mpCat - A Secure Multi-Party Chat Protocol
Second Draft

June 17, 2014

Abstract

In this document we present the second draft of a secure multi-party chat protocol meant to address a variety of use-cases. We include its design rationale, choice of security features, adversarial models, schematic and detailed specification of sub-protocols and primitives. The second draft is built on the commentary received from the specifications review team.

I. Introduction

In this document we describe the rationale behind designing mpCat, a secure multi-party chat protocol meant for use in a variety of real-world use cases.
In the following section we skim over relevant publications and their results. In Section III, we describe our approach and choice of security features. In Section IV, we overview the properties that we are aiming for in this protocol. In Section V we give basic mathematical definition needed to model the chat session and security proofs for various security aspects of the protocol. Section VI provides definitions and references to the adversarial models for each property. In Section VII we describe various parts of the protocol and present choices for each sub protocol and primitives for each step of the general protocol. In Section VIII, we present the revised draft of mpCat. Finally, we conclude by describing the remaining steps.

II. History and literature review

Two-party off the record messaging (OTR) has been introduced in [BGB04] as a better alternative to PGP for casual Internet chat. OTR authors argue that using PGP for Internet chat is problematic due to the PGP scheme's lack of forward secrecy and deniable transcript featchoicures. These properties are expected in Internet chat, since it mimics casual day-to-day real-world conversations where future deniability is implicit.

[BGB04] offers OTR as an alternative approach to PGP for simulating casual two party chat on the Internet. While OTR uses symmetric encryption and message authentication to secure confidentiality and message integrity, it uses Diffie-Hellman key exchange as an approach to deniably authenticate the other party in the chat.

There have been various security analysises and some criticisms of OTR since its introduction in 2004. For example, [BM] shows that the unauthenticated exchange of the OTR version identifier can pose a threat to authenticity: the adversary can force clients to downgrade to an older, insecure version of the protocol. They also make note of the Diffie-Hellman key exchange failure in delivering authentication in the presence of an active adversary. Furthermore, they show that the early publication of MAC keys for the purpose of forgeability can easily enable the active adversary to forge messages during the conversation (instead of the intended forgeability after the conversation has ended). Finally, they argue that in an environment where the adversary is controlling the whole network, she can effectively disarm the protocol of its forgeability property.

In [RGK05], researchers criticize OTR's approach in which the authenticity of the renewed ephemeral session keys is provided by the property of confidentiality, and is therefore dependent on the secrecy of the conversation. Hence, breaking the secrecy of the conversation (by the leak of the session key, for example) will lead to false authentication as well. They offer two authenticated deniable key exchange protocols, which also provide forward secrecy, as a replacement for OTR's original key exchange. Furthermore, they argue that forgeability and malleability do not have any mathematical consequence in improving deniablity if the parties have been authenticated by a deniable key exchange scheme. They argue that as these properties pose potential security threats, it is desirable to omit them from the protocol entirely.

In [GUVGC09], the authors offer a generalization of two-party OTR to the multi-party case. However, they do not specify the cryptographic primitives, neither do they give a formal definition of the adversaries or the proof of algorithm's security (reduction). Although a more robust key exchange is proposed, some primary performance analysis

of the implementation of the key agreement protocol has been shown to be impractically slow, especially on mobile devices.

Various attempts have been made to construct an efficient multiparty (known as group) authenticated key exchange protocol. Protocols proposed in [BCP01] and [BCPQ01] have been shown to be not secure against various adversarial models [GBNM11] [Man06]. [BVS05] shows that the protocol introduced in [KLL04] is not secure against replaying the user's message in another chat. The authors offer a slightly modified version of the protocol to remedy this.

Authors of [RGK05], introduce 2 protocols with forward secrecy to replace the vulnerable deniable authentication of OTR. Both [RGK05] and [BS07] argue that SIGMA does not meet the definition of a truely deniable algorithm and the latter shows how it fails the deniability adversarial model introduced in [BS07]. Alternatively, [BS07], using the Schnorr zero-knowledge proof and signature algorithm, introduce a 4-round challenge-based authentication scheme that grants deniability to the two-round authenticated protocol described in [BVS05].

[ACMP10] offers a more efficient protocol than [BS07], in the sense that ephemeral Diffie-Hellman elements are reusable to regenerate keys when some of the participants change. As such, it offers a one-round protocol to generate a key for a subgroup of the originacl conversation.

In designing mpCat's deniable authentication and key agreement protocol, we have followed the main idea of [BS07] of choosing a provably secure authenticated key exchange method and replacing the signature based authentication with a deniable one. We have chosen the protocol introduced in [ACMP10] instead, due to its efficiency superiority. We have chosen the two rounds SKEME-based Triple Diffie-Hellman deniable key authentication instead of Schnorre based signature suggested in [BS07] for it saves us two critical rounds for authentication. We have also modified the protocol to represent the chat condition where participants sequentially join and leave the chat.

Another major difference between mpCat and the suggested original protocol for mpOTR in [GUVGC09] is transcript authentication for every time a participant receives a message. This is an optimistic approach based on the assumption that the XMPP server provides a reliable and orderly message delivery.

III. Design rationale

The main motivation behind the development of mpCat is the lack of provably secure, implementable, end-to-end encrypt multiparty chat protocols that apply to a variety of use-cases. Our approach for the mpCat design was based on the following rationales, listed in order of importance:

- A protocol that is provably secure in a sufficiently strong adversarial model which addresses the most urgent requirement of users in need of security. These are confidentiality, authenticity and forward secrecy.
- Usability according to real world use cases, including asynchronous use cases.
- Providing some degree of deniability when it does not hurt usability or our fundamental security goals.
- Addressing security flaws in the OTR protocol.

To achieve these goals, we focused our studies on the OTR protocol and various subsequent protocols evolved from OTR such as [Sys], as well as papers offering security analysis of the original OTR protocol. We designate the protocol suggested in [GUVGC09] as our starting point and apply various modifications to reach a desirable protocol which satisfies our goals.

A significant portion of this research suggests a better performing, more secure alternative to the key exchange protocol suggested in [GUVGC09] which is considered by various researchers to be one of the most troubling and inefficient aspects of the proposal.

Additionally, based on the conclusions of [BM] and [RGK05], we are taking the following points into consideration:

- Using a more secure deniable key exchange algorithm instead of naive Diffie-Hellman and a more practical algorithm rather than the peer-to-peer signature key exchange suggested in [GUVGC09].
- Omitting forgeability and malleability from the protocol as recommended by [RGK05] and refraining from broadcasting the expired ephemeral authentication keys. We propose the possibility of using block-based, rather than stream-based, encryption for the symmetric encryption primitives.
- Offering provably secure models for every aspect of the algorithm which we have singled out as critical to every day use cases.

III.1 On Deniability

Deniability can be partially achieved by using a deniable key exchange algorithm. At this stage of the design we do not define an adversarial model for deniability. However, as our deniable authentication takes the same approach of [BS07], it seems possible to prove deniability based on the model introduced in [BS07].

IV. Security Properties

MpCat aims to secure the following properties in a multi-party chat session:

- **Participant deniable authenticity** based on their long term persistent identity: While a participant in a chat can be sure of another participant's authenticity, they cannot prove their confidence to anybody else.
- **Message origin authenticity** against both outsider intrusion and the impersonation of existing participants by other malicious participants in the session.
- **Transcript integrity**, where all participants are confident that they have been participating in the same conversation and have seen the same messages.
- **Confidentiality** of the conversation so its content is not accessible by an outsider.
- **Forward secrecy** of the conversation, so its content remains inaccessible in the event of the long term private key of a participant (which represents their long term identity) being compromised after session key establishment.

According to each requirement, we need to examine our protocol against the adversaries of

1. Deniable authenticated key exchange.
2. Message origin authenticity.
3. Consensus.
4. Confidentiality.
5. Forward secrecy.

V. Chat Session Model

In modelling our chat session for various adversarial model and protocol specifications, we are following the notation of [GBNM11].

Definition 1 Multi-party chat session: *Let $\mathcal{U} = \{U_1, ..., U_n\}$ be the set of possible participants. A multi-party chat session is an ordered pair $S = (\mathcal{S}, sid)$ in which $\mathcal{S} \subset \mathcal{U}$ and sid the unique session id. Furthermore, it is assumed that party U is presented and identified verifiably by a long-term persistence key pair (PK_U, SK_U).*

Definition 2 *An* **authenticated group key exchange (AGKE)** *is Algorithm π that is running on each party's computer. For the sake of defining AGKE we need the following definitions:*

- **Session id as seen by** *U: Session id sid will be derived during the run of the protocol. The session id is computed by π_U (the instance of the protocol run by U) and is indicated by sid_U.*
- **Participant list**: *$plist_U^S$ is the list of participants which U believes participating in the chat session S.*
- **Session key as seen by** *U: sk_U^S is the session key as computed by π_U.*
- **Accepted state**: *A party enters the accepted state if it computes sk_U^S.*
- **Partnered instances**: *Two instances π_U and $\pi_{U'}$ are considered partnered if and only if both instances have accepted $sid_U = sid_{U'}$ and $plist_U = plist_{U'}$.*

We say an AKGE algorithm is **valid**, *if in the case of an adversary honestly forwarding all messages, all participants ultimately are partnered and all compute equal sk_U's.*

VI. Adversarial models

Adversarial models are explained as a game, in which the advantage of the adversary winning the game should be translatable to the advantage of breaking the cryptographic primitives.

Accordingly to our requirement, we need to examine our algorithm against the adversaries of

1. Deniable Authenticated key exchange (including a forward secrecy adversary)
2. Message origin authenticity
3. Confidentiality
4. Consensus

In following sections, we are defining adversaries which represent the threats concerning us in each of above properties.

VI.1 Deniable Authenticated Key Exchange Adversary

We use the adversarial model for authenticated key exchange from [GBNM11]. which can promise resistance against internal/external key compromise impersonation (KCI). Similar models have been introduced in [BVS05] [ACMP10]. The deniablity adversary is presented in [BS07].

VI.2 Message Origin Authentication Adversary

The message origin adversary is a typical adversary against a signature scheme.

VI.3 Message Confidentiality Adversary

The goal of adversary $\mathcal{A}_c$ is to read at least part of the transcript during the session. $\mathcal{A}_c$ will be in effect only after the session key sk_S is established and all instances agree on *plist*, and assuming no party is corrupted by $\mathcal{A}_c$.

Initially, a secret random bit b is generated. The adversary sends two sequences to instance π_U: $(M_{0,i}, M_{1,i})$. In reply, π_U broadcasts $C_{b,i} = E_{sk_{id}}(M_{b,i})$ where $E_K(M)$ is mpCat's encryption scheme encrypting message M with key K. $\mathcal{A}_c$ wins if its guess of b' has a non-negligible $prob(b' == b)$.

As we use AES in counter mode, the proof of confidentiality of the message encryption goes along the same lines of the standard proof of the same scheme.

VI.4 Forward Secrecy Adversary

We do not define an independent forward secrecy adversary. Forward secrecy can be derived by resistance against the confidentiality adversary as well incorporating a forward secure key exchange as described in [GBN10].

VI.5 Consensus Adversary

Definition 3 *Consensus Adversary $\mathcal{A}_{con}$ is given the ability to corrupt, reveal the key, and basically impersonate participants at all stage of the session. We say $\mathcal{A}_{con}$ wins if the protocol is secure in our key exchange security assumption and there are at least two uncorrupted parties U, U' for which U and U''s ending sessions equal $T_U \neq T'_U$ while actually believing $T_U = T'_U$.*

VII. Protocol High Level Design

To achieve security properities mentioned in section IV, we break the protocol in following sub-protocols:

1. **Deniable authenticated signature key and session key exchange**, where participants deniably authenticate each other and agree on a common session key, meanwhile exchanging ephemeral signing keys.
2. **Communication**, where parties send authenticated confidential messages.

3. **Consensus verification**, where parties verify that all have received and seen identical transcript during the chat session.

our choice of sub-protocols for mpCat has been to use the same sub-protocols and primitives suggested in [BGB04] and [GUVGC09], unless there has been a practical or security-related reason to deviate from those recommendations.

We have completely replaced the session and signature key establishment protocol as the original choice of [GUVGC09] for this phase proved impractical in previous implementations. We have moved to elliptic curve cryptography to save on key and signature length in asymmetric primitives.

Following the conclusion of [RGK05] we have dropped forgeability (mandatory publication of ephemeral signature/MAC keys) and malleability. As it is argued in [RGK05], the deniability of the protocol is based on deniable key exchange, and while they increase the complexity of the algorithm, the above properties do not mathematically contribute to deniability. Taking this step also significantly improves the efficiency of the protocol which is a main focus for mpCat.

We ensure consensus whenever the underlying transport guarantees the reliable delivery of the messages in the same order for all participants.

In the following section we briefly describe our choice of the sub-protocol for each of the required tasks for a multi-party chat session.

VII.1 Design of Deniable Authenticated Signature Key Exchange

We have chosen our deniable signature key exchange protocol following the conclusions in [Gun13a] - by identifying a secure key exchange protocol that satisfies our needs. We then apply the triple Diffie-Hellman authenticated exchange to grant it properties of deniability. Subsequently, one can apply the same approach presented in [Gun13a] to communicate ephemeral signature keys during the key establishment process. However, for efficiency, we use the same ephemeral Diffie-Helman secret and public values to produce ephemeral signatures.

For the choice of the base authenticated key exchange protocol, we suggest a variant based on [ACMP10]. The rationale of the choice can be itemized as follows:

- The base of the design of the protocol in [ACMP10] is the same as in [BVS05]. However, a simpler protocol is presented in [ACMP10].
- [ACMP10] offers a peer-to-peer key exchange with no extra rounds, if needed.
- [BVS05] and [ACMP10] are superior to the widely studied [BCPQ01] and its dynamic variation [BCP01] both in security (against malicious insiders) and performance (O(1) rounds).
- [BVS05] has been suggested by [Gun13a] for the reason described in [Gun13b]. We believe that the new deniable authentication approach, as it is similar to the SKEME protocol, should satisfy the properties of deniability which [BVS05] considered crucial.
- Security analysis of [GBNM11] and [BCGNP08] both find [BVS05] is provably secure against all attacks (including the insider attacks) they consider.
- It is a two-round protocol and hence offers competitive efficiency considering the security property that it provides.
- The security proof of [BVS05]and [ACMP10] in the random oracle model is acceptable considering the importance of usability and efficiency as our goals.
- [BVS05] has existed for years and its various security aspects have been investigated by several researchers including [GBNM11] and [BCGNP08] which gives [BVS05] an advantage over newer algorithms.
- [ACMP10] only needs one round key re-agreement in case of a participant leaving the chat, while [BVS05] enforces re-computation of Diffie-Hellman ephemeral keys and hence needs a minimum of two rounds plus overhead of re-authenticating the new ephemeral keys. This can significantly improve the efficiency of casual chat sessions where participants frequently enter and exit the chat.
- Triple Diffie-Hellman authenticated key exchange only needs two rounds of communication and can be done alongside the key agreement steps, while the Schnorr based algorithm suggested in [BVS05] needs four rounds.

VII.2 Message Authentication

As message authentication needs to be resistant to malicious insiders, following the original [GUVGC09], mpCat signs each message using a public key signature scheme. ED25519 has been chosen as the signature primitive due to its efficiency and more secure implementability over other elliptic-curve digital signature algorithms. The messages are signed with the ephemeral key of the sender. The authenticity of the origin can be verified by the public ephemeral key of the party distributed during the key exchange period.

VII.3 Message Encryption

We are using AES-256 in counter mode with the group key for message encryption, as suggested by the original OTR protocol.

VII.4 Transcript Authentication

Because each message sent by each participant is signed by the ephemeral private key generated for the specific session, it is not possible for the internal or external adversary to forge a message on behalf of an uncorrupted participant.

However, if the adversary is controlling the network structure, denial or delay of service is always possible. Hence, the consistency of the transcript (meaning that all participants see the same transcript in the same order) relies on the means of transport guaranteeing reliable delivery, with a single order, to every participant.

The protocol offered in this document examines the transcript for such consistency. In the case that the underlying transport fails to provide this level of consistency, obviously the consistency test will fail. However, the implementation can safely ignore failure that is due to the absence of a reliable transport.

mpCat performs transcript authentication whenever a message is received. This is to guarantee consensus and protect the protocol against the consensus adversary. The procedure is similar to the procedure described in [GUVGC09], except we also require message order to be preserved for the following reasons:

1. XMPP, as the main protocol in focus for this design, delivers messages to all clients in the same order.
2. mpCat protocol detects if the adversary has mingled with the order of the messages rather than only dropping undesirable messages
3. It is simpler to authenticate an ordered transcript compare to an unordered transcript.

VII.5 Asynchronous communication and Forward Secrecy1

The protocol is primarily targeted to synchronous cases, however, with some modification it can be used for asynchronous cases.

Provided that the participants are not concerned with authenticating the list of participants (it is OK if Eve impersonate Bob as long as she is unable to read Bob's emails) participants can communicate using their pairwise exchanged ephemeral Diffie-Hellman keys, until all participants finish the second round of authentication.

As soon as a deniable handshake has been established among a set of participants, any subset of them can communicate and authenticate their messages using the "session key" and their ephemeral signature key.

The protocol does not enforce explicitly a time limit on renewing the session key shares and can be used for an asynchronous high latency transport after the key establishment state.

The downside of using a session key for a long time is that a compromised session key will reveal all past communication during that session. This does not pose an imminent threat when the life span of a chat is short. However, in the context of asynchronous high latency transport, it is of a more serious concern.

The protocol requires the participants to preemptively update their ephemeral signature/shares and propagate them as part of the messages they are already sending. Subsequently, they also update their key share with their neighbours, as soon as the neighbours also propagate their new ephemeral signature keys.

As the assumption of having a continuous heartbeat might not be realistic in various asynchronous cases, the implementation can assume specific deadlines for dropping users who did not communicate their new keys or shares.

Because communicating multiple keys and shares might be more time consuming as compared to a dual party chat, the protocol can be extended to use the hybrid key exchange and hash ratchet model described in [Per] to provide some level of forward secrecy before the session key gets updated.

VIII. mpCat Protocol: Step by Step

In this section, we present the mpCat protocol in algorithmic format. All user Ids should be considered modulo number of participants in the room.

At a glance, deniable authentication is derived from the triple Diffie-Hellman algorithm presented in [Sys]. Joining the room is a variation of the two-round mBD+P protocol presented in [ACMP10] where the authentication step has been made deniable. Leaving the room is the one-round mBD+S from [ACMP10].

VIII.1 Schematic view of the key exchange

For simplicity, group operation is written multiplicatively, although it is actually elliptic curve points operation, traditionally represented by addition.

Rnd	Description	Pseudo-code
1	Generate ephemeral DH private key	$x_i \leftarrow [0, order(g)]$
	Generate DH key for BD, Triple DH and Signature	$y_i \leftarrow g^{x_i}$
	Broadcast User identity and the DH key	(U_i, y_i)
2	Compute Session Id	$sid \leftarrow (U_1 \vert y_1, \ldots, U_n \vert y_n)$
	Generate Triple Deffie-Hellman P2P keys	$k_{i,j} \leftarrow \boxed{H(g^{lp_i} LP_j y_j^{x_i})}$
	Generate key confirmations	$\boxed{kc_i \leftarrow (H(k_{i,1}, U_1), \ldots, H(k_{i,n}, U_n))}$
	Generate secret shares	$z'_{i,l} \leftarrow H(k_{i,i-1}, sid_i), z'_{i,r} \leftarrow H(k_{i,i+1}, sid_i)$
	Generate public shares	$z_i \leftarrow z'_{i,l} \oplus z'_{i,r}$
	Sign identity, shares	$\sigma_i \leftarrow Sign_{x_i}(U_i, z_i, sid)$
	Broadcast key shares and confirmation	$(U_i, z_i, sig_i, \boxed{kc_i})$
-	check validity of key confirmation	$\boxed{kc_i[j] == kc_j[i] \text{ for } j \in \{1, \ldots, n\}}$
	check signatures	$verify_{y_i}(\sigma_j)$ for $j \in \{1, \ldots, n\}$
	Recover secret shares	$z'_{j,r} \leftarrow z'_{j-1,r} \oplus z_j$
	Generate session key	$k_i \leftarrow H(z'_{1,r} \ldots z'_{n,r}, sid_i)$

VIII.2 Chat setup

In almost any practical case, participants join the chat sequentially. It is assumed that multiple participants cannot join simultaneously. For the sake of efficiency one can tweak the implementation to have a threshold to wait and start a chat with more participants. However, this makes the implementation significantly more complicated without an evident efficiency benefit.

Therefore, our assumption is that a secure chat is always set up when a participant starts the chat room. Additional participants would be added sequentially using Algorithm VIII.3, as they enter the chat. Algorithm 1 describes the chat room setup protocol.

Algorithm 1 Chatroom setup

procedure CHAT INITIATOR INIT($newRoomName$, $participantNick$)
  Global $myId := 1$
  Global $Nick_{myId} := participantNick$
  Global $roomName := newRoomName$
  Global $x_{myId}, y_{myId} :=$ GENERATE INITIAL PARAMTERS($myId$)
  Global $signatureKey_{myId} := (x_{myId}, y_{myId})$
  $participantList := [Nick_{myId}]$
  $ephemeralPublicPointList := [y_{myId}, y_{other}]$
end procedure
procedure VERIFY VERIFIER GENERATE INIT KEY($schnorrRandomPoint_{other}$, Hv_{other}, v_{other}, y_{other}, $Nick_{other}$)
  VERIFY VERIFIERS
  Global $sessionKey := SHA-512(x_{myId}y_{other}, sessionId)$
  $toBeSigned := SHA-512(SHA-512(sessionId||SHA-512(y_1, v_1)||SHA-512(y_2, v_2)))$
  SIGN SESSION AND SEND($toBeSigned$)
end procedure

VIII.3 Joining

Joining a chat involves two different procedures: the Join procedure, described in Algorithm 2, which runs on the new participant's instance, and an Accept New Participant Procedure, described in Algorithm 3, which runs on the clients of participants that are already in the chat.

When a new participant U_{n+1} joins the chat, current participants can still use their established authenticated ephemeral public key (to derive the $sessionKey_{new}$ and as their signature verification key). Confidentiality of $sessionKey_{old}$ is guarded against the new participant by Diffie-Hellman key shares hashed alongside the session id (which is dependent on the list of participants). The new participant cannot combine the old and new shares to recover $sessionKey_{old}$. The fact that old participants do not need to compute new ephemeral keys (and re-verify their ephemeral identities) decreases the computational complexity of the protocol.

Algorithm 2 Join

procedure JOIN($newRoomName$, $Nickname_{myId}$, $participantId$)
  Global $myId := participantId$
  Global $roomName := newRoomName$
  $x_{myId}, y_{myId} :=$GENERATE INITIAL PARAMTERS(Participant ID $myId$)
  Global $signatureKey_{myId} := (x_{myId}, y_{myId})$
  BROADCAST(":3mpCat:3Join:3", $myId$, $Nickname_{myId}$, y_{myId})
  Global $participantList, ephemeralPublicPointList :=$ RECEIVE
  Global $sessionId :=$ COMPUTE SESSION ID($roomName$, $participantList$, $ephemeralPublicPointList$)
  SIGN AND SEND KEY CONFIRMATION AND SHARES
  WAIT ON RECEIVE(":3mpCat:3KeyConfirmationShare:3")
  Global $keyShareList, keyConfirmationList, signatureList :=$ RECEIVE
  VERIFY KEY CONFIRMATIONS AND SIGNATURES($keyConfirmationList$, $signatureList$)
  UPDATE SESSION KEY
end procedure
procedure RECEIVE SESSION DIGEST($currentSessionHistoryDigest$)
  Global $sessionDigest := currentSessionHistoryDigest$
end procedure

VIII.4 Leave

Leaving a chatroom involves only one procedure for those who are staying in the chatroom (Procedure Farewell) which is described in Algortihm 4. The remaining participants only need a notice from the server that the user is leaving to

Algorithm 3 Protocol for other participants already in the chat to accept the newcomer

procedure ACCEPT(newParticipant)
  BROADCAST(":3mpCat:3Join:3", $myId$, $Nickname_{myId}$, y_{myId})
  WAIT ON RECEIVE(":3mpCat:3Join:3")
  Global $nick_{NewParticipant}$, $ephemeralPublicPoint_{NewParticipant}$:= RECEIVE
  UPDATE LISTS($nick_{NewParticipant}$, $ephemeralPublicPoint_{NewParticipant}$)
  Global $sessionId$:= COMPUTE SESSION ID($roomName$, $participantList$, $ephemeralPublicPointList$)
  SIGN AND SEND KEY CONFIRMATION AND SHARES
  WAIT ON RECEIVE(":3mpCat:3KeyConfirmationShare:3")
  Global $keyShareList$, $keyConfirmationList$, $signatureList$:= RECEIVE
  VERIFY KEY CONFIRMATIONS AND SIGNATURES($keyConfirmationList$, $signatureList$)
  UPDATE SESSION KEY
  SEND($sessionDigest$)
end procedure

re-run the one round key update algorithm. Also, failure to receive a heartbeat from a user will result in executing 4 excluding users which did not update their key.

Algorithm 4 Farewell

procedure SHRINK ON LEAVE($leaverId$)
  Remove $leaverId$ from $participantIdList$
  Global $sessionId$:= COMPUTE SESSION ID
  if $|participantList| > 1$ **then**
    SIGN AND SEND KEY SHARES
    WAIT ON RECEIVE(":3mpCat:3KeyShare:3")
    $keyShareList$:= Receive
    UPDATE SESSION KEY($keyShareList$)
  end if
end procedure
procedure SIGN AND SEND KEY SHARES
  Global $z_{myId-1,myId} := SHA-512(k_{myId,myId-1}, sessionId)$
  Global $z_{myId,myId+1} := SHA-512(k_{myId,myId+1}, sessionId)$
  $keyShare_{myId} := z_{myId-1,myId} \oplus z_{myId,myId+1}$
  $originAuthSignature$:= ED25519SIGN($SignatureKey$, $sessionId \mid\mid z_{myId}$)
  BROADCAST(":3mpCat:3KeyShare:3", $myId$, $keyShare_{myId}$, $originAuthSignature$) ▷ we can send this encrypted but leaving person can read it, hence theoretically it is the same as sending it unencrypted.
end procedure

VIII.5 Secure Send and Receive

After the session key is established, participants will use Algorithms 5 and 6 to communicate securely.

On send, the protocol checks the status of the new ephemeral Diffie-Hellman and key share using messages it receives from participants. It (re)sends any missing pieces. It also informs other participants which part of the key share is received by that user. The meta data flag indicates if the message being sent only contains meta data (e.g. heatbeat) or actual user communication.

On receive, the protocol updates who has which pieces of the key shares. The protocol also generates a new group key; if the new key shares have been received from all participants or those who have not updated their key shares time out on their heartbeat interval.

Algorithm 5 Send

procedure SEND(Message)MetaMessage, message
  keyShareMessage = NEWKEYSHAREMESSAGE(MetaMessage)
  cryptMessage := AES CTR ENCRYPT(*sessionKey*,*message*|*keyShareMessage*)
  originAuthSignature := ED25519SIGN(*SignatureKey*, *sessionId* || *cryptMetatMessage*)
  sessionDigest := COMPUTE SESSION DIGEST(*lastMessage*)
  BROADCAST(":3mpCat:3", *sessionId*, *cryptMessage*, *sessionDigest*, *originAuthSignature*,":3")
end procedure

Algorithm 6 Recieve

procedure RECEIVE(*sender*, *encryptedMessage*, *originAuthSignature*, *sessionDigest*)
  v := ED25519VERIFYSIGNATURE(*ephemeralPublicKeyList*[*Sender*], *sessionId*||*encryptedMessage*, *originAuthSignature*)
  ASSERT(v) or **return** Reject
  message, *keyShareMessage* := AES CTR DECRYPT(*sessionKey*, *encryptedMessage*)
  isMetaMessage =UPDATENEWKEYSTATUS(*keyShareMessage*)
  VERIFY DIGESTS(*sessionDiges*)
  return *isMetaMessage*, *message*
end procedure

VIII.6 Common functions

Algorithm 7 Common functions used by other procedures in different stages

procedure GENERATE INITIAL PARAMTERS(*myId*)
  signaturePrivateKey := RANDOMBITS(256)
  x_{myId} := ED25519 SCALAR(*signaturePrivateKey*)) ▷ This is both Diffie-Hellman secret and ephemeral signature private key
  $y_{myId} := x_{myId}P$ **return** x, y
end procedure
procedure VERIFY KEY CONFIRMATION AND SIGNATURES(*signatureList*, *keyConfirmationList*)
  for all *participant* ∈ *participantList* **do**
    if $keyConfirmationList[participant][myId] \neq SHA-512(k_{myId,participant}, U_{myId})$ **then**
      HALT
    else
      if ED25519VERIFYSIGNATURE(*ephemeralPublicKeyList*[*particicpant*], *sessionId*||*keyShares*[*myId*], *originAuthSignature*) = Fail **then**
        HALT
      end if
    end if
  end for
end procedure
procedure COMPUTE SESSION ID(*participantList*, *ephemeralPublicPointList*)
  return $SHA-512(roomeName, zip(participantList, ephemeralPublicPointList))$ ▷ $zip([a,b],[c,d]) := [(a,c),(b,d)]$
end procedure
procedure VERIFY SIGNATURES(*longPublicList*,*schnorrRandomPointList*,)
end procedure

Algorithm 8 ...Common functions continued

procedure SIGN AND SEND KEY CONFIRMATION AND SHARE(*schnorrRandomPointList*)
  for all *participant* ∈ *participantList* **do**:
    $k_{myId,participant} := H(g^{lp_{myId}} LP_{participant} y_{participant}^{x_{myId}})$ ▷ Triple DH
    $kc_{myId} := kc_{myId} | H(k_{myId,participant}, U_{participant})$
  end for
  Global $z_{myId-1,myId} := SHA-512(k_{myId,myId-1}, sessionId)$
  Global $z_{myId,myId+1} := SHA-512(k_{myId,myId+1}, sessionId)$
  $keyShare_{myId} := z_{myId-1,myId} \oplus z_{myId,myId+1}$
  $originAuthSignature :=$ ED25519SIGN(*SignatureKey*, $sessionId \,||\, z_{myId}$)
  BROADCAST(":3mpCat:3KeyConfirmationAndShare:3", *myId*, $keyShare_{myId}$, $originAuthSignature$, kc_{myId})
end procedure
procedure UPDATE SESSION KEY(*keyShareList*)
  $i := myId$
  for all do$j \in [i, ..., i+n-1]$
    $z_{j,j+1} := z_{j-1,j} \oplus keyShareListe[j+1]$
  end for ▷ recovered $z_{i-1,i}$ should be equal to its original value
  Global $sessionKey := SHA-512(z_{j,j+1} | j \in [1...n])$
end procedure
procedure SIGN PARAMS UPDATE SESSION KEY(*toBeSigned*, *signatureList*, *keyShareList*)
  UPDATE SESSION KEY
  $toBeSigned := SHA-512(sessionId, ||SHA-512(verifierList, ephemeralPublicPointList, keyShareList)))$
  $signature_{myId} :=$SIGN SESSION AND SEND(*toBeSigned*)
  BROADCAST(":3mpCat:3SignedSessionParameters:3",$signature_{myId}$)
end procedure
procedure COMPUTESESSIONDIGEST(*lastMessage*)
  for all *message* in Messages Received from *lastDigestedMessage*+1 till *lastMessage* **do**
    sesionDigest := SHA-512(*sessionDigest*, *message*),
    LRU CACHE STORE DIGEST(*sessionDigest*, *message*)
  end for
  return *sessionDigest*,*lastMessageId*
end procedure

IX. CONCLUSION

In this document, we presented an early first draft of mpCat. We hope for this draft to evolve into a usable encrypted multi-party chat protocol. We have referenced the adversarial models which the protocol is supposed to resist, with proof of resistance to the adversarial model presented in the original papers. We plan to publish comprehensive proof for this algorithm against the presented adversaries shortly.

REFERENCES

[ACMP10] Michel Abdalla, CÃľline Chevalier, Mark Manulis, and David Pointcheval. Flexible group key exchange with on-demand computation of subgroup keys. In Dan Bernstein and Tanja Lange, editors, *Third African International Conference on Cryptology (AfricaCrypt '10)*, volume 6055 of *LNCS*, page 351âĂŞ368, Stellenbosch, South Africa, 2010. Springer.

[BCGNP08] Colin Boyd, Yvonne Cliff, Juan Gonzalez Nieto, and KennethG. Paterson. Efficient one-round key exchange in the standard model. In Yi Mu, Willy Susilo, and Jennifer Seberry, editors, *Information Security and Privacy*, volume 5107 of *Lecture Notes in Computer Science*, pages 69–83. Springer Berlin Heidelberg, 2008.

[BCP01] Emmanuel Bresson, Olivier Chevassut, and David Pointcheval. Provably authenticated group diffie-hellman key exchange - the dynamic case. In Colin Boyd, editor, *Advances in Cryptology - Proceedings of ASIACRYPT '01*, volume 2248 of *LNCS*, page 290âĂŞ309, Gold Coast, Australia, 2001. Springer.

[BCPQ01] Emmanuel Bresson, Olivier Chevassut, David Pointcheval, and Jean-Jacques Quisquater. Provably authenticated group diffie-hellman key exchange. In Mike Reiter, editor, *Proceedings of the 8th ACM Conference on Computer and Communications Security (CCS '01)*, page 255âĂŞ264, Philadelphia, Pennsylvania, 2001. ACM Press.

[BGB04] Nikita Borisov, Ian Goldberg, and Eric Brewer. Off-the-record communication, or, why not to use PGP. In *Proceedings of the 2004 ACM Workshop on Privacy in the Electronic Society*, WPES '04, page 77âĂŞ84, New York, NY, USA, 2004. ACM.

[BM] Joseph Bonneau and Andrew Morrison. Finite-state security analysis of OTR version 2.

[BS07] Jens-Matthias Bohli and Rainer Steinwandt. Deniable group key agreement. In Phong Q. Nguyen, editor, *VIETCRYPT*, volume 4341 of *Lecture Notes in Computer Science*, pages 298–311. Springer, 2007.

[BVS05] Jens-Matthias Bohli, Maria Isabel Gonzalez Vasco, and Rainer Steinwandt. Secure group key establishment revisited. *IACR Cryptology ePrint Archive*, 2005:395, 2005.

[GBN10] M. Choudary Gorantla, Colin Boyd, and Juan Manuel GonzÃąlez Nieto. *One Round Group Key Exchange with Forward Security in the Standard Model*. 2010. Published: Cryptology ePrint Archive, Report 2010/083 http://eprint.iacr.org/.

[GBNM11] M. Choudary Gorantla, Colin Boyd, Juan Manuel GonzÃąlez Nieto, and Mark Manulis. Modeling key compromise impersonation attacks on group key exchange protocols. *ACM Trans. Inf. Syst. Secur.*, 14(4):28, 2011.

[Gun13a] Matthew Van Gundy. Improved deniable signature key exchange for mpOTR, April 2013.

[Gun13b] Matthew Van Gundy. [OTR-dev] improved deniable signature key exchange for mpOTR, March 2013.

[GUVGC09] Ian Goldberg, Berkant Ustao\uglu, Matthew D. Van Gundy, and Hao Chen. Multi-party off-the-record messaging. In *Proceedings of the 16th ACM Conference on Computer and Communications Security*, CCS '09, page 358âĂŞ368, New York, NY, USA, 2009. ACM.

[KLL04] Hyun-Jeong Kim, Su-Mi Lee, and Dong Hoon Lee. Constant-round authenticated group key exchange for dynamic groups. In *ASIACRYPT*, pages 245–259, 2004.

[Man06] Mark Manulis. Security-focused survey on group key exchange protocols. *IACR Cryptology ePrint Archive*, 2006:395, 2006.

[Per] Trevor Perrin. Axolotl ratchet.

[RGK05] Mario Di Raimondo, Rosario Gennaro, and Hugo Krawczyk. Secure off-the-record messaging. In *WPES*, pages 81–89, Alexandria, VA, USA, November 2005.

[Sys] Whisper Systems. TextSecure ProtocolV2.