**Department of Science and Technology**

CST2560

**Internet Scale Application**

Coursework 3 – Telegram Application

Name of Student: Amal Nazeer

MISIS: M00691054

Module Tutor : Mr.Jaspreet Sethi

Year : 2020-21

# Abstract

An indept study of the world’s largest hybrid public and private messenger, Telegram describing System Architecture. Protocols used and Databases with an overview of the privacy,security and legal issues pertaining to it.   
As a popuar IM(Instant Messaging) Application,the open based program has reached upto 500 million users as of february 2021.The clean and responsive interface, designed to be light-weight ,reliable and is now available in multiple platforms and is amply famous for its emphasis on privacy, especially that of of phone numbers are some of the most popular features of this relatively new yet potential rival of established giants such as WhatsApp and Facebook.   
For the purpose of an indepth understanding, aspects such as System Architecture,and the used Communication protocols and the symmetric encryption scheme, MTProto Mobile Protocol, service APIs, CDNs and the source code examination is done.

Some finding of the study includes inference on how the system is achieved ……. With reference to …….

# Introduction

The following report describes Briefly describe what the report is about (do not describe the coursework task or list the requirements). Include a paragraph at the end of the introduction describing the layout of the rest of the report.  
  
The layout of the report is structured such that the major sections – System Description, Architecture and Protocols and Security and Legal Issues are followed by the conclusion and findings and references.

**Telegram** is founded by Russian entrepreneur Pavel Durov and was launched for iOS on 14 August 2013 and Android in October 2013.

Authorization Key (auth\_key)

A 2048-bit key shared by the client device and the server, created upon user registration directly on the client device by exchanging Diffie-Hellman keys, and never transmitted over a network. Each authorization key is user-specific. There is nothing that prevents a user from having several keys (that correspond to “permanent sessions” on different devices), and some of these may be locked forever in the event the device is lost. See also [Creating an Authorization Key](https://core.telegram.org/mtproto/auth_key).

Server Key

A 2048-bit RSA key used by the server digitally to sign its own messages while registration is underway and the authorization key is being generated. The application has a built-in public server key which can be used to verify a signature but cannot be used to sign messages. A private server key is stored on the server and changed very infrequently.

Key Identifier (auth\_key\_id)

The 64 lower-order bits of the SHA1 hash of the authorization key are used to indicate which particular key was used to encrypt a message. Keys must be uniquely defined by the 64 lower-order bits of their SHA1, and in the event of a collision, an authorization key is regenerated. A zero key identifier means that encryption is not used which is permissible for a limited set of message types used during registration to generate an authorization key in a Diffie-Hellman exchange. **For MTProto 2.0, SHA1 is still used here, because auth\_key\_id should identify the authorization key used independently of the protocol version.**

Session

A (random) 64-bit number generated by the client to distinguish between individual sessions (for example, between different instances of the application, created with the same authorization key). The session in conjunction with the key identifier corresponds to an application instance. The server can maintain session state. Under no circumstances can a message meant for one session be sent into a different session. The server may unilaterally forget any client sessions; clients should be able to handle this.

Server Salt

A (random) 64-bit number periodically (say, every 24 hours) changed (separately for each session) at the request of the server. All subsequent messages must contain the new salt (although, messages with the old salt are still accepted for a further 300 seconds). Required to protect against replay attacks and certain tricks associated with adjusting the client clock to a moment in the distant future.

Message Identifier (msg\_id)

A (time-dependent) 64-bit number used uniquely to identify a message within a session. Client message identifiers are divisible by 4, server message identifiers modulo 4 yield 1 if the message is a response to a client message, and 3 otherwise. Client message identifiers must increase monotonically (within a single session), the same as server message identifiers, and must approximately equal unixtime\*2^32. This way, a message identifier points to the approximate moment in time the message was created. A message is rejected over 300 seconds after it is created or 30 seconds before it is created (this is needed to protect from replay attacks). In this situation, it must be re-sent with a different identifier (or placed in a container with a higher identifier). The identifier of a message container must be strictly greater than those of its nested messages.

**Important**: to counter replay-attacks the lower 32 bits of **msg\_id** passed by the client must not be empty and must present a fractional part of the time point when the message was created.

Content-related Message

A message requiring an explicit acknowledgment. These include all the user and many service messages, virtually all with the exception of containers and acknowledgments.

Message Sequence Number (msg\_seqno)

A 32-bit number equal to twice the number of “content-related” messages (those requiring acknowledgment, and in particular those that are not containers) created by the sender prior to this message and subsequently incremented by one if the current message is a content-related message. A container is always generated after its entire contents; therefore, its sequence number is greater than or equal to the sequence numbers of the messages contained in it.

Message Key (msg\_key)

In **MTProto 2.0**, the middle 128 bits of the SHA-256 hash of the message to be encrypted (including the internal header and the padding bytes for MTProto 2.0), prepended by a 32-byte fragment of the authorization key.

In **MTProto 1.0**, message key was defined differently, as the lower 128 bits of the SHA-1 hash of the message to be encrypted, with padding bytes excluded from the computation of the hash. Authorization key was not involved in this computation.

Internal (cryptographic) Header

A header (16 bytes) added before a message or a container before it is all encrypted together. Consists of the server salt (64 bits) and the session (64 bits).

External (cryptographic) Header

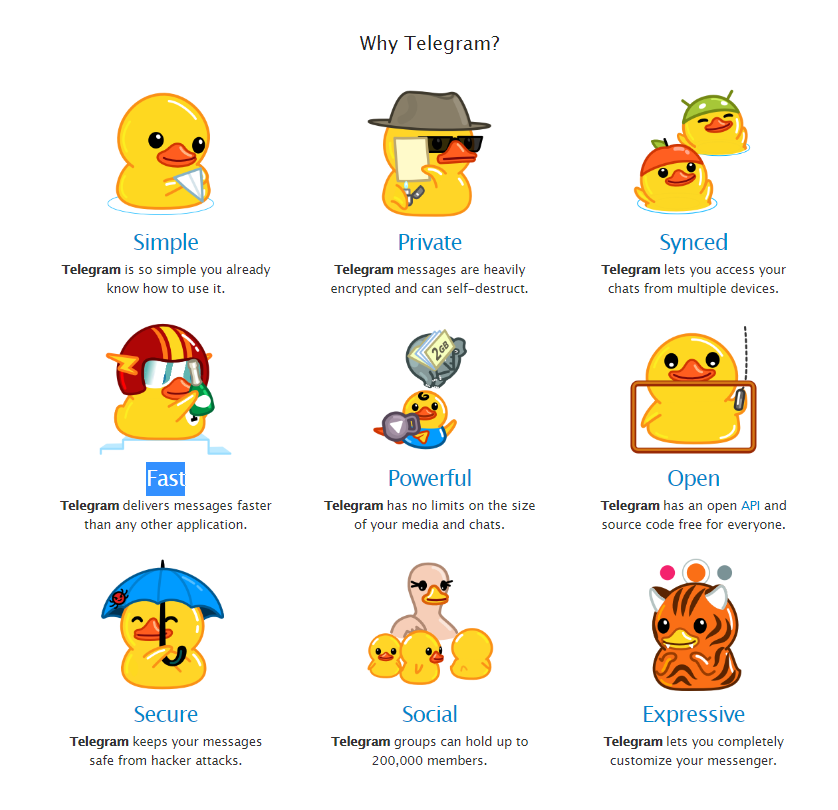
A header (24 bytes) added before an encrypted message or a container. Consists of the key identifier **auth\_key\_id** (64 bits) and the message key **msg\_key** (128 bits).

Payload

External header + encrypted message or container.

# System description

# The following are the features of telegram high;ighted in the official website:



Simple :

* Easy to use.

Private

* Encryption

Synched

How is it achieved

Speed

Powerful

Open

Secure

Social

Expressive

# Architecture and protocols

According the to creators of telegram , Telegram is a *“Telegram is a cloud-based mobile and desktop messaging app with a focus on security and speed*.”

| [**Protocol**](https://en.wikipedia.org/wiki/Communications_protocol) | **Creator** | **First public release date** | **License** | **Identity (not inc. alias)** | **Asynchronous message relaying** | [**Transport Layer Security**](https://en.wikipedia.org/wiki/Transport_Layer_Security) | **Webcam/Video** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **MTProto (**[**Telegram**](https://en.wikipedia.org/wiki/Telegram_(software))**)** | **Telegram Messenger LLP** | **2013 Aug** | [**Open standard**](https://en.wikipedia.org/wiki/Open_standard) | **Phone number (e.g. +1234567890), nickname (e.g. @example)** | **Yes** | **Yes** | **Yes** |

| [**End-to-end encryption**](https://en.wikipedia.org/wiki/End-to-end_encryption) | **Unlimited number of contacts** | **Bulletins to all contacts** | **One-to-many routing**[4](https://en.wikipedia.org/wiki/Comparison_of_instant_messaging_protocols#fn_4) | [Spam](https://en.wikipedia.org/wiki/Messaging_spam)**protection** | **Group, channel or conference support** | **Audio/VoIP support** |
| --- | --- | --- | --- | --- | --- | --- |
| **No end-to-end encryption for group chats on any client. No end-to-end encrypted one-on-one chats on desktop clients** | **Yes** | **No** | **Yes** | **Yes, contact blocking** | **Yes** | **Yes**[[3]](https://en.wikipedia.org/wiki/Comparison_of_instant_messaging_protocols#cite_note-3) |

| **Batch file sharing** | **Media synchronisation** | **Serverless**[6](https://en.wikipedia.org/wiki/Comparison_of_instant_messaging_protocols#fn_6)**(decentralized)** | [Protocol](https://en.wikipedia.org/wiki/Communications_protocol) |
| --- | --- | --- | --- |
| **Yes** | **Yes** | **No** | **MTProto (**[Telegram](https://en.wikipedia.org/wiki/Telegram_(software))**)** |

Most Depending on the IM protocol, the technical architecture can be [peer-to-peer](https://en.wikipedia.org/wiki/Peer-to-peer) (direct [point-to-point](https://en.wikipedia.org/wiki/Point-to-point_(telecommunications)) transmission) or [client–server](https://en.wikipedia.org/wiki/Client%E2%80%93server_model) (an [Instant message service center](https://en.wikipedia.org/wiki/Instant_message_service_center) retransmits messages from the sender to the communication device).

The telegram Apps are **open source** and support **reproducible builds**.

#### Telegram Database Library (TDLib)

* [TDLib](https://core.telegram.org/tdlib) – a cross-platform client designed to facilitate creating custom apps on the Telegram platform.
* 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**.
* [Telegram X for Android](https://play.google.com/store/apps/details?id=org.thunderdog.challegram&hl=en) – a slick experimental Telegram client based on TDLib.

# Security, privacy and legal issues

# Mobile Protocol: Detailed Description

### Storing an Authorization Key on a Client Device

It may be suggested to users concerned with security that they password protect the authorization key in approximately the same way as in ssh. This can be accomplished by prepending the value of cryptographic hash function, such as SHA-256, of the key to the front of the key, following which the entire string is encrypted using AES in CBC mode and a key equal to the user’s (text) password. When the user inputs the password, the stored protected password is decrypted and verified by checking the SHA-256 value. From the user’s standpoint, this is practically the same as using an application or a website password.

### Unencrypted Messages

Special plain-text messages may be used to create an authorization key as well as to perform a time synchronization. They begin with auth\_key\_id = 0 (64 bits) which means that there is no auth\_key. This is followed directly by the message body in serialized format without internal or external headers. A message identifier (64 bits) and body length in bytes (32 bytes) are added before the message body.

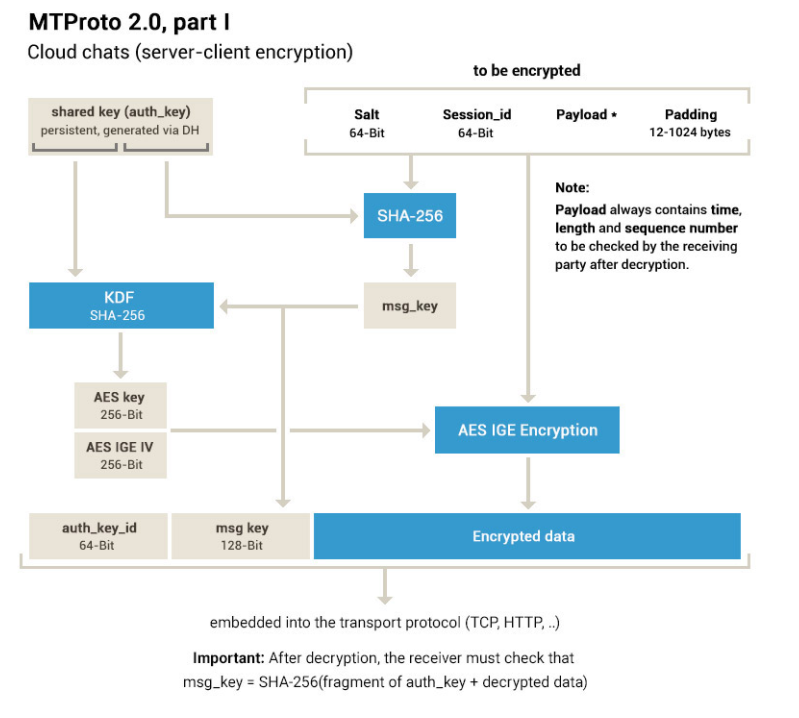
Only a very limited number of messages of special types can be transmitted as plain text.

### Creating an Authorization Key

An authorization key is normally created once for every user during the application installation process immediately prior to registration. Registration itself, in actuality, occurs after the authorization key is created. However, a user may be prompted to complete the registration form while the authorization key is being generated in the background. Intervals between user key strokes may be used as a source of entropy in the generation of high-quality random numbers required for the creation of an authorization key.

See [Creating an Authorization Key](https://core.telegram.org/mtproto/auth_key).

During the creation of the authorization key, the client obtains its server salt (to be used with the new key for all communication in the near future). The client then creates an encrypted session using the newly generated key, and subsequent communication occurs within that session (including the transmission of the user's registration information and phone number validation) unless the client creates a new session. The client is free to create new or additional sessions at any time by choosing a new random session\_id.



# Conclusion

# References

[1]Calvo, D. et al. (2017) ‘Multiplatform Career Guidance System Using IBM Watson, Google Home and Telegram: A User Experience and Usability Evaluation’, in Ubiquitous Computing and Ambient Intelligence. Cham: Springer International Publishing, pp. 689–700. doi: 10.1007/978-3-319-67585-5\_67.

[2] Dargahi Nobari, A. et al. (2021) ‘Characteristics of viral messages on Telegram; The world’s largest hybrid public and private messenger’, Expert systems with applications, 168, p. 114303–. doi: 10.1016/j.eswa.2020.114303.