Deuxdrop Protocol Basics

Crypto Primitives

Boxing: Public-key authenticated encryption primitive. The same shared secret is derived from both possible pairings of one user's public key and another user's private key. Combined with a nonce, the message payload and an authenticator are encrypted. Boxed messages are inherently repudiable because the ability to decrypt and verify a message also endows the ability to forge a message. Alice knows a message from Bob must be from Bob because she did not create it herself, but cannot prove that to Trent because she could have just as easily created the message herself.

Secret Boxing: Uses a shared key with nonces to authenticate and encrypt.

Signing: Public-key signatures without nonces or complicated padding.

Crypto Keys Used : Users / Clients

Envelope Box Key: Used to encrypt envelope data sent to the user. The user provides this key to their (mailstore) server.

Body Box Key: Used to encrypt body data sent to the user. The user could optionally provide this key to their (mailstore) server if they wanted it to be able to index the contents of their messages.

Announce Signing Key: Signs messages sent to conversations or any case where individually boxed messages are not viable. Kept just to the clients.

Tell Key: Authorship key for boxed messages (to servers, other users, etc.) Kept just to the clients.

There is also a key authorization hierarchy that is not entirely important to the protocol implementations and therefore elided here for now.

Noteworthy side-effects of the boxing primitives on envelope and body keys: Possession of these keys does not allow the server to masquerade as the user to other users. However, the possessor is inherently able to forge anything it can decrypt given the use of the box primitive. This is why authorship is done using the "Tell" key.

Crypto Keys Used : Servers

Server Box Key: Used for receiving messages from users and sending messages to users. There is no need for multiple keys because no entity other than the server acts on the behalf of the server.

Identities

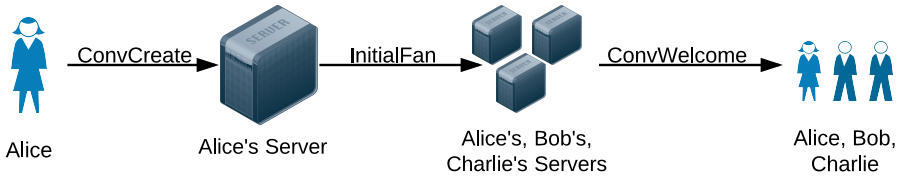
People are identified by a public signing key, providing a unique identifier (their **root key**) and a means of generating authorization chain to keys.

Self-Identity: This is a signed (JSON) object that contains the user's root key, an authorization chain from that root key to their current keyring, details about themselves in the form of a Portable Contacts (PoCo) object, and the identity information for the server they use.

Other Person Identity: A (JSON) object signed by Alice that contains Bob's (signed) self-identity plus a Portable Contacts (PoCo) that provides Alice's name/nickname for Bob (ex: "Bibbity Bob").

Conversation Creation

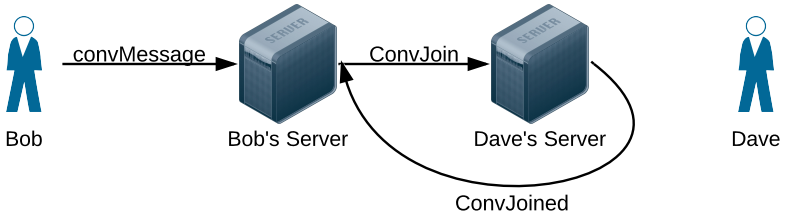
A conversation is created on the creator's server. The server relays the contents to all invited participants (included the conversation creator.)



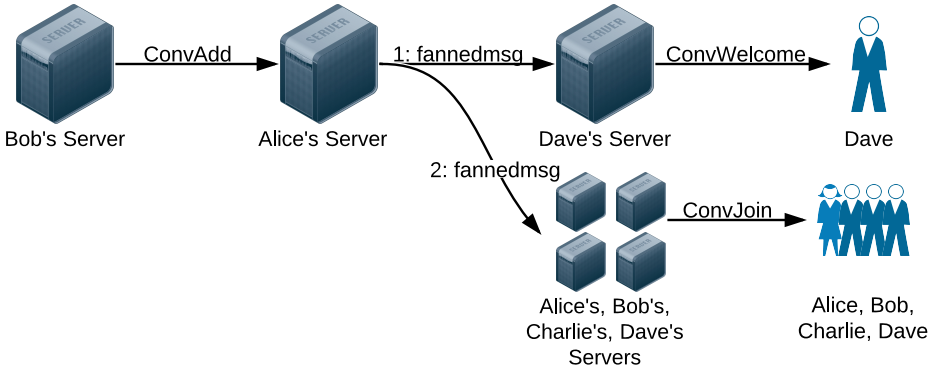
Adding a new Participant

Any participant in the conversation can invite someone new to join the conversation.

1) Because the invitee may not have a contact relationship with the conversation creator, the invitation process first requires the inviter (who must have a contact relationship with the invitee) to tell the invitee's server to authorize messages from the conversation.

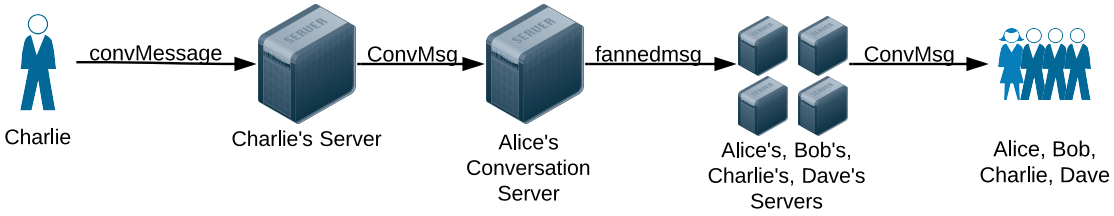


2) The inviter's server completes the addition process once notified by the invitee's server that the conversation join has been authorized. The conversation server provides the new participant with the conversation backlog and notifies the existing participants of the new participant.



Sending Messages (Human and Metadata)

All participants can send messages to the conversation. Messages can be human-readable ("yo!") or metadata (read markers).



Conversation Creation

data details: 1 of 2

Alice
(Conversation Creator)

ConvCreate

Alice's Server

Sends...

ConvCreate => *Alice's Server*

- Invite Proof #A => *Alice's Server*
- Invite Env #A => *Alice*
  - *Conversation Envelope Key*
  - Invite Body #A
    - *Conversation Body Key*
- Join Env #A
  - Join Body #A
    - Attestation #A
      - *key A is "Alice"*
- Invite Proof #B => *Bob's Server*
- Invite Env #B => *Bob*
  - *Conversation Envelope Key*
  - Invite Body #B
    - *Conversation Body Key*
- Join Env #B
  - Join Body #B
    - Attestation #B
      - *key B is "Bob"*
- Invite Proof #C => *C's Server*
- Invite Env #C => *Charlie*
  - *Conversation Envelope Key*
  - Invite Body #C
    - *Conversation Body Key*
- Join Env #C
  - Join Body #C
    - Attestation #C
      - *key C is "Charlie"*
- Message Env
  - Message Body
    - Message Text
      - *"Hello, world!"*

Sees... Parenthesized blocks in grey are opaque blobs to the server.

ConvCreate

- (Invite Proof #A)
- (Invite Env #A)
- (Join Env #A)
- (Invite Proof #B)
- (Invite Env #B)
- (Join Env #B)
- (Invite Proof #C)
- (Invite Env #C)
- (Join Env #C)
- (Message Env)

Creates...

InitialFan #A => *Alice's Server*

- (Invite Proof #A)
- ConvWelcome #A
  - (Invite Env #A)
  - Backlog:
  - (Join Env #A)
  - (Join Env #B)
  - (Join Env #C)
  - (Message Env)

InitialFan #B => *Bob's Server*

- (Invite Proof #B)
- ConvWelcome #B
  - (Invite Env #B)
  - Backlog:
  - (Join Env #A)
  - (Join Env #B)
  - (Join Env #C)
  - (Message Env)

InitialFan #C follows the pattern...

Crypto Legend

Boxed (Public Key Crypto)

- Server -> Server
- Tell <-> Server
- Tell/Server -> Envelope
- Tell -> Body

Secret-Boxed (Shared Secret Crypto)

- Conversation Envelope
- Conversation Body

Signed (Public Key Signature)

- Signed

Conversation Creation

data details: 2 of 2

Sends...

InitialFan #B

(Invite Proof #B)

ConvWelcome #B

(Invite Env #B)

Backlog:

(Join Env #A)

(Join Env #B)

(Join Env #C)

(Message Env)

Sees...

InitialFan #B

Invite Proof #B

ConvWelcome #B

Invite Env #B

Conversation Envelope Key

(Invite Body #B)

Backlog:

Join Env #A

(Join Body #A)

Join Env #B

(Join Body #B)

Join Env #C

(Join Body #C)

Message Env

(Message Body)

Sends...

Invite Env #B

Conversation Envelope Key

(Invite Body #B)

Join Env #A

(Join Body #A)

Join Env #B

(Join Body #B)

Join Env #C

(Join Body #C)

Message Env

(Message Body)

Sees...

Invite Env #B

Conversation Envelope Key

Invite Body #B

Conversation Body Key

Join Env #A

Join Body #A

Attestation #A

key A is "Alice"

Join Env #B

Join Body #B

Attestation #B

key B is "Bob"

Join Env #C

Join Body #C

Attestation #C

key C is "Charlie"

Message Env

Message Body

Message Text

"Hello, world!"

Crypto Legend

Boxed (Public Key Crypto)

Server -> Server

Tell <-> Server

Tell/Server -> Envelope

Tell -> Body

Secret-Boxed (Shared Secret Crypto)

Conversation Envelope

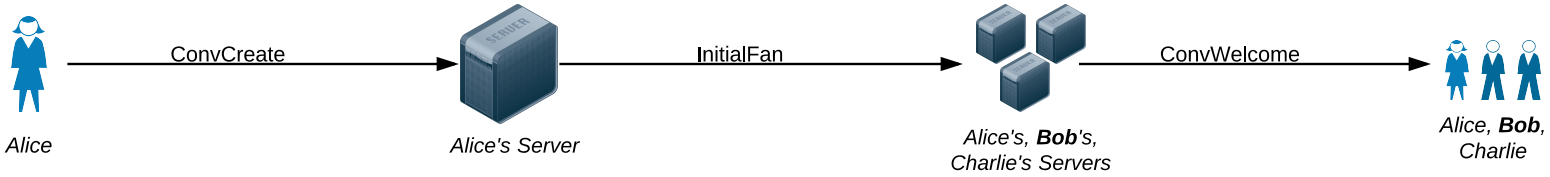
Conversation Body

Signed (Public Key Signature)

Signed

Conversation Creation

actions and validation: 1 of 1



Alice creates a new signing keypair whose public key serves as the identifier for the conversation. Alice also creates two shared secret keys: one for encrypting message envelopes sent to the conversation, and one for encrypting message bodies.

For each participant (including herself), Alice generates:

- An "Invite Proof" message encrypted to the participant's server that names the conversation and participant.
- A layered "Invite" message encrypted to the participant's envelope and body keys

  Invite Env #A
  Conversation Envelope Key
  Invite Body #A
  Conversation Body Key

- A layered "Join" message for the conversation encrypted using the conversation's secret keys. The payload of the body is an attestation signed by Alice that certifies the self-identity of the Invitee and provides Alice's own name for that person.

  Join Env #A
  Join Body #A
  Attestation #A
  key A is "Alice"

Finally, Alice generates a message to the conversation containing the text she wanted the other participants to read.

Message Env
Message Body
Message Text
"Hello, world!"

Alice's server validates that the message is from one of its users.

Alice's server validates the conversation is well formed in that Alice is first adding herself to the conversation.

Alice's server verifies that there is no existing conversation with the given id.

For each participant (including Alice), Alice's server generates an InitialFan message for the Participant's server consisting of the "Invite Proof" and a ConvWelcome encrypted to the participant's Envelope key. The ConvWelcome message consists of the "Invite" message that contains the conversation keys, all of the "Join" messages for the initial participants, and the "Message" with Alice's text.

Bob's server validates the "Invite Proof" by making sure that it was sent by Alice, that Bob has a contact relationship with Alice, and that the proof names the conversation and Bob.

(The ConvWelcome is passed from the maildrop side of the implementation to the mailstore side of the implementation.)

Bob's server processes the ConvWelcome, relaying the "Invite", "Join"s, and "Message" to Bob as separate messages.

Bob's client processes the received messages and builds its own local representation of the state of the messages.

Conversation Invitations

data, actions, and valdiation: 1 of 4

Sends...

convMessage
- ConvJoin => *Dave's Server*
  - Resend => *Bob's Server*
    - ConvAdd => *Conversation Server*
      - Invite Env #D
        - *Conversation Envelope Key*
        - Invite Body #D
          - *Conversation Body Key*
      - Join Env #D
        - Join Body #D
          - Attestation #D
            - *key D is "Dave"*

Bob creates a layered message which consists of:

- A ConvAdd message to Alice's (conversation hosting) server containing:
  - A layered invitation message to Dave that contains the conversation's shared keys and meta information.
  - A layered conversation "Join" message that provides Bob's other-person identity for Dave and notifying them that Dave has been looped into the conversation.
- A "Resend" message that wraps the ConvAdd message and is in turn wrapped by the ConvJoin message. Our goal is to not send the ConvAdd message to the conversation server until we are sure that Dave has authorized the conversation. We bounce the (wrapped) payload around to this end, although it could maintain local state and send an opaque identifier instead.

Sees...

convMessage
- (ConvJoin)

Sends...

(ConvJoin)

Bob's server relays the Join message to Dave's server.

Sees...

ConvJoin
- (Resend)

Sends...

ConvJoined
- (Resend)

Dave's server validates the join request by ensuring that Bob is a contact of Dave and that Box generated the join request. This is analogous to the "Invite Proof" of the conversation creation process.

Dave's server authorizes the receipt of messages for the conversation from Alice's server and retransmits the Resend payload back to Bob's server.

Sees...

ConvJoined
- Resend
  - (ConvAdd)

Bob's server validates the Resend message, and pulls out its payload (a ConvAdd message), to transmit to Alice's conversation server.

Crypto Legend

Boxed (Public Key Crypto)	Secret-Boxed (Shared Secret Crypto)
Server -> Server	Conversation Envelope
Tell <-> Server	Conversation Body
Tell/Server -> Envelope	
Tell -> Body	

Signed (Public Key Signature)

Signed

Conversation Invitations

data, actions, and valdiation: 2 of 4

Sends...

(ConvAdd)

Bob's server retransmits the contents of its Resend message to Alice's (converation) server.

Sees...

ConvAdd

(Invite Env #D)

(Join Env #D)

Sends... #1

FannedMsg *=> Dave's Server*

ConvWelcome #D *=> Dave*

(Invite Env #D)

Backlog:

(Join Env #A)

(Join Env #B)

(Join Env #C)

(Message Env)

Alice's server validates that the request is from a participant in the conversation.

A ConvWelcome message is generated for Dave containing the Invite message created by Bob (which contains the conversation secret keys) and the message backlog not including this new join message.

The new join message is issued separately; see the next page.

Sees...

FannedMsg

ConvWelcome #D

Invite Env #D

Conversation Envelope Key

(Invite Body #D)

Backlog:

Join Env #A

(Join Body #A)

Join Env #B

(Join Body #B)

Join Env #C

(Join Body #C)

Message Env

(Message Body)

Dave's server verifies that the FannedMsg is from an authorized server and for an authorized conversation.

(The ConvWelcome is passed to the mailstore layer.)

Dave's server processes the ConvWelcome, relaying the "Invite", "Join"s, and "Message" to Dave as separate messages.

Crypto Legend

Boxed (Public Key Crypto)

Server -> Server

Tell <-> Server

Tell/Server -> Envelope

Tell -> Body

Secret-Boxed (Shared Secret Crypto)

Conversation Envelope

Conversation Body

Signed (Public Key Signature)

Signed

Conversation Invitations

data, actions, and valdiation: 3 of 4

Sends...

- Invite Env #B
  - *Conversation Envelope Key*
  - (Invite Body #B)
- Join Env #A
  - (Join Body #A)
- Join Env #B
  - (Join Body #B)
- Join Env #C
  - (Join Body #C)
- Message Env
  - (Message Body)

Sees...

- Invite Env #B
  - *Conversation Envelope Key*
  - Invite Body #B
    - *Conversation Body Key*
- Join Env #A
  - Join Body #A
    - Attestation #A
      - *key A is "Alice"*
- Join Env #B
  - Join Body #B
    - Attestation #B
      - *key B is "Bob"*
- Join Env #C
  - Join Body #C
    - Attestation #C
      - *key C is "Charlie"*
- Message Env
  - Message Body
    - Message Text
      - *"Hello, world!"*

Dave's client processes the received messages and builds its own local representation of the state of the messages.

Sends... #2

- FannedMsg *=> Bob's Server*
  - ConvJoin *=> Bob*
    - (Join Env #D)

The conversation server creates a ConvJoin message that contains the layered "Join" message for each participant and sends it.

Sees...

- FannedMsg
  - ConvJoin
    - Join Env #D
      - (Join Body #D)

Bob's server verifies that the FannedMsg is from an authorized server and for an authorized conversation.

(The ConvJoin is passed to the mailstore layer.)

Bob's server processes the ConvJoin, relaying the "Join" message to Bob.

Crypto Legend

Boxed (Public Key Crypto)

- Server -> Server
- Tell <-> Server
- Tell/Server -> Envelope
- Tell -> Body

Secret-Boxed (Shared Secret Crypto)

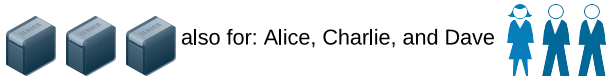
- Conversation Envelope
- Conversation Body

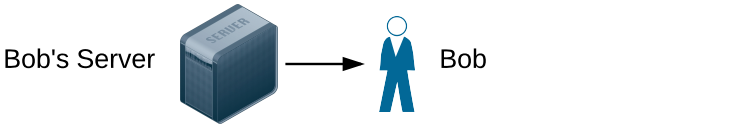
Signed (Public Key Signature)

- Signed

Conversation Invitations

data, actions, and valdiation: 4 of 4





Sends...

Join Env #D

(Join Body #D)

Sees...

Join Env #D

Join Body #D

Attestation #D

key D is "Dave"

Bob's client processes the received message and builds its own local representation.

Crypto Legend

Boxed (Public Key Crypto)

Server -> Server

Tell <-> Server

Tell/Server -> Envelope

Tell -> Body

Secret-Boxed (Shared Secret Crypto)

Conversation Envelope

Conversation Body

Signed (Public Key Signature)

Signed

Sending Messages (Human and Meta)

data, actions, and valdiation: 1 of 2

Sends...

convMessage

ConvMsg => *Conversation Server*

Message Env #2

Message Body #2

Message Text #2

"Yay, a world!"

Charlie creates his message text and signs it, wrapping it in messages encrypted with the conversation's shared secret body and envelope keys.

He then creates and sends a message to Alice's (conversation hosting) server containing the conversation message.

Sees...

convMessage

(ConvMsg)

Charlie's server relays the ConvMsg to Alice's (conversation hosting) server.

Sends...

(ConvMsg)

Sees...

ConvMsg

(Message Env) #2

Alice's server validates that the message is from a participant in the conversation.

The conversation server creates a ConvJoin message that contains the layered "Join" message for each participant and sends it.

Crypto Legend

Boxed (Public Key Crypto)

- Server -> Server
- Tell <-> Server
- Tell/Server -> Envelope
- Tell -> Body

Secret-Boxed (Shared Secret Crypto)

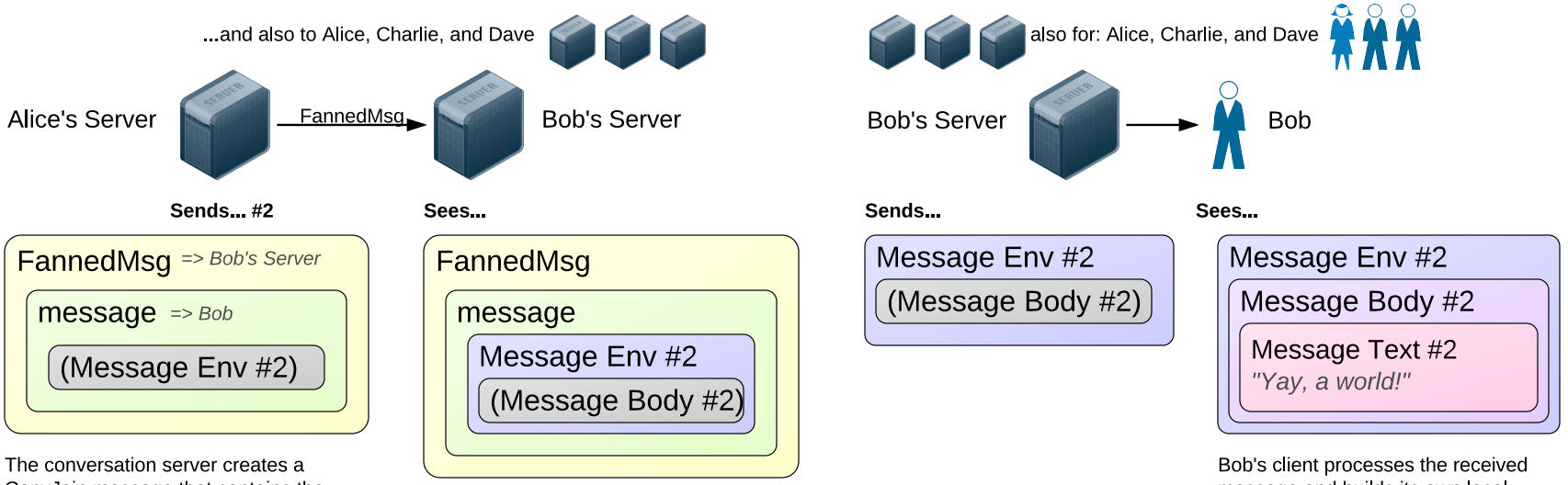
- Conversation Envelope
- Conversation Body

Signed (Public Key Signature)

- Signed

Sending Messages (Human and Meta)

data, actions, and valdiation: 2 of 2



The conversation server creates a ConvJoin message that contains the layered "Join" message for each participant and sends it.

Bob's server verifies that the FannedMsg is from an authorized server and for an authorized conversation.

(The ConvJoin is passed to the mailstore layer.)

Bob's server processes the "message", relaying the layered conversation message "Message Env" onwards to Bob.

Bob's client processes the received message and builds its own local representation.

Crypto Legend

Boxed (Public Key Crypto)

- Server -> Server
- Tell <-> Server
- Tell/Server -> Envelope
- Tell -> Body

Secret-Boxed (Shared Secret Crypto)

- Conversation Envelope
- Conversation Body

Signed (Public Key Signature)

- Signed