (OTR) (Borisov et al., 2004) are used to provide secure messaging (Frosh et al., 2016).

Basically, OTR provides an authenticated and confidential channel for an online conversation, but the content of the conversation is repudiable afterward (Liu et al., 2013). Besides pairwise encryption and mutual authentication, OTR also provides advanced security properties, such as perfect forward secrecy (PFS) and deniability. To have end-to-end confidentiality and integrity with messaging, a novel fine-grained key refreshing feature called OTR ratcheting is used. Since it involves asymmetric (public key) cryptography, it is known as asymmetric ratcheting. To this end, OTR clients continuously establish a fresh Diffie–Hellman (DH) ephemeral shared secret in each round trip message (Cohn-Gordon et al., 2017). Moreover, by revealing the used authentication keys, OTR mimics the features of an actual face-to-face conversation in a virtual environment (Daghmehchi Firooza et al., 2017a; Liu et al., 2013; Rastogi and Hendler, 2017). These features make the OTR protocol a proper solution for privacy preservation in online messaging services.

In general, asynchronous messaging, where the sending and receiving of a message do not need to happen simultaneously, is a normal phenomenon in the most IM systems. Since each key is securely chained to the previous key in the current session, the applicability of OTR is limited to synchronous messaging in which both conversation parties are online (Frosh et al., 2016). In spite of the security features of OTR, this limitation makes it an improper choice for emergency systems that should support any kind of communication. For instance, if a disaster leads to an unstable communication link, such that the receiver goes offline and becomes unreachable, any offline encrypted messages will be unreadable.

Despite a relatively limited adoption of OTR, its ratcheting technique can be seen in other security protocols (Cohn-Gordon et al., 2017). TextSecure (Frosh et al., 2016) and its successor, Signal(S) protocol, applied the idea of the ratchet technique in OTR to the key update protocol (Cohn-Gordon et al., 2017; Marlinspike, 2013). Signal provides an end-to-end encryption as well as PFS and future secrecy in synchronous and asynchronous messaging environments. The double ratcheting algorithm (DR algorithm) (Perrin and Marlinspike, 2016) has been used to update the keys with an initial shared secret (also known as root key) (Rosler et al., 2018). The initial shared key between two parties is calculated with X3DH key agreement protocol (Marlinspike and Perrin, 2016) that uses static and ephemeral DH shares of both parties. The public keys of each party should be uploaded to the server at the registration phase (Marlinspike and Perrin, 2016; Rosler et al., 2018).

This key distribution mechanism requires a trusted third party who is responsible for managing the public keys of parties in a secure manner (Marlinspike and Perrin, 2016). In comparison, the end-to-end confidentiality and integrity provided by OTR independent of any key distribution server make us consider this protocol as the better solution for privacy-preserving purposes.

To provide a secure and efficient messaging system for emergency communications, we extended the original OTR to create a new protocol named offline OTR (O²TR). Basically, the ephemeral key exchange in OTR is based on a three-step ratchet as a mechanism of advertising keys and receiving confirmations for those keys in subsequent messages (Marlinspike, 2013). After establishing a private session, OTR parties exchange their next ephemeral secrets in each data packet. In fact, the key materials in OTR are chained together. O²TR utilizes this feature to handle offline messages. Our experiments showed the complete practicability of the O²TR-based system to handle offline messaging. Although the processing time to handle O²TR messages is almost the same as OTR’s, O²TR provides faster private session resumption. Overall, the O²TR-based system does not need any changes in the existing messaging infrastructure and is able to retrieve the offline messages during a network disruption in an emergency communication.

The main contributions of this paper are summarized as follows:

- We introduced a secure and reliable messaging system based on the O²TR protocol for emergency requests. The proposed system provides the end-to-end confidentiality and integrity services in which the service provider has no role in the credential management. In this system, the user’s privacy is completely preserved in the presence of a curious server. Furthermore, the system reliably retrieves any offline messages encrypted in the last disconnected private session.
- To address the limitation of OTR in the asynchronous messaging, we design a mechanism in O²TR to regenerate the session keys of the last interrupted private session to handle offline messages. We develop a solution to renew and replace the stored keys based on the incoming and outgoing messages.
- To evaluate the performance of our model in a real case, we implemented O²TR on the Gajim IM system launched on an XMPP/Jabber platform. We also proved the secrecy of O²TR using the AVISPA automated verification tool.

The rest of this paper is organized as follows. In Section 2, an overview of OTR is presented. Section 3 defines the problem and explains the threat model. The O²TR protocol and its model are explained in Section 4. The implementation of O²TR and the evaluation results are respectively explained in Sections 5 and 6. In Sections 7 and 8, we analyzed the security of O²TR, summarized our conclusions, and defined our future work, respectively.


2. Overview of OTR

Initially, Borisov et al. (2004) introduced OTR protocol in 2004. OTR is built on a high-level cryptographic abstraction, in which AES (128-bit), SHA-256, and SHA-1 hash functions are used in the latest version of OTR, Ver. 3 (Goldberg et al., 2005). In this protocol, SHA-256 is used in the handshake process and each encrypted message is authenticated by SHA-1 as an HMAC function (Di Raimondo et al., 2005). Furthermore, all exponentiation operations are done modulo a particular 1536-bit prime (Goldberg et al., 2005). Basically, the authentication process in OTR is performed after setting up a secure channel. In practice, OTR's parties first perform an unauthenticated DH key exchange to set up an encrypted channel. Afterwards, they implement a mutual authentication (Goldberg et al., 2005). A version of SIGMA is used as an authenticated key exchange (AKE) and both parties, Bob and Alice, use their long-term authentication public keys, PubA and PubB.

To generate DH keys, the values of and need to be at least 320 bits, and they are randomly picked. As shown in Fig. 1, in the first AKE message, to prevent any man-in-the-middle (MitM) attack, the encrypted value of Bob's DH exponent (which is hashed with SHA-256) is sent to Alice. When Alice replies, she sends her DH encryption key (which is hashed in an unencrypted form). From the third message, an encrypted channel is opened. All encryption keys, MAC keys, and DH's shared value, are computed by hashing (SHA-256) the DH's shared value, . The details of generating those keys are explained in Goldberg et al. (2005). After exchanging DH secrets, the parties authenticate each other by their long-live DSA public keys, which are used only for authentication (Alexander and Goldberg, 2007; Goldberg et al., 2005), in an encrypted channel. By verifying and , the authentication will be performed at the end of the handshake process.

To provide a secure data exchange, a key derivation function (KDF) is used to calculate the sending/receiving encryption and authentication keys based on DH shared key; s. KDF generates two 128-bit AES encryption keys, which are the “sending k_s” and the “receiving k_r,” and two 160-bit SHA1-HMAC keys, which are the “sending k_m” and the “receiving k_m,” for each message (Goldberg et al., 2005). To compute s in data exchanges, both sides use the keyids listed in the data

Fig. 1 – Handshake process in OTR.
Computes: \( s = (g^r)^y \)

\( k_e = kdf(s) \& k_m = hash(k_e) \)

Old MAC keys: \( K_j^* \)

Picks \( ctr \Rightarrow (\text{key}_A, \text{key}_B, ctr) \) be unique for each message

\[
T_A = (\text{keyid}_A, \text{keyid}_B, \text{next\_dh}, \text{ctr}, \text{AES} - CTR_{k_e,ctr}(\text{msg}))
\]

\[
T_A, MAC_{k_m}(T_A), K_j^* + 1
\]

\[
T_B, MAC_{k_m}(T_B), K_j^* + 2
\]

Fig. 2 – Data exchange in OTR.

message to select the ephemeral DH keys. Since the keys are changed per each message, the keys are used to make sure that a unique set of keys is being used (Alexander and Goldberg, 2007).

To provide the feature of PFS, a fine-grained key refreshing is achieved by the OTR ratcheting in each messaging round. As shown in Fig. 2, in any data message, a new DH secret, in the next\_dh, is suggested for the next data message. This DH secret is MAC protected and cannot be used until being acknowledged by the recipient. In each data message, a value of the counter, ctr, is selected such that the triple \((\text{key}_A, \text{key}_B, \text{ctr})\) is never repeated in a private session (Goldberg et al., 2005). To provide for the deniability, MAC keys are revealed one round later to the public. Because of this feature, once the old MAC keys, \( k_j^* \), are released, anyone can modify the message and the receiver cannot prove the sender’s authorship. Since the MAC keys are not revealed until their corresponding public keys are being discarded, there is no danger of accepting a message as valid which uses a MAC key that has already been revealed (Goldberg et al., 2005).

Fig. 3 shows the format of OTR data packet in which the header shows the protocol version, the message type, the sender and the receiver of the message as well as the required flags. In the payload, all fields except \( k_j^* \), which is released publicly, are authenticated. The MAC value of the keyid of the sender and the receiver, next\_dh, ctr, and Encrypted data is settled in the Authenticator field. The Encrypted data field consists of the encryption of the message and TLV (type/length/value) records, if needed. Generally, TLV records are used for special purposes besides the message exchanging (Goldberg et al., 2005). The MAC keys which were used before are revealed publicly in the Old MAC keys field.

3. Problem definition and threat model

In this section, we first discuss the privacy issues in a messaging system and key requirements for emergency communication. Moreover, we present the threat model and our motivation.

3.1 Privacy concerns

In messaging systems, users’ personal data (e.g., profile, private messages/photos, and location information) can be disclosed when its message database is accessed by unauthorized users. End-to-end secure communication is not sufficient for protecting user privacy. User’s personal information is typically stored in a database on the messenger server. As a consequence, user’s personal information at the server would become one of the most attractive targets for intelligence agencies (Lustgarten, 2016).

Due to the priority of the rescuing purposes, a client’s privacy may be ignored unintentionally in order to have more accurate emergency reporting. Therefore, providing an end-to-end confidentiality and integrity with no interference of any third parties or service providers is required to protect clients’ privacy and data confidentiality.
3.2. Retrieve undelivered messages

Due to the nature of emergency situations, efficient data needs to be delivered immediately and accurately to the rescue service centers. In this view, every tiny clue may change a victim's fate in a life-threatening condition. Therefore, each emergency request should be considered carefully. As a basic rule of all emergency rescue facilities, it is required that: they do not ignore any request. To get some insight into the problem, consider that a network disruption, whether it was planned (e.g., upgrading Reitblatt et al., 2012 or a service disruption Xu et al., 2015) or unplanned because of a natural disaster, occurs in which the server is unable to deliver the emergency requests. What would happen with an undelivered emergency request? Perhaps that message was the last chance for a victim to send his request. Therefore, any emergency facility requires a reliable messaging system that guarantees to save messages and delivers them meaningfully to the receiver.

3.3. Threat model

Compromising user’s privacy is considered as the main security threat in our model. To define the threat model, we assumed the messaging service provider is a curious provider that is able to monitor message traffic and open the messages before being delivered to the receiver. As shown in Fig. 4, in this scheme the service provider performs credential management and provides a secure channel to deliver the clients’ messages. This capability makes a curious service provider able to freely review all messages on the server side.

In this threat model, no messages were considered dropped and the server delivers all the messages, but it is assumed that they are reviewed or manipulated by the server.

3.4. Motivation

OTR secure channel prepares end-to-end confidentiality in which a curious server has no role or interference over the credential management. Therefore, message retention on the server side cannot compromise the system’s security (e.g., key extraction). Because of OTR ratcheting, this protocol is only applicable in synchronous communications (Daghmehchi Firoozjaei et al., 2017a; Frosch et al., 2016). Based on this limitation, an OTR-based system does not seem to be the appropriate choice for any network disruptions that lead to unstable connections. When a receiver goes offline and loses an on-the-fly OTR session, an offline message in the server won’t be readable when the receiver gets back online and starts a new OTR session. This case of communication seems to be possible in the most unwilling disruption cases, such as natural disasters or man-made violations. This situation gets worse when a victim does not have a chance to resend his emergency request. Therefore, to address the limitation of OTR to handle offline messages, we introduced O²TR protocol to retrieve any undelivered messages.

Basically, O²TR is an extension of the original OTR to cover asynchronous messaging. In addition to the basic features of OTR, O²TR supports offline messaging by exploiting the key refreshing property of OTR.

4. Offline OTR (O²TR)

In OTR, the authentication state and the message state are used to indicate the current state of each conversation. The authentication state shows the authentication progress (e.g., none, awaiting_dhkey, and awaiting_sig). While the message state indicates the ciphering state of the messaging (e.g., plaintext, encrypted, and finished) (Goldberg et al., 2005). To prevent any flaw and inconsistency in sending messages, the message state controls what happens during the OTR conversation. For instance, if during a private session a party logs out his OTR client the correspondent will be notified to switch to msgstat_finished mode. This notification prevents a conversation party from unwillingly sending any message in plaintext mode (Goldberg et al., 2005).

Based on this structure, what would happen if an OTR client accidentally loses a private session (e.g., due to connection lost)? Since the OTR private session depends on the
running IM chat room, the message state mismatching occurs when a client is disconnected and leaves the chat room while its OTR session has not finished. In this case, the current private session is not terminated on the other side. Therefore, any message sent by the correspondent is based on the previous unfinished private session. An unrecoverability error will occur when the client receives an offline message from the previous unfinished private session.

As long as an OTR party, as receiver, gets disconnected, any undelivered message that remains on the server side is considered an offline message. Although the offline message will be delivered when the receiver goes back online, it will not be readable. The delivered message was encrypted by the keys that have already expired. The new connection requires a new OTR session that requires a new handshake and new parameter sharing. In other words, the synchronous communication dependency of OTR makes it impossible to decrypt the offline messages.

To prevent any parameter mismatching in the asynchronous communication, we introduce the O^2TR to handle the offline messages. Especially, O^2TR can guarantee that any offline emergency requests are securely decryptable after any network disruption. As stated earlier, the ciphering keys are renewed with each message exchange on both sides by OTR ratcheting. Based on the suggested DH secret share for the next ephemeral key, each party can generate the key for the next message pair. By retaining these shares in O^2TR, we are able to retrieve any offline message. In this case, a conversation party can authenticate and decrypt an offline message that was created in the last disconnected session.

In the original OTR, practically all the keys are separately generated on each side of the conversation and the keyids are used to identify which pair of exponents were used in any given message. Since the keys are changed per each message pair and in order to use a unique set of keys (Alexander and Goldberg, 2007), the keyid is increased by generating a new DH key. This sequence restarts in each new session, and it is impossible to accept a message with a different keyid from a past session. To handle this impossibility, we considered the keys independently of their keyid when an urgent event is detected. In our emergency communication model, the urgent event is defined as a condition when a client unwillingly loses a private session without any signaling (i.e., without any sign of msgstat_finished mode).

4.1. Model description

Basically, the O^2TR model consists of two extra parts in comparison to the original OTR, emergency key management (EKM) and offline message detection (OMD). As shown in Fig. 5, the ephemeral keys generated by KDF are saved in EKM as well. These keys are generated based on the DH secrets of both parties, next_dh and aur_dh. Since the keys are renewed for each O^2TR message, the registered keys in EKM are replaced with new ones. Therefore, in each conversation EKM consists of the latest keys for both O^2TR parties. In fact, EKM retains the ephemeral keys one more round even if the O^2TR session is finished or interrupted.

To handle the offline message in O^2TR model, OMD checks the state of the incoming message. In the case of an offline message, an encrypted message is delivered to the client with no private session. Therefore, the state of the offline message (in msgstat_encrypted mode) does not match with the state of the receiver, which is in plaintext mode. As shown in Fig. 5, by detecting this mismatch in OMD, the message is considered an offline message and is handled by EKM stored keys instead of being rejected. The offline message is authenticated and decrypted to extract the plaintext by EKM driven keys. This situation is critical with an emergency communication when the victim unwillingly loses a private session. In fact, O^2TR gives the receiver the ability to handle an offline message without a live private session. Since the offline message is created by the last unfinished O^2TR session, no MAC keys are released and they are still valid. Therefore, it is impossible to compromise the validity of the offline message before releasing its MAC key in the next data message (Goldberg et al., 2005).

In response to the offline message, the receiver refreshes the O^2TR session to prevent MitM and replay attacks. This refreshing is based on the secret shared by the next_dh in the offline message. In fact, O^2TR parties resume the interrupted last private session without the need to perform a complete handshake process.

Consequently, O^2TR not only retrieves the offline message but also speeds up the private session establishing progress. Fig. 6 shows how O^2TR resumes the previous private session without any interruption. By retrieving the saved keys of the last private session from EKM, the receiver is able to verify and decrypt the delivered offline message. By selecting a new DH secret (x) and the received y, which is in the next_dh field of the offline message, a new DH key; s = (g^xy), is generated.

5. Implementation

To implement O^2TR model, the Gajim messaging application was selected as an IM system. Furthermore, an ejabberd XMPP server was used to provide an XMPP/Jabber platform to launch the Gajim IM. Due to the ability of ejabberd to provide secure client-to-server connections (i.e., SSL/TLS connection) and a storage for retaining undelivered messages, this server was selected for our implementation. The credential management authority allows the ejabberd server to freely review and scrutinize all messages. Based on this, we considered the ejabberd XMPP server as the curious server for our threat model. To enable OTR for the Gajim XMPP/Jabber client, we used the OTR plugin implemented by K. Braden available on GitHub. Furthermore, the Pure-Python-OTR, a Python OTR implementation package, also available on GitHub, was installed to enable the OTR plugin. To implement O^2TR in the Gajim XMPP/Jabber client, we optimized the Pure-Python-OTR package and OTR plugin to add EKM and OMD modules.

During each message handling, sending and receiving, the latest keys are saved by EKM module. These keys consist of sendenc, sendmac, recvenc, and recvmac, which are generated for each message. The first two keys are the sender’s keys for the

---

5 https://www.process-one.net/en/ejabberd/
Fig. 5 – O²TR model.

Fig. 6 – Private session resumption in O²TR.
current sending message and the second two keys are the keys related to the message that will be received in the next round.

By detecting an offline message in OMD, the receiver handles this message with the rcvenc and rcvmac keys that are available on EKM. As shown in Fig. 6, by verifying and decrypting the offline message the correspondent’s DH secret, y, is available to make a new DH key, s. Based on this, EKM helps O²TR parties to continue the chain of ephemeral keys without any interruption. Having resumed an O²TR session, the conversation is continued privately.

6. Experiments

To evaluate the performance of the O²TR-based messaging system, we tested the reliability of the messaging system to handle offline messages as well as the processing time. For a clear comparison, these items were scrutinized by the IM communication between the Gajim XMPP/Jabber clients in two models: OTR and O²TR.

6.1. Messaging reliability

To evaluate the reliability of O²TR in offline message handling, we considered an unwilling event that caused a disconnection of a normal Gajim IM conversation. This event was carried out for more than 200 separate conversations. As a result, all the offline messages were recovered (100%) and decrypted completely with the Gajim-O²TR while it is not possible in the normal Gajim-OTR model. Therefore, the suggested O²TR-based messaging system provides a full messaging reliability in the cases of a network disruption and a private session interruption.

Based on the time of disconnection, our experiments showed that different conditions happen to handle an offline message. These conditions are different when an O²TR party goes offline after sending a message or goes offline after receiving a message. Depending on whether a message is being sent or received, the arrangements of the keys are different. For instance, if several O²TR messages are received without any message being sent, the rcvenc and rcvmac keys should be updated respectively in EKM and vice versa. To prevent any key mismatching, O²TR runs KDF and updates EKM after handling each message, whether it being sending or receiving. Generally, the ejabberd server respectively forwards messages based on the time of origin. Since the ephemeral keys are chained to the previous keys, our experience showed that O²TR successfully handles any number of offline messages retained on the messaging server.

6.2. Processing time

To provide a practical comparison, we evaluated the processing time of the message exchange, the handshaking and the session refreshment in both protocols of O²TR and OTR. Fig. 7 depicts the processing time to handle the incoming and outgoing messages in OTR and O²TR protocols. Unlike outgoing messages, handling incoming messages takes more time due to performing some additional tasks. To verify and handle an incoming message, the receiver scrutinizes the current policies (e.g., accepting encrypted or non-encrypted messages), the message type (data packet or AKE message), OTR’s parameters (e.g., version and validity), and the key IDs, and then it authenticates the received message (Goldberg et al., 2005).

Fig. 8(a) shows the processing time to handle the offline incoming messages, which is the main feature of O²TR. In comparison to normal incoming messages, an offline message takes more time to be handled. Detecting the offline message and extracting the saved keys from EKM are the main reasons for this time difference. We noted that if an O²TR party receives multiple offline messages, the processing time to handle the first message is longer than the processing time for the rest of the offline messages. Unlike the first offline message, which needs to extract the saved keys, the keys required to handle the remaining messages are prepared differently. Since all offline messages are encrypted with the same key in the multiple offline messages, they can be decrypted with the same encryption key. Despite the fact that they have different authentication keys and ctr, the encryption key for the first offline message, extracted from the saved keys, can be used for other offline messages. Therefore, for the second and subsequent offline messages, the processing time is significantly shorter. Based on our experiments, the
average processing time to handle the first offline message by O²TR is 29.95 ms, while the second offline message takes 9.53 ms on average for processing.

As a significant feature of OTR, the conversation parties are able to refresh the current private session and renew the DH keys anytime during their conversation. Based on this, we provided a condition in order to resume the private session in O²TR, when an offline message is received, instead of starting a new handshake, which takes longer. Fig. 8(b) shows the processing time for both the OTR handshaking process and the O²TR private session resumption. On average, an OTR handshake takes 58.29 ms while the resumption of the private session in O²TR takes 43.49 ms. Based on these results, in O²TR, a private session is refreshed 34% faster than creating a new private session in OTR.

Table 1 compares the average processing times to handle all messages in OTR and O²TR models. To clarify the difference between the distributions of the processing time in both models, we performed a Mann–Whitney U test. As shown in Table 1, there is a statistical difference in the processing time for handling the incoming messages in OTR and O²TR ($p < 0.001$) while we failed to show a difference for handling outgoing messages ($p = 0.0867$). Based on this result, the average processing time to handle the outgoing messages are almost same in both models but handling the incoming messages in O²TR is averagely longer. In comparison to the incoming message handling, the processing time to handle the offline messages in O²TR is almost two times longer.

### 7. Security analysis

Centrally distributing public keys leaves open the possibility that the key distributing server itself could carry out MitM attack, if it gets compromised or is under the coercion of the law enforcement authorities (Bai et al., 2016; Shirvanian et al., 2017). Security code verification has been used in several applications (e.g., Signal systems), to mitigate MitM attack. In this case, the usability challenges (e.g., false reject or communication delay), caused by users’ less knowledge or application’s incorrect delay, may lead to serious security problems. For instance, an inadequate warning may lead to aborting to reestablish the secure session after MitM attack detection (Schröder et al., 2016).

In Signal systems, each party generates a set of DH keys containing (Cohn-Gordon et al., 2017; Marlinspike and Perrin, 2016; Schröder et al., 2016):

- a long-term identity key
- a medium-term signed pre-key
- a set of one-time pre-keys

In the registration phase, the Signal party uploads the public keys corresponding to these values to the key distribution server, together with a signature on medium-term signed pre-key (Cohn-Gordon et al., 2017). The server retains this set of keys to allow other parties to establish a private session with this client. To perform a key agreement and a private session, a Signal party contacts the key distribution server and fetches a pre-key bundle from the receiver. After verifying the pre-key signature, the Signal party generates an ephemeral key pair with the public key (Marlinspike and Perrin, 2016). Registering the pre-keys to the intermediate server makes a Signal party able to establish a private session even when the receiver is offline.
Despite the explosive uptake of Signal (e.g., Signal\(^8\) and WhatsApp\(^9\)), there are some issues in the usability and security aspects. The lack of clear instructions for code verification and insufficient information related to the possible consequences of users’ actions lead to decrease the verification success and underestimate the actual risks (Mujaj, 2017; Schröder et al., 2016). To decreases the human errors in code verification, a semi-automated code verification is provided by Signal. In this case, the received code is copied to the clipboard by user and the application verifies it by comparing with the locally generated code (Shirvanian et al., 2017). Despite this improvement, granting access to the clipboard opens the possibility of access to the private data and side channel issues for Signal users.

In the view of security, the server trust is the main security issue of the Signal protocol. Refusing to hand out one-time pre-keys by a compromised server causes forward secrecy for the secret key to depend on the signed pre-key’s lifetime (Marlinspike and Perrin, 2016). This reduction in the initial forward secrecy could also happen if one party maliciously drains another party’s one-time pre-keys or impersonates himself with a real client’s identity key along with the cooperation of a compromised key distribution server (Marlinspike and Perrin, 2016; Schröder et al., 2016). Unlike the Signal protocol, both OTR and O\(^2\)TR need no third party for key distribution purposes. Implementing the secure sessions independently causes the O\(^2\)TR’s parties do not face these kinds of security issues.

To evaluate the security properties of an O\(^2\)TR-based messaging system, we analyzed the confidentiality provided in the presence of an untrusted messaging service provider. Based on our threat model, the adversary is able to log the messages and monitor their traffic. The capability of the attacker to retain the messages leads us to consider the ciphertext-only attack (COA) as the attack model for this analysis. In this attack model, the attacker can stop the message delivery and force the sender to send messages encrypted with the same key. On the other hand, the attacker has information about the plaintext, which is useful. For example, the message may consist of English words encoded in ASCII codes or a JPEG image. The attack is successful if the corresponding plaintext can be decrypted or if the key is derived.

As one of the basic security properties of OTR as well as O\(^2\)TR, ephemeral key exchanges are used to generate unique session keys for each message. Based on the chain of key generation in OTR, each data packet consists of a new DH exponent suggested by the sender to generate a DH key that is to be used for the next message. In practice, the key cannot be used until the receiver has acknowledged this advertised key. As a result, if an OTR party needs to send multiple messages before receiving any replies, he/she needs to keep using the current key and keep advertising the same key (Frosch et al., 2016; Marlinspike, 2013). This condition may lead to a possible flaw when a curious messaging server, as an active attacker, filters the packets and forces the client to send all messages with the same key. In this regard, retaining offline messages encrypted with the same key is a possible issue for the suggested O\(^2\)TR-based system.

Basically, the attack of COA is aided when the attacker has multiple pieces of ciphertext generated from the same key (Tipton and Krause, 2007). The two-time pad (Boneh and Victor, 2016) problem shows the insecurity of a stream cipher when the same stream cipher key is reused. In this situation, the ciphertexts generated from two messages; \(m_1\) and \(m_2\) from a key \(s\):

\[
c_1 \leftarrow m_1 \oplus G(s) \text{ and } c_2 \leftarrow m_2 \oplus G(s)
\]

where \(G(s)\) shows the key-stream, can be XORed to recover \(m_1 \oplus m_2\) (Diesburg et al., 2008):

\[
c_1 \oplus c_2 = (m_1 \oplus G(s)) \oplus (m_2 \oplus G(s)) = m_1 \oplus m_2
\]

Not surprisingly, English text contains enough redundancy that given \(m_1 \oplus m_2\) the adversary can recover both \(m_1\) and \(m_2\) in the clear (Boneh and Victor, 2016). For instance, Diesburg et al. (2008) showed that English plaintexts can be automatically extracted from two-time pads via the simple application of a Bloom filter (Bloom, 1970).

To avoid any birthday collision, an upper bound on the number of key-stream blocks that can be generated from a single key is assigned based on the block size (McGrew, 2002). For instance, for modern block ciphers, like AES, in which has a block size of 128, the birthday bound is \(2^{64}\) blocks, which corresponds to a whopping 1 million petabytes of data (Gueron and Lindell, 2017), which is far from practical to be breakable (Biryukov et al., 2010; McGrew, 2002). For proof of this boundary, we refer to the security margin defined by Bellare et al.’s previous results (Bellare et al., 1997) for a block cipher. The attacker’s capability to distinguish the ciphertext from a random stream is defined as the advantage function (\(\text{Adv}[\cdot]\)). Therefore, \(\text{Adv}[\text{PRF}]()\) and \(\text{Adv}[\text{PRP}]()\) show the advantage of the attacker to distinguish respectively a pseudorandom function (PRF) and a pseudorandom permutation (PRP) from random. Based on this definition, in a block cipher with \(l\) bit-length of cipher size:

\[
\text{Adv}[\text{PRF}](t, q) \leq \text{Adv}[\text{PRP}](t, q) + q^2 2^{\frac{l-1}{2}}
\]

Where \(q\) is the number of known plaintext blocks; \(t\) indicates the time used in the attack, and the advantage of the attacker to distinguishing \(X\) from random is denoted as \(\text{Adv}[X]\). By assuming \(\text{Adv}[\text{PRF}](t, q) = \alpha\) and \(l = 128\) for AES, it can be assumed that \(\text{Adv}[\text{PRP}](t, q) = 0\) since AES cannot be distinguished from a PRP (McGrew, 2002). Based on this:

\[
q = \sqrt{\alpha} \cdot 2^{64.5}
\]

Therefore, the adversary’s advantage is limited to \(\alpha\), if the generated key-stream blocks are less than \(2^{64}\) (McGrew, 2002). Despite this theoretical boundary, the actual amount is less in reality. The standard NIST recommendation is to stop using a key when the probability of leakage, which is an advantage to the adversary, exceeds \(2^{-32}\). Thus, after encrypting \(2^{48}\) blocks (i.e., \(2^{32}\) messages of length \(2^{16}\)), keys must be changed (Gueron and Lindell, 2017). An O\(^2\)TR message is encrypted with an AES-128 in the counter mode. In each data

---

8 https://signal.org/
9 https://www.whatsapp.com/
message, a value of the counter, \( c \), is selected such that a unique triple \((key_a, key_b, c)\) can be used (Goldberg et al., 2005). In a practical view, the standard implementation of AES-128 uses a 96-bit IV, and thus collisions occur in the IV with a probability \( 2^{-96} \) after encrypting only \( 2^{32} \) different messages (with fresh random IVs) (Gueron and Lindell, 2017). Therefore, the actual upper bound to use a key to encrypt several messages in AES-128 is counter mode is \( 2^{32} \) messages. Consequently, the upper bound of the messages that can safely be retained in the offline messages queue in the suggested \( O^2TR \)-based messaging system is defined as:

\[
k \leq 2^{32}
\]

(5)

Wherein \( k \) shows the number of messages encrypted with the same key in a private session. Although this bound seems unreachable in the offline message case, in our model we suggested to define a practical threshold (e.g., \( k = 1024 \)) to use an encryption key to do multiple encryption rounds. By this limitation, no \( O^2TR \) party can use the same encryption key more than \( k \) times for multiple messages. Otherwise, sending is paused until receiving a reply or refreshing to a new private session. In this case, the attacker has a limited amount of ciphertext and achieves a smaller chance for key extraction.

Unlike the Signal protocol, where the key distribution server buffers the pre-keys (Cohn-Gordon et al., 2017), in \( O^2TR \) the required keys are saved in the local memory and are refreshed in each message handling process. By assuming a safe local environment that does not have any unauthorized access, there is no key derivation possibility for the messaging server. Furthermore, authenticating \( O^2TR \) data packet’s fields (e.g., keyid, next_dh, and etc.), prevents any unauthorized manipulation during transmission. Based on this, no MitM attack can be considered by the curious messaging server in the proposed model. On the other hand, since no ephemeral keys are reused, the property of PFS is not compromised. Based on this fact, the attacker has no chance to decrypt the private session key even by logging the revealed MAC keys. Consequently, any kind of replay attack is not practical to compromise \( O^2TR \)-based messaging system.

To verify the safety of \( O^2TR \), we exploited the AVISPA\(^{10} \)\) cryptographic protocol verification tool. The AVISPA tool provides a suite of applications for the purpose of analyzing formal models of the security protocols. AVISPA provides a language called the high-level protocol specification language (HLPSL) (Chevalier et al., 2004) to describe the security protocols and specify their intended security properties, as well as be used as a set of tools to formally validate them. HLPSL is a modular and role-based language that allows for the specification of complex control-flow patterns, data-structures, and different intruder models (Chevalier et al., 2004). To implement the AVISPA verification in a virtual environment, we used a SPAN\(^{11} \) animator, a security protocol animator for HLPSL specifications (Genet, 2015).

Basically, SPAN translates the CAS\(^{12} \) language into an HLPSL specification and includes a local graphical interface similar to the one of AVISPA. The CAS\(^+ \) language represents the implementation of a security protocol and its components (e.g., identifiers, messages, session, and intruder’s knowledge and goal) in a set of simple phrases and symbols. The evaluation of AVISPA on SPAN animator indicates our protocol as a safe security protocol. In that evaluation, it was verified that both of the goals of the intruder, which are to detect message text and session ID, were unsuccessful. In those attack models, the adversary aims to decrypt the message and extract the session ID to recover the encryption keys respectively. Based on our threat model, the adversary’s knowledge is limited to IDs of the sender and receiver and their public keys (in the handshake stage). Our HLPSL evaluation codes are presented in Appendix A.

8. Conclusions

In this paper, we extended OTR into a more generalized protocol to provide a secure and efficient communication even when the base network system is disrupted. Our proposed system provides both confidentiality and integrity of the messages exchanged between clients with no need for any key distribution entity. In this model, the curious server cannot compromise the user’s privacy despite the capability of message retention. We analyzed the possible security threats and showed that the proposed system is secure against such threats. To evaluate the safety of \( O^2TR \), we proved its secrecy using the AVISPA tool implemented on a SPAN animator.

While the OTR protocol has a significant limitation in asynchronous messaging, the \( O^2TR \)-based system can effectively be deployed in both synchronous and asynchronous messaging environments. To retrieve any offline emergency messages, \( O^2TR \) uses a mechanism to regenerate the session keys. In this case, the proposed model guarantees to retrieve any undelivered messages due to a lost connection or a session interruption. Our experiments showed its complete capability to handle offline messages in various conditions. We noted that \( O^2TR \) does not impose significant processing penalties to clients in its communication system. Furthermore, \( O^2TR \) provides an efficient session resumption, which is about 34% faster than creating a new private session in OTR.

For future work, we plan to develop our messaging system to provide immediate and secure messaging communication system. To this end, we are designing a model to speed up the handshake process of \( O^2TR \). Eventually, our secure and reliable communication system will immediately provide a secure connection between clients and emergency service centers.

Appendix A. HLPSL evaluation code for \( O^2TR \)

To evaluate the security of \( O^2TR \) protocol, we modeled \( O^2TR \) protocol using AVISPA: Fig. A.9 describes a model for the protocol between client A and client B in \( O^2TR \). Fig. A.10 describes a model for the session and environment in \( O^2TR \).

---

10 http://www.avispa-project.org/.  
role role_A(A:agent,B:agent,Ka:public_key,SND,RCV:channel(dy))
played by A
def=
    local State,Mdf:nat,N,F:text
    init State := 0 /
    Mdf := 0
    transition
    0. State := 0 /\ RCV({B.N}.Ka) =\> State' := 2 /\ Mdf' := 100
    2. State := 2 /\ RCV({B.F}.Ka) =\> State' := 0 /\ Mdf' := 200
end role
role role_B(B:agent,A:agent,S:text,Ka:public_key,SND,RCV:channel(dy))
played by B
def=
    local State:nat,
    F,N:text
    init State := 1
    transition
    1. State := 1 /\ RCV(start) =\> State' := 3 /\ SND({B.N}.Ka)
    3. State := 3 /\ RCV(start) =\> State' := 1 /\ SND({B.F}.Ka)
end role

Fig. A.9 – Client A and B in $O^2$TR.
role session (A: agent, B: agent, S: text, Ka: public key)
def=
    local
    SND2, RCV2, SND1, RCV1: channel (dy)
    composition
    role_B (B, A, S, Ka, SND2, RCV2) \ role_A (A, B, Ka, SND1, RCV1)
end role
role environment ()
def=
    const
    alice, bob: agent,
    ka: public key,
    s1: text,
    sec_1: protocol_id
    intruder.knowledge = {alice, bob, ka}
    composition
    session (alice, bob, s1, ka)
end role
goal
secrecy_of_sec_1
end goal
environment ()

Fig. A.10 – Session and environment in O²TR.

REFERENCES


Boneh D, Victor S. A graduate course in applied cryptography. https://crypto.stanford.edu/dabo/cryptobook/draft_0.3.pdf; 2016. Version 0.3.


http://doi.acm.org/10.1145/1102199.1102216.


http://doi.acm.org/10.1145/2342356.2342427.


Mahdi Daghmecheh Firoozjai is a postdoc research fellow in the Department of Computer and Information Security, College of Software & Convergence technology, Sejong University. He received a B.S. degree in telecommunication engineering from the Scientific-Applying Faculty of Post and Telecommunication, a M.S. degree in cryptography from Imam Hossein Comprehensive University, Tehran, Iran, and his Ph.D. degree from the Department of Electrical and Computer Engineering, Sungkyunkwan University, Suwon, Korea in 2000, 2005 and 2018, respectively. In 2006, he joined the Telecommunication Company of Mazandaran (TCM) as a network expert until 2014. His research interest focuses on Blockchain, network security and privacy preserving in the online services.

MinChang Kim is a Ph.D. student in the Department of Computer Science and Engineering, College of Information and Communication Engineering, Sungkyunkwan University. He received his M.S. degree in business administration from Sungkyunkwan University, Seoul, Korea, in 2009. Since 2006, he has been working in the Software Content Research Laboratory of Electronics and Telecommunications Research Institute (ETRI). His research interests are security engineering, mobile security, IoT Security, and artificial neural network.

JaeSeung Song is an associate professor, leading the Software Engineering and Security group (SESelab) in the Computer and Information Security Department at Sejong University. He is holding the position of oneM2M Test Working Group Chair, IEEE ComSoc Standards for IoT Architectures Research Group Chair, and TTA IoT/M2M Convergence Special Project Gror oup Vice Chair. Prior to his current position, he worked for NEC Europe Ltd. between 2012 and 2013 as a leading senior researcher. At that time, he actively participated in IoT related R&D projects such as the Building as a Service (BaaS) FP 7 project and IoT/M2M standardizations (i.e., ETSI TC M2M and oneM2M). From 2002 to 2008, he worked for LG Electronics as a senior researcher leading a 3GPP SA standard team. He received a Ph.D. at Imperial College London in the Department of Computing, United Kingdom in 2011. He hold BS and MS in Computer Science from Sogang University, respectively in 2000 and 2002.

Hyungshick Kim is an assistant professor in the Department of Computer Science and Engineering, College of Information and Communication Engineering, Sungkyunkwan University. He received a B.S. degree from the department of Information Engineering at Sungkyunkwan University, a M.S. degree from the Department of Computer Science at KAIST and a Ph.D. degree from the Computer Laboratory at University of Cambridge in 1999, 2001 and 2012, respectively. After completing his Ph.D., he worked as a post-doctoral fellow in the Department of Electrical and Computer Engineering at the University of British Columbia. He previously worked for Samsung Electronics as a senior engineer from 2004 to 2008. He also served as a member of DLNA and Coral standardization for DRM interoperability in home networks. His current research interest is focused on social computing and usable security.