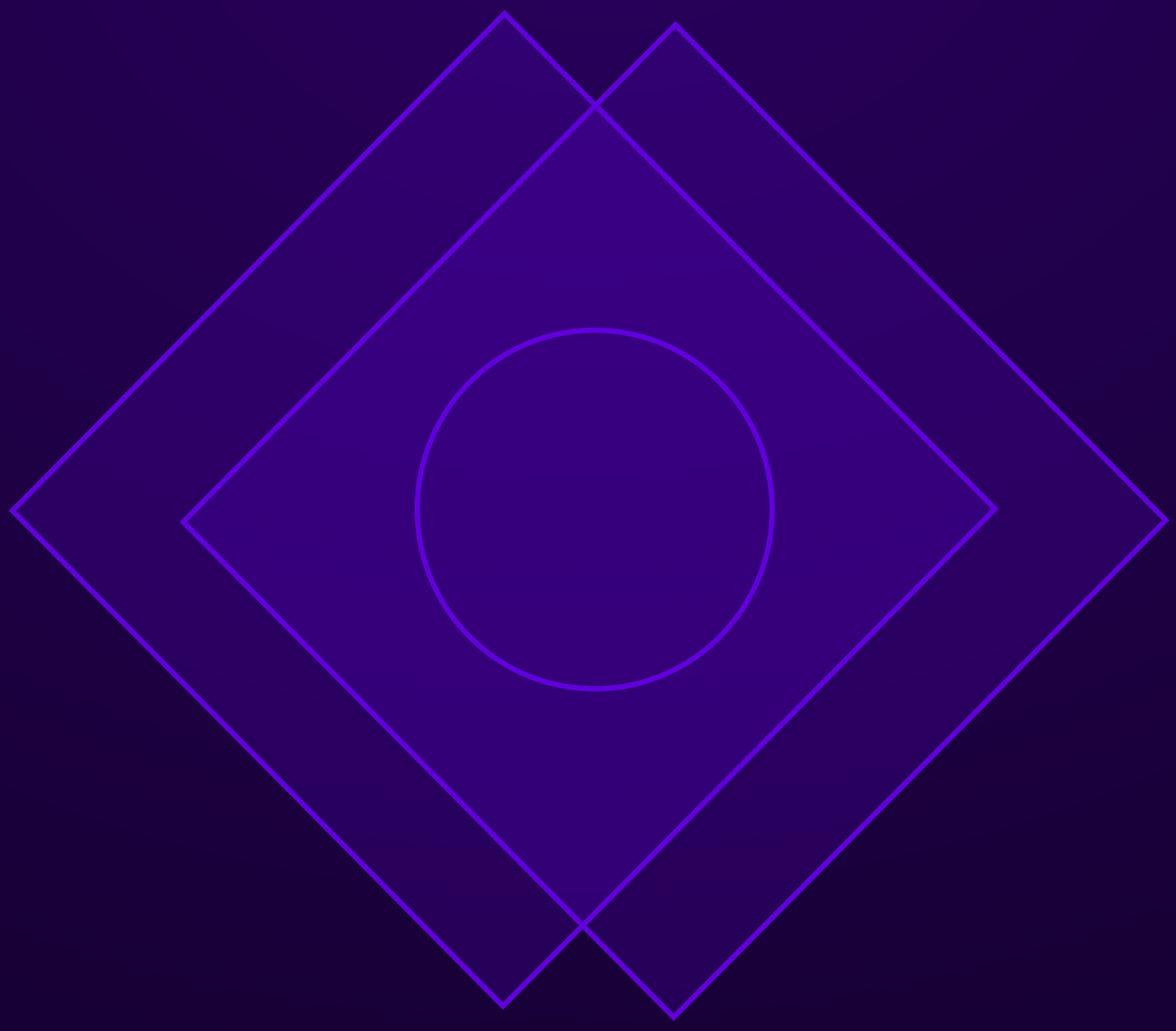


RING RELAY

Technical Documentation

[April 2023]

Table of Contents

Purpose

What is this section answering

- ☐ What is the Ring Relay and its use-cases
- ☐ What features does it provide

The Ring Relay and its use-cases

The Ring Relay is an end-to-end encrypted messaging application that allows users to securely exchange messages of various types using public-key cryptography. For transparency purposes, this project is open-source and its architecture is explained in this document.

Features

Security:

- ☐ End-to-end encryption for all message types
- ☐ [Optional] Activity Logs
- ☐ Encrypted Local Private Keys Backup
- ☐ Security Signatures

Message Types:

- ☐ Text
- ☐ Images [Coming Soon]
- ☐ Color
- ☐ Location

Purpose

Features

Messaging Modes:

- ☒ Normal
- ☒ Ghost

Account Management

- ☒ Change Username
- ☒ Change Password
- ☒ Log Collection Controls
- ☒ Notifications Config
- ☒ Key Pairs Management
- ☒ Key Pairs Regeneration
- ☒ Account Data Controls

Application Architecture

What is this section answering

- ☒ What services is the Ring Relay using
- ☒ How are these services linked and what purpose do they have
- ☒ How is authentication handled
- ☒ How are the cryptographic features implemented

Application Architecture

What services is the Ring Relay using

In order for the Ring Relay to support all the features presented previously, this application uses various services such as:

Vercel Serverless Functions

Vercel Serverless Functions act like the middle-man between the user and other services such as the databases and a 3rd party notification service. Those functions also contain the logic for how to handle various tasks a user requests by using the app while handling authentication and ensuring the identity of the user by several means.

Firebase Realtime Database

This database is used as the backbone for the real-time message buffer that allows users to receive live updates for actions like message reactions, message deletions and more.

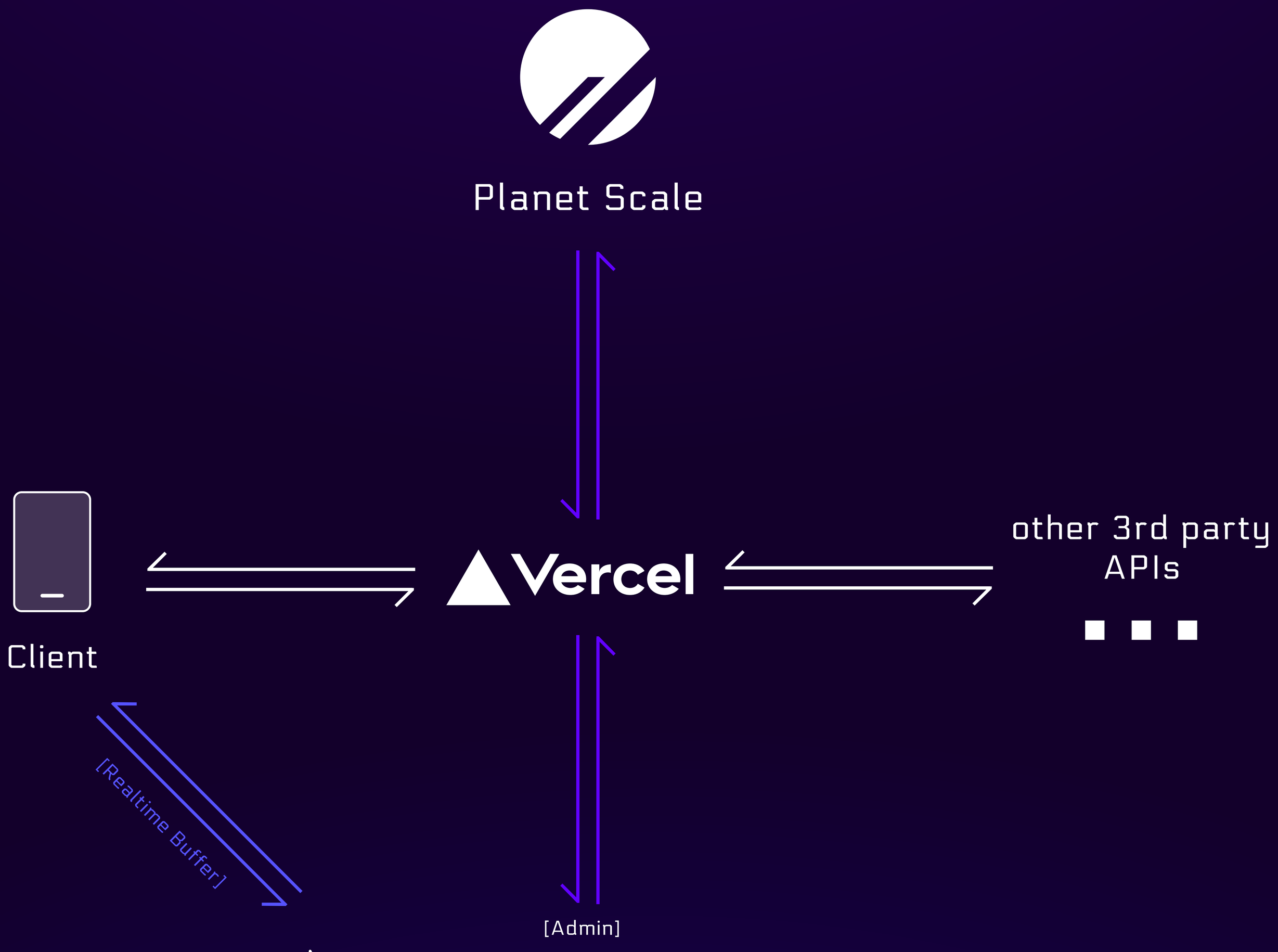
Besides real-time communication between users, this service is also used for storing temporary security tokens used for authentication and other features.

Planet Scale Database

This is the main database that stores all permanent data such as user account info, encrypted conversations, and any other data required for the features this app provides.

Application Architecture

How are those services linked



Web Sockets

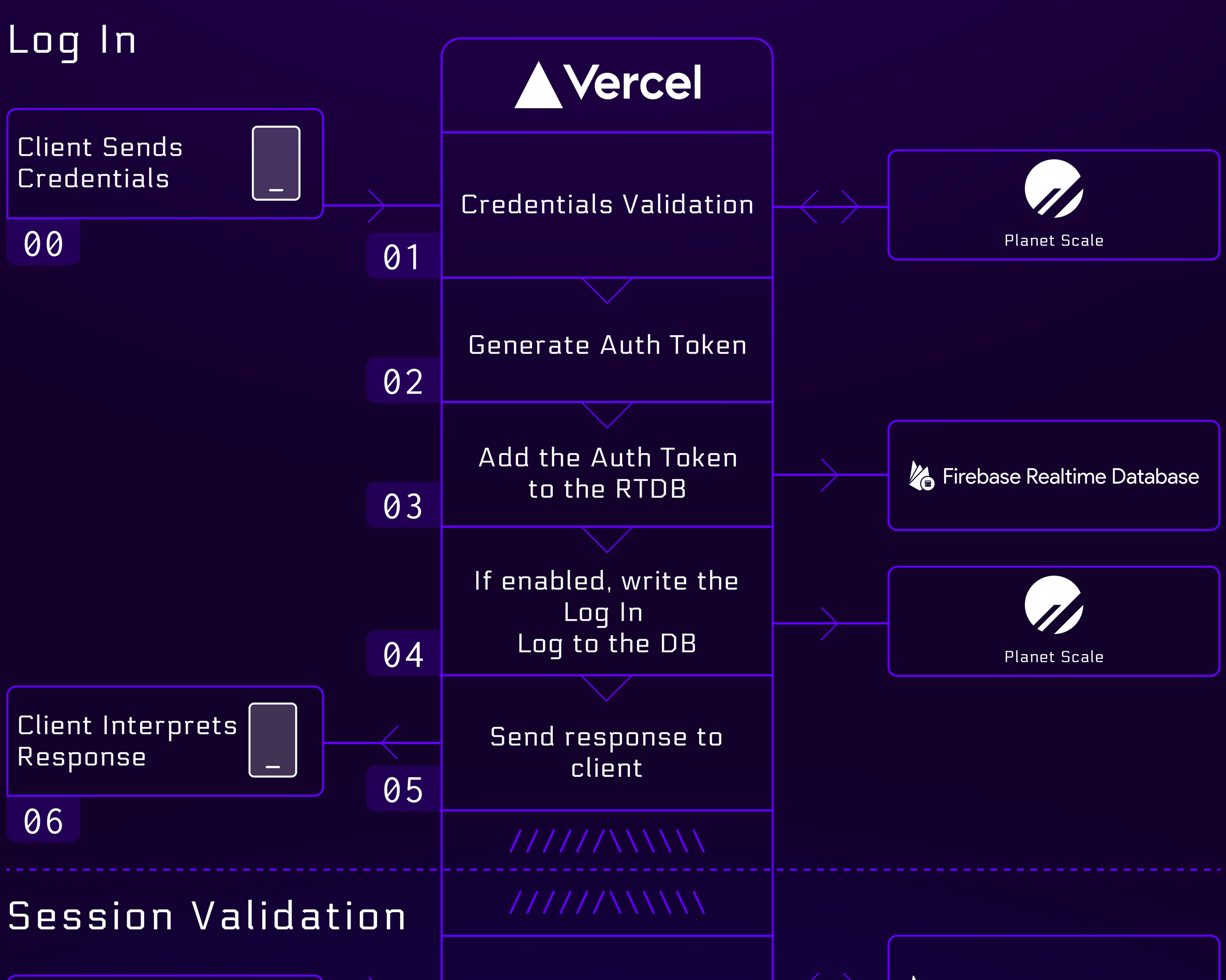
NPM Package

POST/GET Request

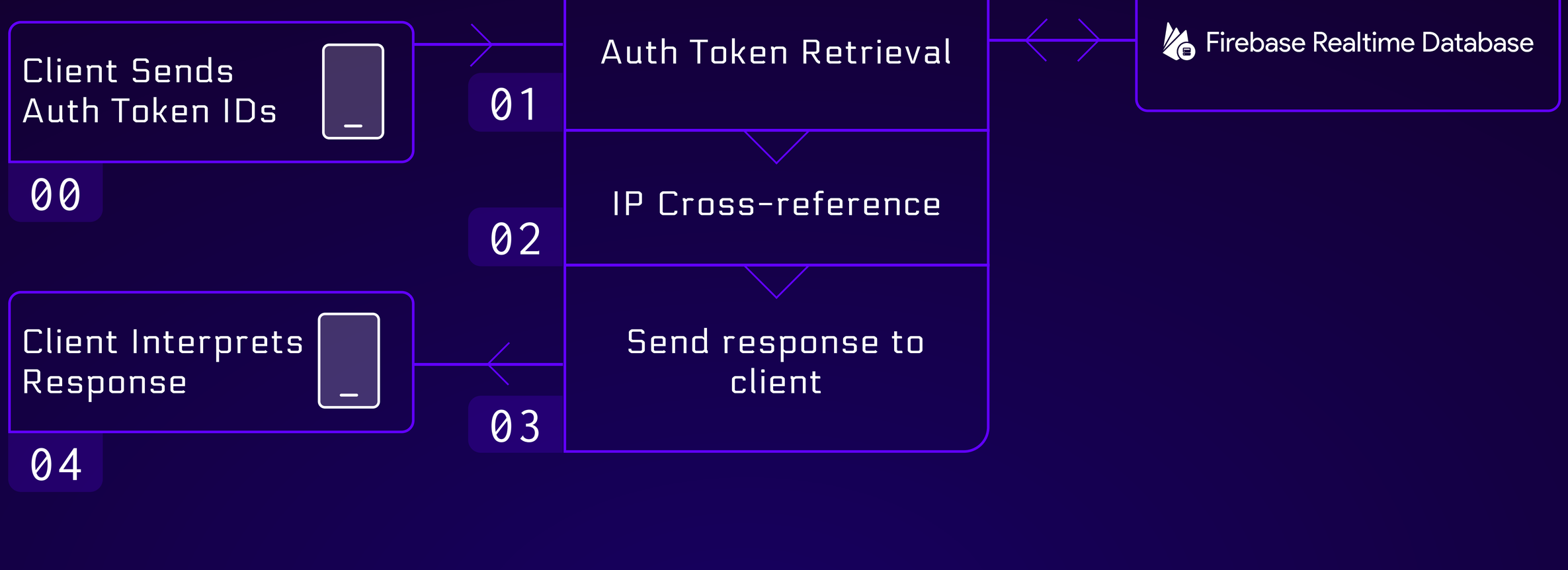
Application Architecture

Authentication

Log In



Session Validation



Application Architecture

Cryptographic Features

Cryptographic Algorithms

Since implementing these algorithms securely is incredibly difficult to do right, this application uses the Subtle Crypto API, which is a standard API across all browsers that offers the primitives required to build cryptographic systems.

Using secure algorithms is useless if the wider architecture is compromised or if somehow data that was supposed to stay secret gets leaked. The next part of this section will detail all considerations made while building this cryptographic system from the building blocks provided by the Subtle Crypto API.

Implementation Considerations

This application uses both symmetric and asymmetric encryption depending on where its being used. Symmetric encryption means the same key can both encrypt and decrypt data, while with asymmetric encryption, different keys are required for the same process.

The table below contains the specifications of all algorithms used, as well as the use-case for each specific key or key pair.

Type	Algorithm	Specs	Use-case
Asymmetric	RSA-OAEP	Modulus Len: 4096 bytes Hash: SHA-256	End-to-end Message Encryption
Asymmetric	ECDSA	Named Curve: P-521	Message Origin and Integrity ensurer
Symmetric	AES-GCM PBKDF2	Hash: SHA-256	Local Private Keys Backup Encryption

Application Architecture

Cryptographic Features

Implementation Considerations

Both private keys for message encryption and message authentication are stored locally and are NEVER transmitted over the network. Here are the only ways to exfiltrate the private keys:

- ⊡ Use the app's export feature to generate QR Codes or a text file containing the encrypted keys
- ⊡ Gain physical access to the device to dump the contents of local storage to some other place
- ⊡ Use social engineering to convince the user to follow some steps to expose the PEMs (PEM is the format the keys are stored in)
- ⊡ Compromise the source code of the application to send the private keys to the bad actor's database

This is how the possibilities above are mitigated:

- ⊡ The app uses the account password as the seed for the key that symmetrically encrypts the private keys. When importing keys using this method, there has to be an authorization token in the RTDB that allows the transfer of the keys between devices. This transfer is always offline (via QR Codes or a text file) and the server is used just to validate the operation. Any ops involving the private keys are also logged as critical.
- ⊡ Since this application is intended for mobile devices only, a bad actor would have to gain physical access to the device, the passcode/biometrics to unlock the device, the same passcode/biometrics to enable developer mode, and a PC to access the browser's local storage.
- ⊡ To try and prevent social engineering attempts, there is a warning message in the console warning the users to never share any contents from this window with anyone.
- ⊡ Any push requests to the repository containing the source code have to be approved before actually being merged.

Application Architecture

Cryptographic Features

Implementation Considerations

Even if a bad actor would be able to somehow compromise any user's PEMs, there could be automatic checks that would prevent any real damage from occurring. Automatic logs related to the import/export of private keys that would be private could be collected to support the following process: in the case a device would have private keys that have never been transferred through legitimate channels, that device would have its auth tokens revoked but not before the server sending a remote wipe command that would delete the stolen private keys. After such a security incident would've been detected, the user would be notified to take action so the account could be resecured.

[This is under active development and will be released with a future update]

Other implementation considerations have their own sections

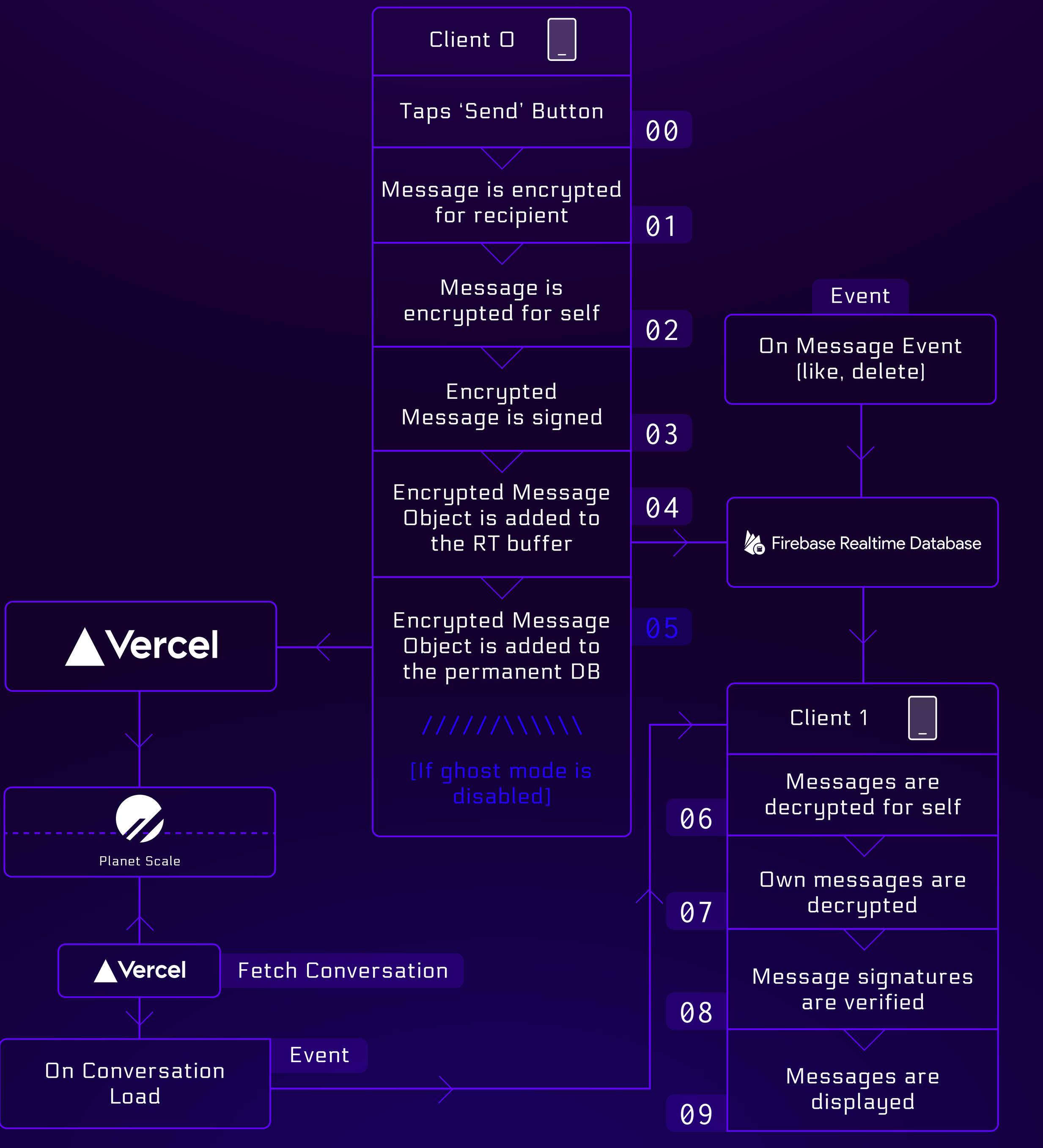
Messages

What is this section answering

- ⊡ How are messages exchanged
- ⊡ How are messages encrypted and authenticated
- ⊡ Messaging Modes

Messages

Message Exchange



Messages

Message Encryption & Authentication

The role of public/private keys

Message Encryption

Since anyone should be able to encrypt messages for a specific recipient, the public key is used to encrypt messages. The encrypted message can then be decrypted only by the corresponding private key that the recipient of that message has.

Message Authentication

Message authentication is used both to confirm the origin of the message (who wrote it), and to make sure the message has not been tempered with at any point during transit.

Since anyone should be able to verify signatures, the public key is used to verify signatures and the private key is used to sign messages.

In which order to perform the ops

While any order might look secure since the message would end up encrypted and authenticated anyway, it has been proven that the encrypt, then authenticate method is unequivocally the best option, which is why this application uses it. See the papers below for more details.

Hugo Krawczyk. The order of encryption and authentication for protecting communications (or: how secure is SSL?). 2001.

https://www.iacr.org/archive/crypto2001/21390309.pdf

Mihir Bellare and Chanathip Namprempre. Authenticated encryption: relations among notions and analysis of the generic composition
paradigm. 2007.

https://eprint.iacr.org/2000/025.pdf

Messages

Messaging Modes

Normal vs Ghost Mode

Feature	Normal Mode	Ghost Mode
Real-time delivery	✓	✓
Permanently Stored	✓	✗
Notifications	✓	✗
Message Reactions	✓	✓
Message Actions	✓	✓
Ghostly Look	✗	✓

How does Ghost Mode work

The Ring Relay uses the Firebase Realtime DB to establish real-time connections between users. Since this works only if both users are active in the conversation, ghostly messages work only then. If one of the users doesn't have the conversation opened and the other user sends a ghostly message, that message can't be displayed to the intended user due to the nature of how this system works. In this case, this is a feature, not a bug, since the Ghost Mode is intended to support stealthy conversations.

Since the Message Buffer resets every 3 messages, ghostly conversations are stored locally for both users and as soon as the users exits the conversation, reopens the app, or exits Ghost Mode, the conversation between them will disappear forever.

Private Keys Management

What is this section answering

- How to manage your private keys / Migrate Account
- When to regenerate your key pairs

How to manage your private keys

If you want to migrate your account to another device, you can export your private keys using one of the following methods:

- QR Code Scan

This mode generates a series of QR Codes you can scan directly from the device you're trying to transfer your private keys(identity) to.

Adjust the screen brightness (up or down) and how far away the screen is and wait until the blue indicator at the bottom turns green.

- Offline Text File Transfer

This mode generates a text file containing your encrypted private keys.

This file can be imported at any time later on in case anything happens to your device. Its important to remember your account password at the time you created your backup since if you changed your password afterwards at any point, the import will fail since your account password also acts as the encryption/decryption key for your private keys.

You should never transfer the file containing your encrypted private keys over the internet or store them in the cloud. Always use offline transfer methods such as using a cable or bluetooth.

Its highly recommended you enable the collection of Security Logs so any actions involving your private keys would be logged and displayed in the ‘Logs’ section of the ‘Settings’ window.

Private Keys Management

When to regenerate your key pairs

The effects of regenerating your key pairs

Regenerating your key pairs means locally creating new keys, then overriding your old private keys with the new ones, and finally, sending your new matching public keys to the server.

Since your private keys are required to decrypt any conversation you had, all conversations will become unreadable and the security signatures for it will mismatch since they were based on your old public keys.

The security signatures have their own section but the TL;DR of it is your contacts will be prompted when entering the conversation with you that these signatures are mismatching. They will have to option to trust the new signatures, but only if they know you were the one who regenerated your key pairs. This is why its important to let your contacts know you're about to perform this action.

The mechanism described above is necessary because if a bad actor would've somehow gained access to your account on a device that doesn't have your private keys, that bad actor would have to first regenerate your key pairs in order for them to send or read any messages meant for you. Your old conversations would be safe anyway since the regen makes them unreadable for everyone.

When to regenerate your key pairs

Regenerating your key pairs should be avoided as much as possible to avoid the effects described above. However, regenerating your key pairs is the only option you have if you somehow lost your private keys.

Remember to let your contacts know through any channel that you were the person who performed that action.

Security Signatures

What is this section answering

- How do Security Signatures work
- What purpose do they have

How do Security Signatures work

<Signature Name>

<SIG Status> <Signature Tag>

<Signature Barcode>

Signature Components

- Signature Name

The signature name indicates what type of signature is being presented.

- Signature Status

This status indicates whether the current signature matches the one the user had at the start of the conversation. These are all possible states for the status:

SIG Fail	The current signature failed to match the original signature at the start of the conversation
Verified	The signatures matched the way they should have
Self	There is no original signature to check against
Invalid	The signature is corrupted
UNKNOWN	The signature hasn't been fetched yet or the SIG fetch failed

Security Signatures

How do Security Signatures work

<Signature Name>

<SIG Status> <Signature Tag>

<Signature Barcode>

Signature Components

- ▣ Signature Tag

  This is a part of the signature

- ▣ Signature Barcode

  This is the barcode for the signature tag. While that barcode contains the Signature Tag, it can't probably be scanned since it's possible its not fully displayed to prevent overflow. This is purely decorative and doesn't have an actual purpose other than visually representing each SIG Tag

How do Security Signatures work

Conversation Signature

SIG Fail vxdL+yndz

AND

Remote SIG Verification Key

Verified ALBc+AFs_

+

Remote Encryption Key

SIG Fail vxNL+GxOE

The conversation signature is the result of the logic operation 'and' (that only returns true if both other signatures are true).

It works this way because the Conversation Signature itself is composed of the parts of both other two.

Security Signatures

What purpose do Security Signatures Have

Security Signatures prevent any bad actors from impersonating any user since after the key pairs have been regenerated, the signatures will fail. If any security signature fails for any reason, the user will have to manually retrust the new signatures, but they should do so only if their contact confirmed they were responsible for the keys regen. As an additional security measure, the user who caused the signatures to mismatch can't trust any new signatures even if they're also prompted with the dialog to do so.

Database Architecture

What is this section answering

- ☐ How is data stored
- ☐ What data is stored and why

Database Schema for User Accounts

UID	[String] User ID
Username	[String]
Password	[String] Hashed password
Email	[String]
MFA_TOKEN	[String|JSON] Multi-factor Auth Token (not yet in use)
Public Key	[String|JWK] Public Key used for message encryption
Public Signing Key	[String|JWK] Public Key used for verifying message signatures

Database Architecture

Database Schema for User Accounts

TX	[Int] The Unix time when the account was created
Notifications Config	[String|JSON] Object containing notification preferences
Logs Config	[String|JSON] Object containing log collection preferences

Database Schema for Messages (MSUID Table)

liked	[Boolean] Is message liked?
tx	[Int] Message transmission time (unix)
seen	[Boolean] Has the message been seen?
ownContent	[String] Encrypted message content for self
remoteContent	[String] Encrypted message content for recipient
originUID	[String] User ID of the sender
targetUID	[String] User ID of the recipient
signature	[String] Encrypted Message Signature
MID	[String] Message ID

Database Schema for (Contacts) References

Own UID	[String] Own User ID
Foreign UID	[String] Remote/Foreign User ID
Status	[String] Contact Request status (Pending/Approved)
MSUID	[String] Conversation Storage Table ID

Database Architecture

Database Schema for (Contacts) References

PKSH	[String] Encryption Keys Signature at conversation start
SPKSH	[String] Signing Keys Signature at conversation start
TX	[Int] Unix timestamp when the Contact Request was first sent
Last TX	[Int] Unix timestamp when the Contact Request Status got approved

Database Schema for Logs

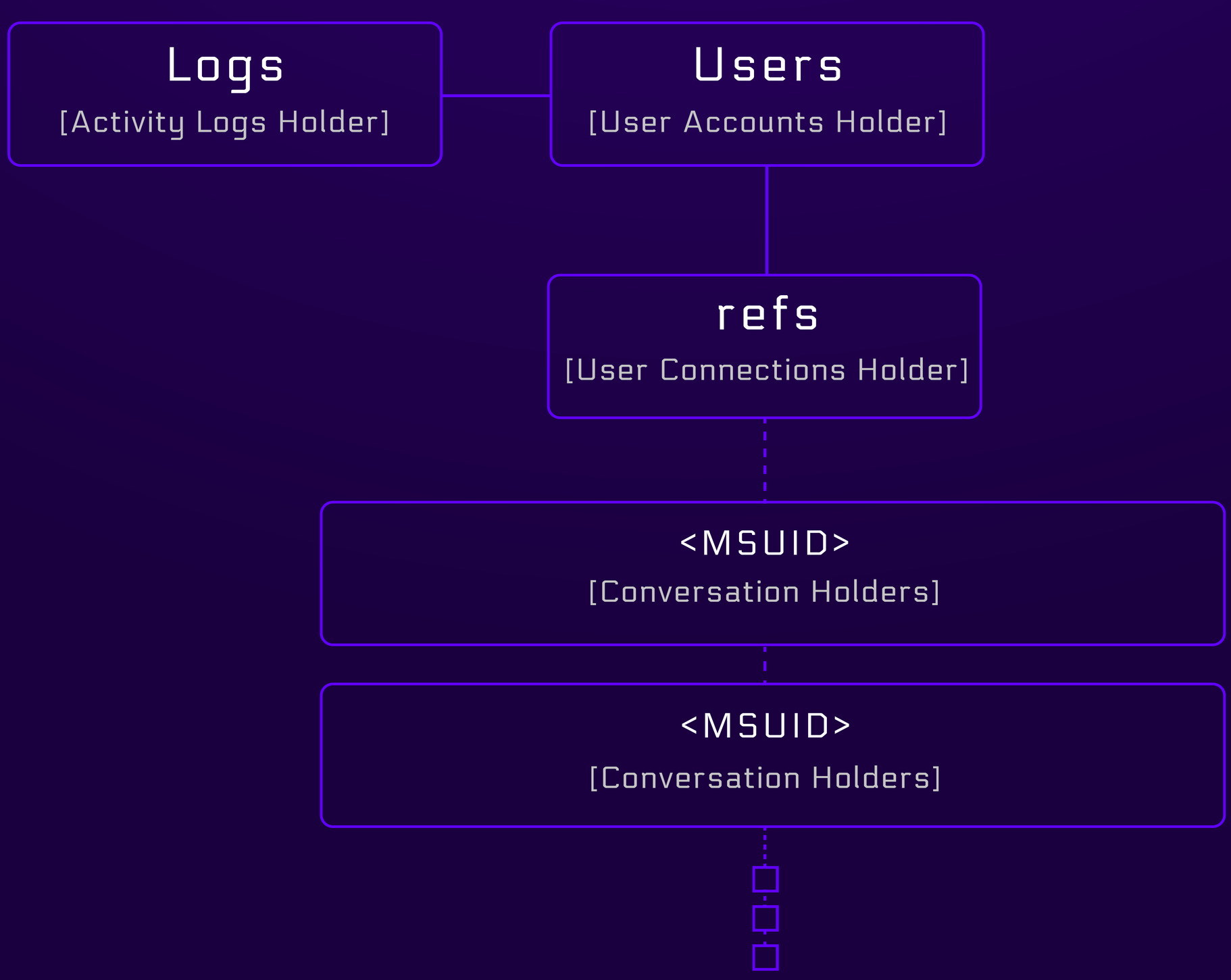
TX	[String] Unix timestamp of when the log was created
UID	[String] The User ID that the log is related to
Severity	[String] Severity Level depending on the event type
Type	[String] Primary Event Type (Account/Security)
Subtype	[String] Detailed Event Type
IP	[String] IP Address of the device triggering the log
(IP) Location	[String|JSON] (Aprx) Location based on the IP Address above
Details	[String] Log specific details

How are those tables linked

The References Table (refs) is the most important table from a logical perspective. This table holds the connections between users as well as some additional information that allows conversations to work properly (Security Signatures, the MSUID).

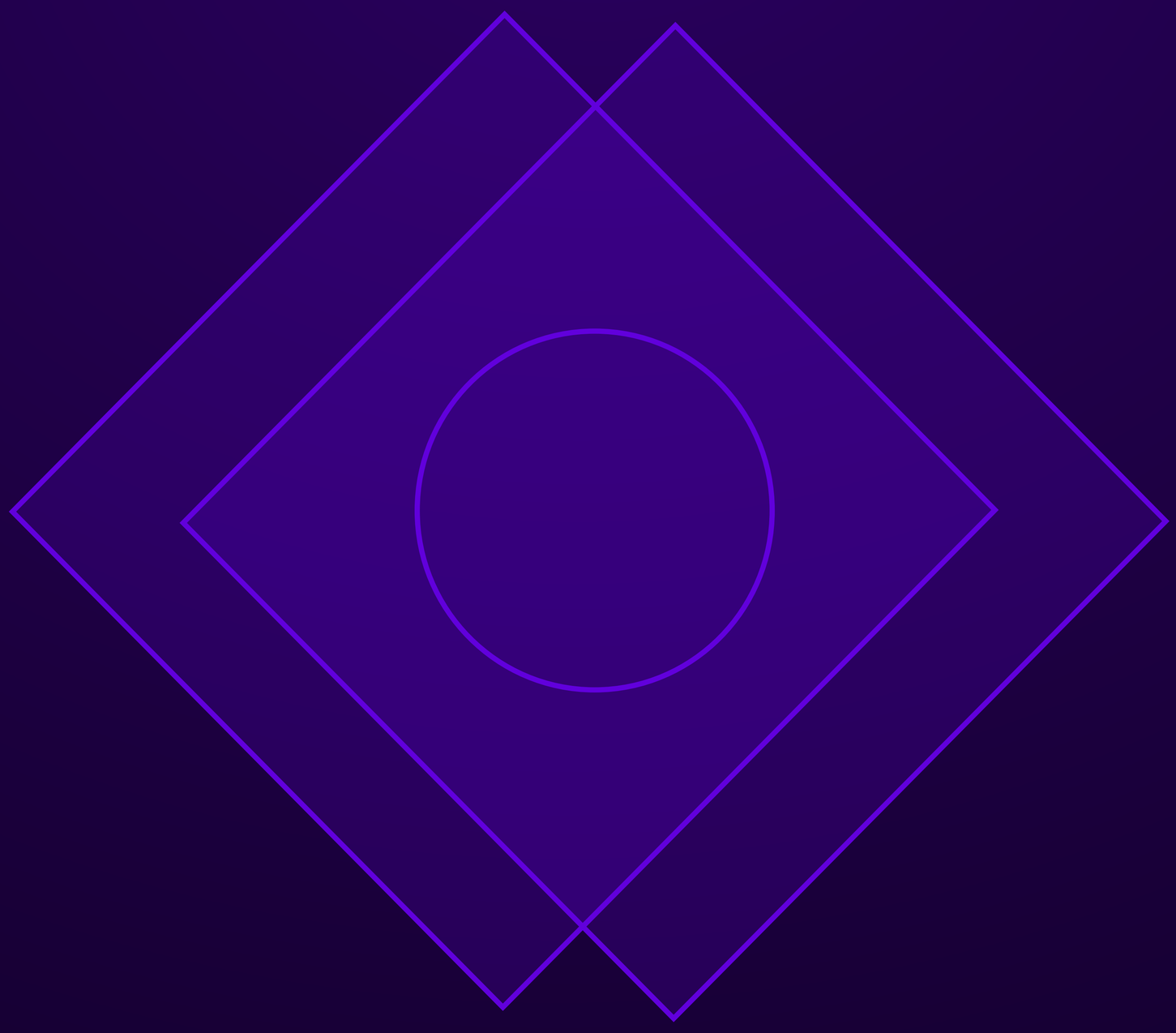
Database Architecture

How are those tables linked



Why this architecture

This architecture was designed with scalability in mind. By far, the biggest volume of data comes from all the conversations between users. To avoid having excessively large tables, every user has their own MSUID table that can be used to store their conversations with other people. The conversation between two people is stored in the MSUID table of the user who made the connection request. This architecture is also very straight-forward and requires very little maintenance which makes it ideal in this context. There are improvements that can be made in the future in the case of the current system under-performing for a way bigger number of users using this app. These improvements could be easily implementable and rolled out, but for the foreseeable future, this system can handle the loads required.



RING RELAY

Technical Documentation

[End Of File]