# MSC2228 (self-destructing events) implementation notes

In the rest of this document, “I” = Brendan Abolivier

These notes describe how one would go and implement the part of MSC2228 that isn’t already covered by https://github.com/matrix-org/synapse/pull/6409, i.e. implementing synthetic redactions and support for `m.self_destruct`, in Synapse. In here, I sometimes use the expression “ephemeral message” to describe self-destructing event, and “expiring” to mean “self-destructing”.

This plan makes one major assumption, which IIRC is left as an implementation detail in the MSC, which is what to do with people joining the room before the event if the history visibility setting of the room isn't `joined`. My opinion is that the server should always show them the synthetic redaction, so the event isn't "revived" each someone joins. We could also think about a slight variant of this strategy where, if the room's history visibility is `invited`, we also consider the users that were invited at the time the event was sent.

It’s also likely that synthetic redactions will need their own stream ID (and entry in the sync token), in order to replicate them to workers correctly. We could also piggyback on the events stream.

First, I think we should create a new table, let's name it `ephemeral_messages` for this explanation. Each time we receive an ephemeral message, we'd store in this table:

- The stream ID
- The event ID
- The room ID
- The value for `m.self_destruct`

We'd also need another table, let's say `synthetic_redactions`, to store synthetic redactions, more specifically:

- The stream ID at which the synthetic redaction was generated, in the synthetic redactions’ own stream, or in the events stream, depending on which solution is chosen
- The event ID for the event to redact
- The user the synthetic redaction is for

We’d also need a third table, let’s call it `ephemeral_messages_expiry`, which would store the expiration timer for each `(event_id, user_id)` tuple, more specifically:

- The event ID
- The user ID

- The timestamp, in milliseconds, of when the event should be synthetically redacted for the user

This list might be expanded to include whatever is necessary to generate a synthetic redaction.

Then, each time we get a read receipt:

1. Fetch the stream ID of the previous read receipt for the same user in the same room
2. Check if there's an entry in `ephemeral_messages_readers` for this room which stream ID is after the one fetched in the previous step
3. If there is at least one entry, for each of them, as a background process:
    1. Check if there's already an entry in the `synthetic_redactions` table for this user and this event, if not continue to the next event
    2. Fetch the list of members in the room when the event was sent
    3. Check if the user is in this list of members, if not continue to the next event
    4. Add a row in `ephemeral_messages_expiry` for this user and this event, and the timestamp of now + the amount of time specified in `m.self_destruct`
    5. Wait for the amount of time specified in `m.self_destruct`
    6. Delete the row in `ephemeral_messages_expiry` for this user and this event
    7. Generate an entry in `synthetic_redactions` for this event and this user
    8. Notify the sync so the redaction is sent to clients

At startup, we would need a background job to fetch all the rows from `ephemeral_messages_expiry` and, for each of them, run steps 3.5 to 3.8 of the algorithm described above.

We would also need to introduce some changes in `filter_events_for_client` so that, if the event its sub-function `allowed` is an ephemeral message:

- If the user wasn't a member of the room when the event was sent, replace it with a redacted or pruned version of the event
- Otherwise:
    - If there is a synthetic redaction stored for this user and this event, replace it with a redacted/pruned version of the event
    - Otherwise just return the event as is (modulo additional checks in this function)

The sync handler would also need to read from the `synthetic_redactions` table and insert synthetic redactions at the right stream IDs.

This way when a user syncs, for each expired ephemeral message:

- If the user wasn't a member when the message was sent, they will only see the redacted/pruned event
- Otherwise:

    - If the sync is incremental and the sync token is after the event is sent, they will see the synthetic redaction and their client should process it as a normal `m.room.redact` event
    - Otherwise they will see both the redacted/pruned event and the synthetic redaction

Additionally, we could also add a looping background process, which, for each ephemeral message would check the event has been seen by every user that was in the room when it was sent, in which case we could plan censoring it like we currently do with redactions, as well as removing its entry in `ephemeral_messages`.

## Potential issues

The MSC doesn't mention what to do when a user opens a room they haven't opened for a while and contains self-destructing messages. The issue it creates is that, if there have been a lot of messages sent to the room since the last time they've opened it, the message might expire before they get a chance to read them. This is a non-trivial issue since only clients know exactly what the user is seeing. A server-side mitigation could be to try to figure out how many events there have been between an ephemeral message and the latest message in the room, and try to adjust the expiration timer, but that sounds wobbly and possibly too complicated, on top of being in contradiction with the MSC. I believe a mitigation for that would have to happen client-side, e.g. by setting the view of the room for the user to the position of their last read receipt, either always or as long as there are ephemeral events in the room.