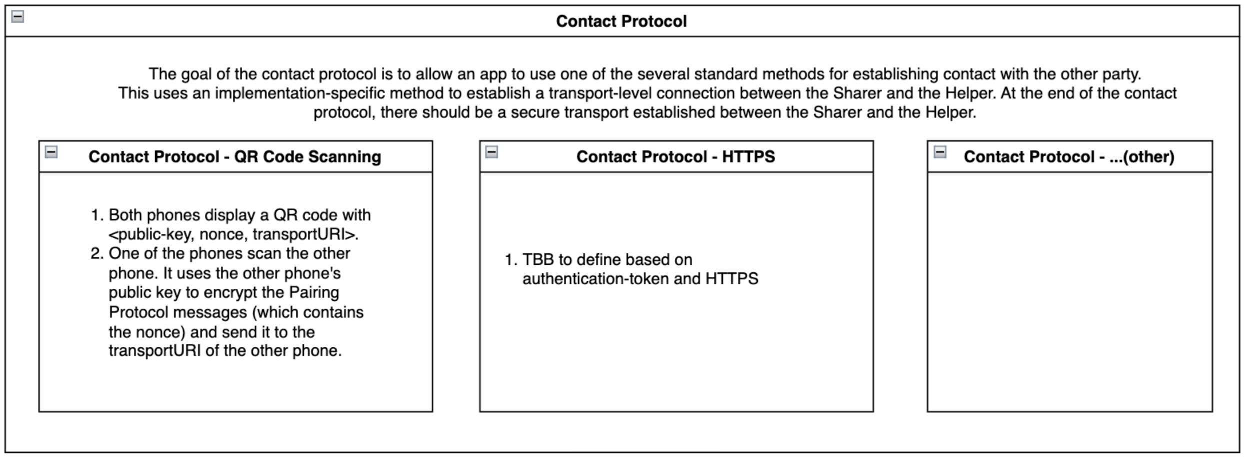
DeRec Protocol

State Diagrams

Contact Protocol



Contact Protocol

The goal of the contact protocol is to allow an app to use one of the several standard methods for establishing contact with the other party.
This uses an implementation-specific method to establish a transport-level connection between the Sharer and the Helper. At the end of the contact protocol, there should be a secure transport established between the Sharer and the Helper.

Contact Protocol - QR Code Scanning

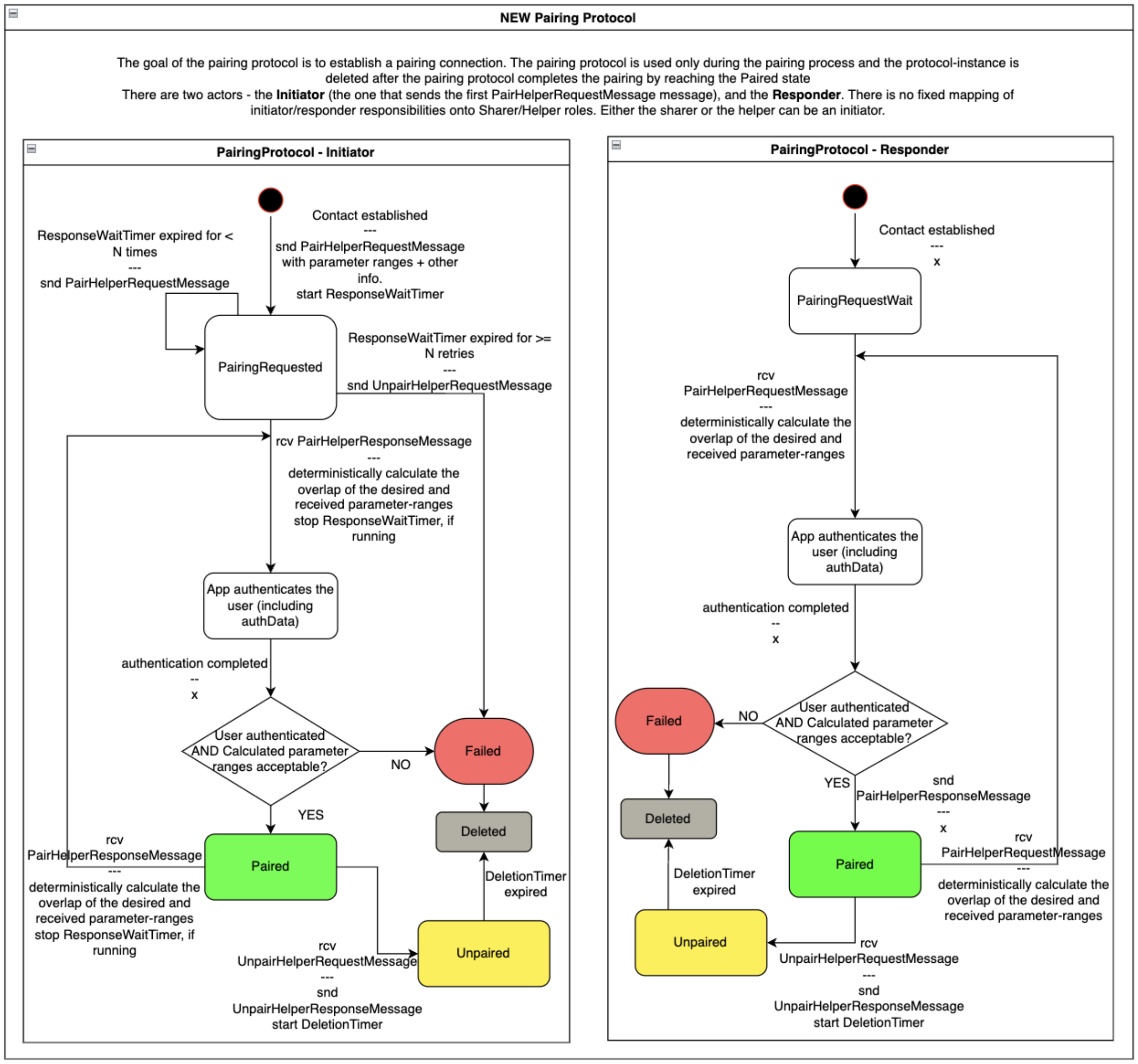
1. Both phones display a QR code with <public-key, nonce, transportURI>.
2. One of the phones scan the other phone. It uses the other phone's public key to encrypt the Pairing Protocol messages (which contains the nonce) and send it to the transportURI of the other phone.

Contact Protocol - HTTPS

1. TBB to define based on authentication-token and HTTPS

Contact Protocol - ...(other)

Pairing Protocol

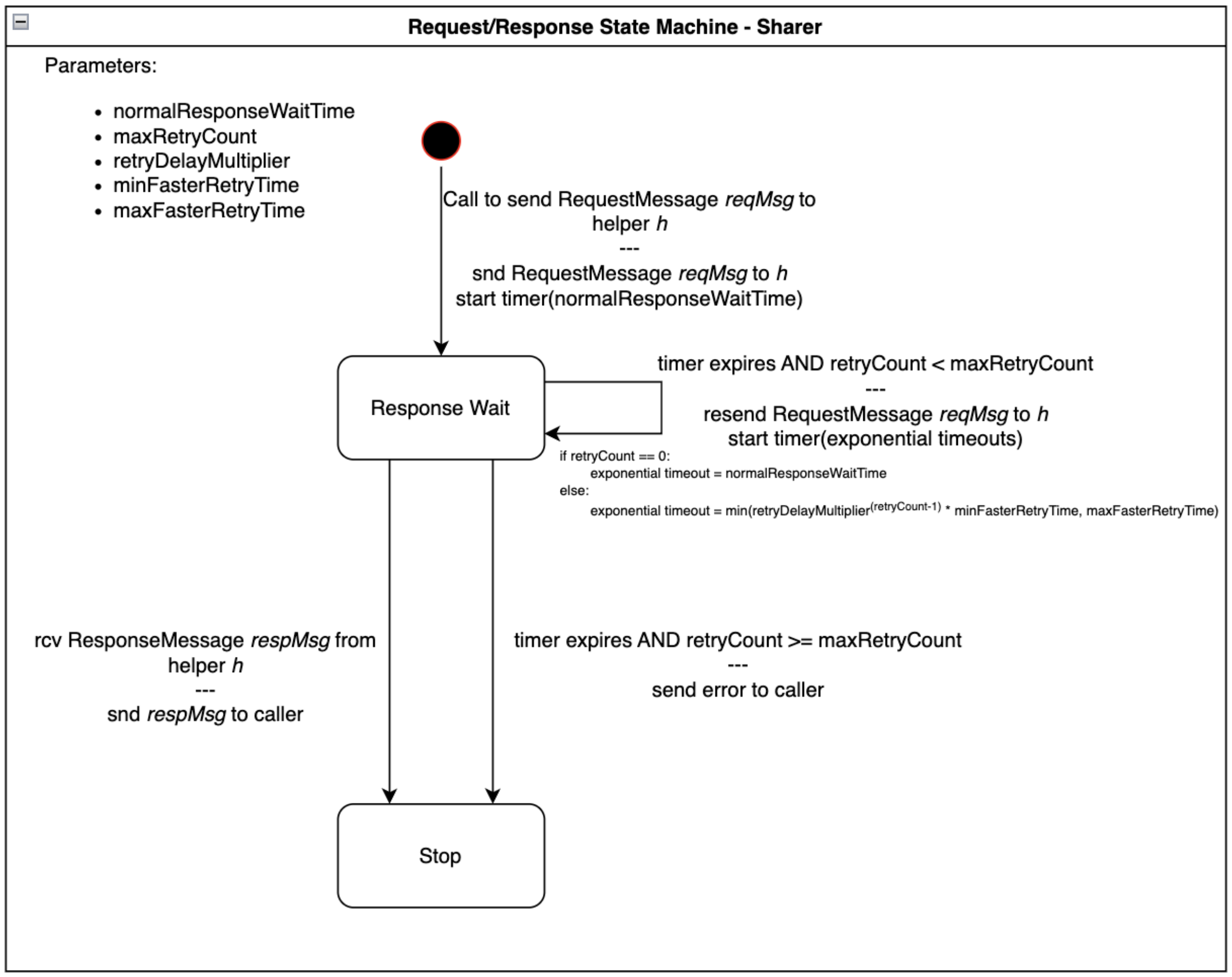


NEW Pairing Protocol

The goal of the pairing protocol is to establish a pairing connection. The pairing protocol is used only during the pairing process and the protocol-instance is deleted after the pairing protocol completes the pairing by reaching the Paired state
There are two actors - the **Initiator** (the one that sends the first PairHelperRequestMessage message), and the **Responder**. There is no fixed mapping of initiator/responder responsibilities onto Sharer/Helper roles. Either the sharer or the helper can be an initiator.

PairingProtocol - Initiator
Contact established --- snd PairHelperRequestMessage with parameter ranges + other info. start ResponseWaitTimer
ResponseWaitTimer expired for < N times --- snd PairHelperRequestMessage
PairingRequested
ResponseWaitTimer expired for >= N retries --- snd UnpairHelperRequestMessage
rcv PairHelperResponseMessage --- deterministically calculate the overlap of the desired and received parameter-ranges stop ResponseWaitTimer, if running
App authenticates the user (including authData)
authentication completed -- x
User authenticated AND Calculated parameter ranges acceptable?
NO
YES
Failed
Deleted
Paired
rcv PairHelperResponseMessage --- deterministically calculate the overlap of the desired and received parameter-ranges stop ResponseWaitTimer, if running
DeletionTimer expired
Unpaired
rcv UnpairHelperRequestMessage --- snd UnpairHelperResponseMessage start DeletionTimer
PairingProtocol - Responder
Contact established --- x
PairingRequestWait
rcv PairHelperRequestMessage --- deterministically calculate the overlap of the desired and received parameter-ranges
App authenticates the user (including authData)
authentication completed -- x
User authenticated AND Calculated parameter ranges acceptable?
NO
Failed
Deleted
YES
snd PairHelperResponseMessage --- x
Paired
rcv PairHelperRequestMessage --- deterministically calculate the overlap of the desired and received parameter-ranges
DeletionTimer expired
Unpaired
rcv UnpairHelperRequestMessage --- snd UnpairHelperResponseMessage start DeletionTimer

Request/Response State Machine


Request/Response State Machine - Sharer
Parameters:
• normalResponseWaitTime
• maxRetryCount
• retryDelayMultiplier
• minFasterRetryTime
• maxFasterRetryTime
Call to send RequestMessage *reqMsg* to helper *h*
---
snd RequestMessage *reqMsg* to *h*
start timer(normalResponseWaitTime)
Response Wait
timer expires AND retryCount < maxRetryCount
---
resend RequestMessage *reqMsg* to *h*
start timer(exponential timeouts)
if retryCount == 0:
exponential timeout = normalResponseWaitTime
else:
exponential timeout = min(retryDelayMultiplier$^{(retryCount-1)}$ * minFasterRetryTime, maxFasterRetryTime)
rcv ResponseMessage *respMsg* from helper *h*
---
snd *respMsg* to caller
timer expires AND retryCount >= maxRetryCount
---
send error to caller
Stop

Messaging Tables

Terms used (from RFC 2119):

- MUST (have to do it)
- SHOULD (recommended unless a good reason not to)
- MAY (an app can do it if the developer chooses to)

Sharer

Phase	Event	Action
Pairing	Contact established as initiator	Send PairRequestMessage to the contact.
Pairing	PairResponseMessage received	App MUST authenticate the user. MUST calculate parameter ranges overlap, and the parameters to use. The “result” field of the response MUST be FAIL if there is no overlap, or for any other reason to not pair.
Pairing	Application unpairs a helper	MUST Send UnpairRequestMessage
Pairing	UnpairResponseMessage received	If successful, MUST delete all data structures for this helper after a brief timeout (unless regulatory compliance requires it to be stored). MUST take the appropriate action for the reduction in number of helpers, which is to either reshare with the remaining helpers (if there are still enough of them), or stop sharing entirely and warn the user (if there are not enough).
Verification	Periodic timer for verification	MUST Send VerifyShareRequestMessage to all peers for whom the shouldSendNewShares flag is true.
Verification	VerifyShareResponseMessage received	If the response has an incorrect hash, MUST resend the correct share to the helper N times. There MUST be a verify after each share (MAY be in the same DeRecMessage).

Verification	VerifyShareResponseMessage is not received	MUST Retry M times, starting with the fast period (of length P), and getting exponentially longer time periods (multiplying by K each time, up to a maximum of Q). (Parameters M and Q are -1 to indicate infinity). The library SHOULD give the app the ability to choose M,P,K,Q, or it MAY make some or all of them fixed. If there's no response too many times then they could eventually be unpaired, at which point the shouldSendNewShares flag is set to false. But MUST keep retrying verification at a very slow frequency, until then. As the number of active helpers decreases, then the sharing will have a lower threshold for recovery, for some apps, and other apps will keep a fixed threshold for recovery. Either way, the user MUST be warned when too few helpers are active. If the recovery threshold is reduced as the active helpers set shrinks, then it MUST never allow the recovery threshold to drop below the security threshold. Instead, it will clip at the security threshold. Every time it is shared, it SHOULD share with everyone with shouldSendNewShares true.
Sharing	User updates their secret OR (Helper paired/unpaired OR Helper became Active/Inactive (in a way that changes the recovery threshold)).	MUST Reshare the secret shares (send StoreShareRequestMessage) to helpers with shouldSendNewShares true, when their frequency limits allow it. If a given helper only allows infrequent updates, then it may be that during the period between updates, there are several sharings that happen. When the period ends, that helper MUST actually be sent

		only the last version. The earlier versions will be skipped. The library MUST inform the app which versions are reliably stored and which are not.
Sharing	StoreShareResponseMessage received	If enough helpers have replied that the new version is safely stored (a parameter specified to the library), then the library MUST update the keepList to clean up old versions (send StoreShareRequestMessage with the updated keepList to all helpers).
Recovery	Helper paired in recovery mode	MUST send GetSecretIdsVersionsRequestMessage to the newly-paired helper.
Recovery	GetSecretIdsVersionsResponseMessage received	MUST send GetShareRequestMessage for each (secretId, version) pair to all helpers for whom we don't yet have this (secretId, version). MUST inform the application of whether this helper had anything, and what they had.
Recovery	GetShareResponseMessage received	MUST Attempt combining the shares. If successful, inform the application. Switch to normal mode.

Helper

Phase	Event	Action
Pairing	PairRequestMessage received after establishing contact (responder)	MUST app authenticates the user. Calculate parameter ranges overlap. Send PairResponseMessage. MUST store whether this secret was paired in recovery mode or not. If the pairing request is in recovery mode, then this secret MUST be connected to all

		other secrets by that same user (so that when the list of secrets and versions is requested, it will return the complete list of secrets for that user).
Pairing	UnpairRequestMessage received	MUST Send UnpairResponseMessage. This will now be out of use, as if the pairing hadn't happened, so the sharer will not be able to recover it. Also, MAY delete all data structures for this sharer (unless regulatory compliance, or their service contract, requires it to be stored).
Verification	VerifyShareRequestMessage received	MUST Calculate the hash, and send VerifyShareResponseMessage with the appropriate error code.
Sharing	StoreShareRequestMessage received	MUST Store/Update share version. MUST delete unneeded versions (outside the keepList). Send StoreShareResponseMessage.
Recovery	GetSecretIdsVersionsRequestMessage received	MUST Send GetSecretIdsVersionsResponseMessage with all secrets and versions for this sharer. This MUST only reveal other secret IDs if it is received through the communication channel of a secretID paired in recovery mode. Otherwise, it only returns the versions associated with the current secretID, with a result status of PARTIAL.
Recovery	GetShareRequestMessage received	MUST send GetShareResponseMessage with the share and appropriate error code. This MUST only return the share if it is either requested through the same secret ID that is being requested, or is requested through a secret ID that was paired in recovery mode.