**Architecture Design**

1. **Telegram Applications**

**Open source** and support **reproducible builds.** This means that anyone can independently verify that our [code on GitHub](https://telegram.org/apps#source-code) is the exact same code that was used to build the apps you download from App Store or Google Play.

1. **Telegram APIS**

We offer two kinds of APIs for developers. The [**Bot API**](https://core.telegram.org/api#bot-api) allows you to easily create programs that use Telegram messages for an interface. The [**Telegram API and TDLib**](https://core.telegram.org/api#tdlib-build-your-own-telegram) allow you to build your own customized Telegram clients.

1. **Bot API**

This API allows you to connect bots to our system. [**Telegram Bots**](https://core.telegram.org/bots) are special accounts that do not require an additional phone number to set up. These accounts serve as an interface for code running somewhere on your server.

To use this, you don't need to know anything about how our MTProto encryption protocol works — our intermediary server will handle all encryption and communication with the Telegram API for you. You communicate with this server via a simple HTTPS-interface that offers a simplified version of the Telegram API.

1. **Telegram API**

This API allows you to build your own customized Telegram clients. It is 100% open for all developers who wish to create Telegram applications on our platform. Feel free to study the open [source code](https://telegram.org/apps#source-code) of existing Telegram applications for examples of how things work here.

1. **TDLib**

Even if you're looking for maximum customization, you don't have to create your app from scratch. Try our [**Telegram Database Library**](https://core.telegram.org/tdlib) (or simply TDLib), a tool for third-party developers that makes it easy to build fast, secure and feature-rich Telegram apps.

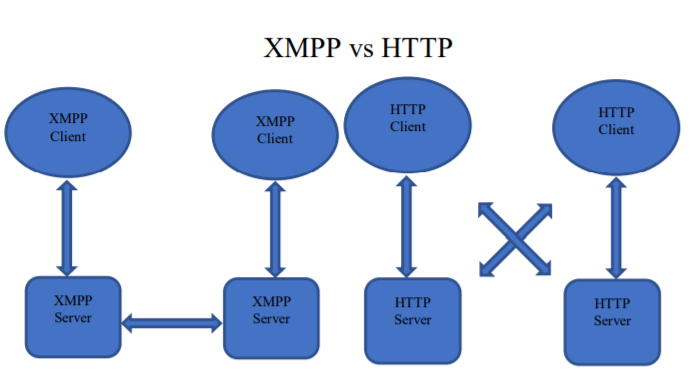
TDLib takes care of all **network implementation** details, **encryption** and **local data storage**, so that you can dedicate more time to design, responsive interfaces and beautiful animations.

TDLib supports all Telegram features and makes developing Telegram apps a breeze on any platform. It can be used on Android, iOS, Windows, macOS, Linux and virtually any other system. The library is open source and compatible with virtually **any programming language**.

1. **Server**

XMPP server (XMPP-Extensible Messaging and Presence Protocol) server used by the Telegram. To transfer message between the open source Ejabberd has been used, it transfer messages between two or more user at any point of time.

Telegram make use of both HTTP and XMPP for the communication. XMPP used during asynchronous federated limited purpose communication whereas HTTP used synchronous unfedered general purpose communication



Telegram's server-side software is closed-source and proprietary.

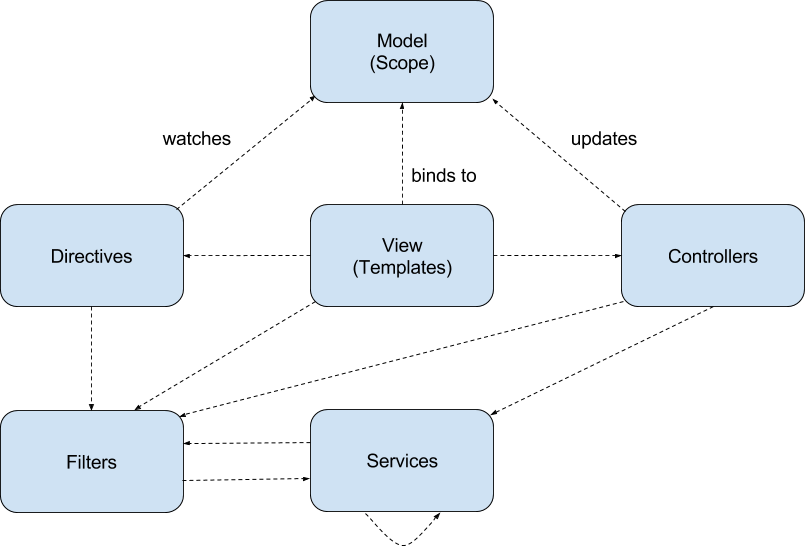
1. **Design model**

Telegram-Web is designed as a [Single Page Application](https://en.wikipedia.org/wiki/Single-page_application) and uses the AngularJS 1 framework. AngularJS was specifically developed to make building single page web applications easy, which is probably why it was used for Telegram-Web. It's based on the Model-View-Controller design pattern, which structures an application in three parts:

* **model:** the data (also called state) of the application;
* **view:** the interface of the application, which shows the data in the model. The view is defined in HTML-based Template files;
* **controller:** component that updates the model when the user makes a change to the view. This is the place where business logic such as validation of the user input happens.

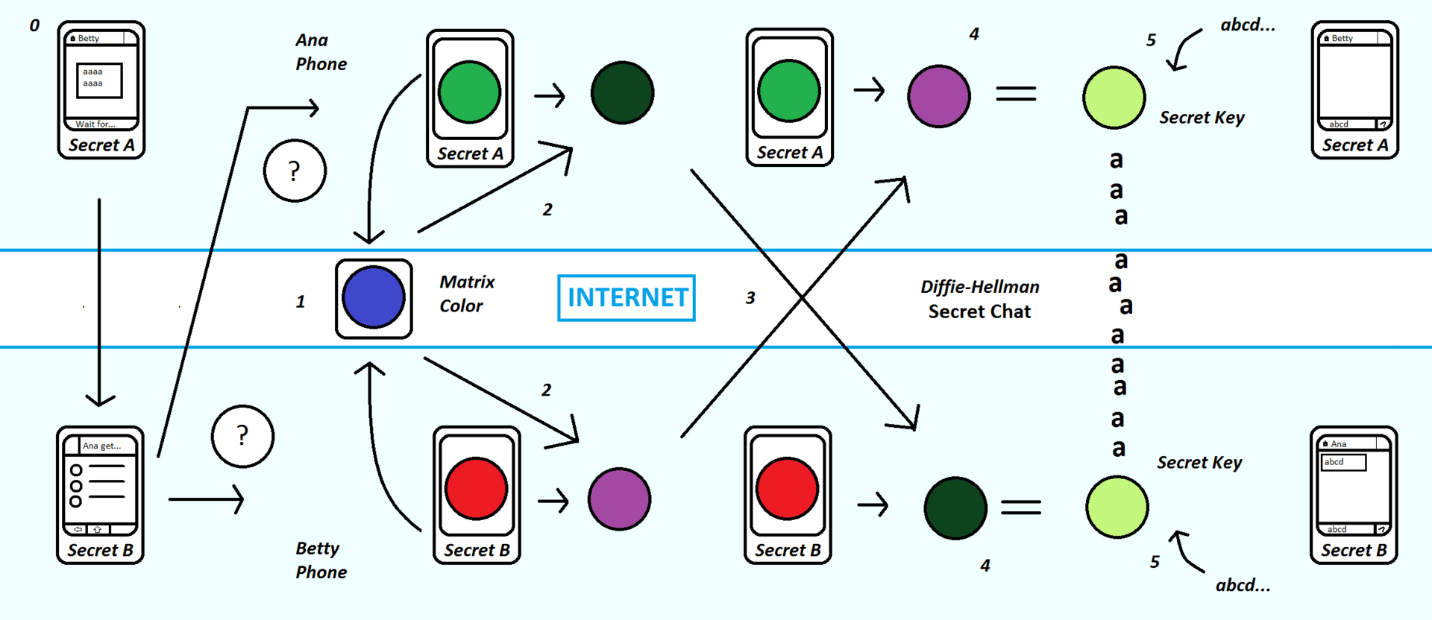
There are some additional types of components in AngularJS:

* **directives:** updates part of the view when the model changes;
* **filters:** functions that format data for display to the user;
* **services:** reusable business logic that is independent of views.



1. **Protocol**
2. **Encryption scheme**

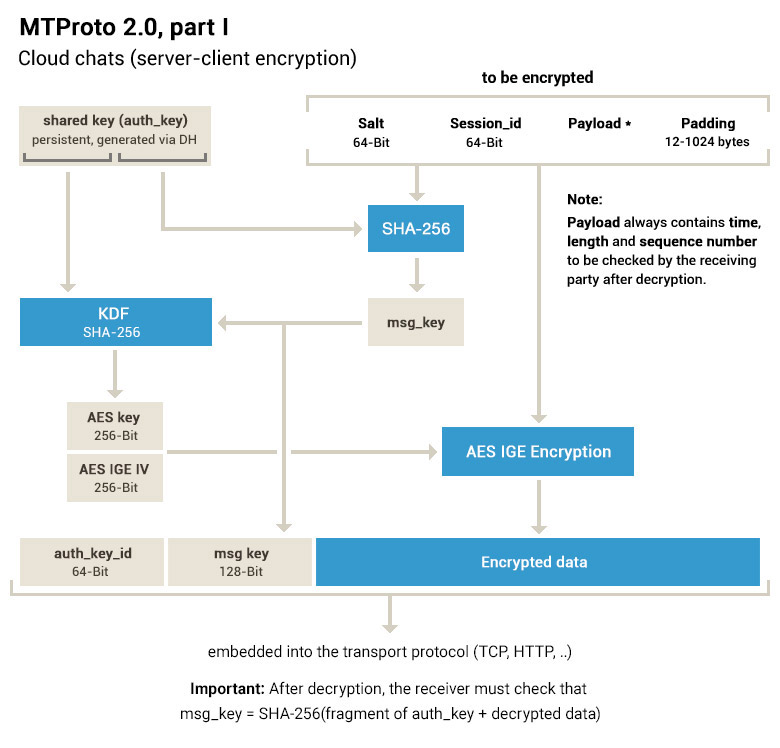
Telegram uses a [symmetric encryption](https://en.wikipedia.org/wiki/Symmetric_encryption) scheme called MTProto



The protocol is designed for access to a server API from applications running on mobile devices. It must be emphasized that a web browser is not such an application.

The protocol is subdivided into three virtually independent components:

* High-level component (API query language): defines the method whereby API queries and responses are converted to binary *messages*.
* Cryptographic (authorization) layer: defines the method by which messages are encrypted prior to being transmitted through the transport protocol.
* Transport component: defines the method for the client and the server to transmit messages over some other existing network protocol (such as HTTP, HTTPS, WS (plain websockets), WSS (websockets over HTTPS), TCP, UDP).



1. **Telegram OSI Model:**

* Layer 7 (Application): [High-level RPC API](https://core.telegram.org/mtproto#high-level-component-rpc-query-languageapi)
* Layer 6 (Presentation): [Type Language](https://core.telegram.org/mtproto/TL)
* Layer 5 (Session): [MTProto session](https://core.telegram.org/mtproto#high-level-component-rpc-query-languageapi)
* Layer 4 (Transport):
  + 4.3: [MTProto transport protocol](https://core.telegram.org/mtproto#mtproto-transport)
  + 4.2: [MTProto obfuscation (optional)](https://core.telegram.org/mtproto/mtproto-transports#transport-obfuscation)
  + 4.1: [Transport protocol](https://core.telegram.org/mtproto#transport)
* Layer 3 (Network): IP
* Layer 2 (Data link): MAC/LLC
* Layer 1 (Physical): IEEE 802.3, IEEE 802.11, etc...
  1. **High-Level Component (RPC Query Language/API)**

From the standpoint of the high-level component, the client and the server exchange messages inside a session. The session is attached to the client device (the application, to be more exact) rather than a specific websocket/http/https/tcp connection. In addition, each session is attached to a user key ID by which authorization is actually accomplished.

Several connections to a server may be open; messages may be sent in either direction through any of the connections (a response to a query is not necessarily returned through the same connection that carried the original query, although most often, that is the case; however, in no case can a message be returned through a connection belonging to a different session). When the UDP protocol is used, a response might be returned by a different IP address than the one to which the query had been sent.

There are several types of messages:

* RPC calls (client to server): calls to API methods
* RPC responses (server to client): results of RPC calls
* Message received acknowledgment (or rather, notification of status of a set of messages)
* Message status query
* Multipart message or container (a container that holds several messages; needed to send several RPC calls at once over an HTTP connection, for example; also, a container may support gzip).

From the standpoint of lower level protocols, a message is a binary data stream aligned along a 4 or 16-byte boundary. The first several fields in the message are fixed and are used by the cryptographic/authorization system.

Each message, either individual or inside a container, consists of a message identifier (64 bits), a message sequence number within a session (32 bits), the length (of the message body in bytes; 32 bits), and a body (any size which is a multiple of 4 bytes). In addition, when a container or a single message is sent, an internal header is added at the top (see below), then the entire message is encrypted, and an external header is placed at the top of the message (a 64-bit key identifier and a 128-bit message key).

A message body normally consists of a 32-bit message type followed by type-dependent parameters. In particular, each RPC function has a corresponding message type. For more detail, see [Binary Data Serialization](https://core.telegram.org/mtproto/serialize), [Mobile Protocol: Service Messages](https://core.telegram.org/mtproto/service_messages).

1. **Transport**

Enables the delivery of encrypted containers together with the external header (hereinafter, Payload) from client to server and back. Multiple transport protocols are defined:

* [TCP](https://core.telegram.org/mtproto/transports#tcp)
* [Websocket](https://core.telegram.org/mtproto/transports#websocket)
* [Websocket over HTTPS](https://core.telegram.org/mtproto/transports#websocket-over-https)
* [HTTP](https://core.telegram.org/mtproto/transports#http)
* [HTTPS](https://core.telegram.org/mtproto/transports#https)
* UDP

1. **MTProto transport**

Before being sent using the selected transport protocol, the payload has to be wrapped in a secondary protocol header, defined by the appropriate MTProto transport protocol.

* [Abridged](https://core.telegram.org/mtproto/mtproto-transports#abridged)
* [Intermediate](https://core.telegram.org/mtproto/mtproto-transports#intermediate)
* [Padded intermediate](https://core.telegram.org/mtproto/mtproto-transports#padded-intermediate)
* [Full](https://core.telegram.org/mtproto/mtproto-transports#full)

The server recognizes these different protocols (and distinguishes them from HTTP, too) by the header. Additionally, the following transport features can be used:

* [Quick ack](https://core.telegram.org/mtproto/mtproto-transports#quick-ack)
* [Transport errors](https://core.telegram.org/mtproto/mtproto-transports#transport-errors)
* [Transport obfuscation](https://core.telegram.org/mtproto/mtproto-transports#transport-obfuscation)

Example implementations for these protocols can be seen in [tdlib](https://github.com/tdlib/td/blob/master/td/mtproto/TcpTransport.cpp) and [MadelineProto](https://github.com/danog/MadelineProto/tree/master/src/danog/MadelineProto/Stream/MTProtoTransport).