Encrpting local app files

Problem

Currently, the files are stored in the file storage unencrypted, unlike the database.

There are multiple operations in the app that access files:

1. Sending files via SMP - chat core reads the files chunk by chunk and sends them. The file can be encrypted once sent and the "encrypted" flag added.

2. Sending files via XFTP - simplexmq encrypts the file first and then sends it. Currently, we are deleting the file from chat, once its uploaded, there is no reason to keep unencrypted file (from XFTP point of view) once its encrypted.

3. Viewing images in the mobile apps.

4. Playing voice files in the mobile apps.

5. Playing videos and showing video previews in mobile apps.

6. Saving files from the app storage to the device.

Possible solutions

System encryption

A possible approach is to use platform-specific encryption mechanism. The problem with that approach is inconsistency between platforms, and that the files in chat archive will probably be unencrypted in this case.

App encryption

Files will be encrypted once received, using storage key, and the core would expose C apis to mobile apps:

1. Read the file with decryption - this can be used for image previews, for example, as a replacement for OS file reading.

2. Copy the file with decryption to some permanent destination - this can be used for saving files to the device.

3. Copy the file into a temporary location with decryption - this can be used for playing voice/video files. The app would remove the files once no longer used, and this temporary location can be cleaned on each app start, to clean up the files that the app failed to remove. Alternative to

that would be to have both encrypted and decrypted copies available for the file, with paths stored in the database, and clean up process removed decrypted copies once no longer used - there should be some flags to indicate when decrypted copy can be deleted.

For specific use cases:

1. Viewing images in the mobile apps.
    - iOS: we use `UIImage(contentsOfFile path: String)`. We could use `init?(data: Data)` instead, and decrypt the file in memory before passing it to the image view. Images are small enough for this approach to be ok, and in any case the image is read to memory as a whole.
    - Android: we use `BitmapFactory.decodeFileDescriptor` (?). We could use ...
2. Playing voice files in the mobile apps.
    - iOS: we use `AVAudioPlayer.init(contentsOf: URL)` to play the file. We could either decrypt the file before playing it, or, given that voice files are small (even if we increase allowed duration, they are still likely to be under 1mb), we could use `init(data: Data)` to avoid creating decrypted file.
    - Android: we use `MediaPlayer.setDataSource(filePath)`. We could use ...
3. Showing video previews.
    - iOS: ...
    - Android: ...

    Possibly, we will need to store preview as a separate file, to avoid decrypting the whole video just to show preview.
4. Playing video files.
    - iOS: we use `AVPlayer(url: URL)`, the file will have to be decrypted for playback.
    - Android: ...
5. Saving files from the app storage to the device. The file will have to be decrypted, passed to the system, and then decrypted copy deleted once no longer needed.

Which key to use for encryption

1. Derive file encryption key from database storage key. The downside for this approach is managing key changes - they will be slow. Also, if file encryption is made optional, and in any case, for the existing users all files are not encrypted yet, we will need somehow to track which files are encrypted.

2. Random per-file encryption key stored in the database. Given that the database is already encrypted, it can be a better approach, and it makes it easier to manage file encryption/decryption. File keys will not be sent to the client application, but they will be accessible via the database queries of course.