telegram app api download github

Telegram Applications.

Our apps are open source and support reproducible builds . This means that anyone can independently verify that our code on GitHub is the exact same code that was used to build the apps you download from App Store or Google Play. Developers are welcome to check out our Guide to Reproducible Builds for iOS and Android.

Mobile apps.

You can also download the latest version of Telegram for Android from this channel.

Desktop apps.

Web apps.

Telegram Database Library (TDLib)

– a cross-platform client designed to facilitate creating custom apps on the Telegram platform. – a slick experimental Telegram client based on TDLib.

Unofficial apps.

(based on TDLib) (desktop and Xbox One)

Source code.

For the moment we are focusing on open sourcing the things that allow developers to quickly build something using our API. We have published the code for our Android, iOS, web and desktop apps (Win, macOS and Linux) as well as the Telegram Database Library.

This code allows security researchers to fully evaluate our end-to-end encryption implementation. It is also possible to independently verify that Telegram apps available on Google Play and App Store are built using the same code that we publish on GitHub.

Telegram Database Library.

Cross-platform library for building custom Telegram apps, see TDLib for details. Licensed under Boost 1.0. GitHub »

Android SDK.

The Android SDK helps you easily integrate Telegram Passport requests into your Android-based apps. Check out our GitHub repository to see samples using this SDK.

Installation.

Installing from Maven.

Telegram Passport SDK is available from the Maven repository. Add this line to the dependencies section in your build.gradle:

and sync your project.

Adding as a module.

Download the library, unzip it and copy the library project to the root of your project directory (the one with settings.gradle and gradle.properties). Then, make the following changes to your Gradle scripts.

In settings.gradle, add ':telegrampassport' to includes:

In the build.gradle file for your app, add this line to the dependencies section:

and sync your project.

Usage.

Adding the button.

The SDK provides the "Log in with Telegram" button which we recommend using for a consistent user experience across different apps. You can either add it from your Java code:

Requesting authorization.

The button doesn't do anything by itself; you need to set an OnClickListener on it to start the authorization flow (replace the comments with actual parameters):

If you need more control over the process, the TelegramPassport class contains several more methods:

Telegram Database Library.

TDLib (Telegram Database Library) is a cross-platform, fully functional Telegram client. We designed it to help third-party developers create their own custom apps using the Telegram platform.

TDLib Advantages.

Cross-platform . TDLib can be used on Android, iOS, Windows, macOS, Linux, WebAssembly, FreeBSD, Windows Phone, watchOS, tvOS, Tizen, Cygwin. It should also work on other *nix systems with or without minimal effort. Multilanguage . TDLib can be easily used with any programming language that is able to execute C functions. Additionally it already has native bindings to Java (using JNI) and C# (using C++/CLI). Easy to use . TDLib takes care of all network implementation details , encryption and local data storage . High-performance . In the Telegram Bot API, each TDLib instance handles more than 24,000 active bots simultaneously. Well-documented . All TDLib API methods and public interfaces are fully documented . Consistent . TDLib guarantees that all updates will be delivered in the right order . Reliable . TDLib remains stable on slow and unreliable Internet connections. Secure : All local data is encrypted using a user-provided encryption key. Fully-asynchronous . Requests to TDLib don't block each other or anything else, responses will be sent when they are available.

Resources.

TDLib is fully open source , all code is available on GitHub.

Reproducible Builds for iOS and Android.

This page contains instructions for verifying that Telegram's open source code is exactly the same as the code that is used to build the apps that are available in the App Store, Google Play and directly on the Telegram website.

Warning: Telegram supports reproducible builds as of version 5.13 . Bear in mind that, at this stage, the verification process should be considered experimental . We will be updating our apps and these instructions to make this process as straightforward as possible.

Please read the relevant notes and troubleshooting section carefully.

Reproducible Builds for Android.

Step 1. Install Docker.

Docker can be obtained here. Once the installation is complete, log into your Docker account > Preferences > Advanced and configure the amount of resources Docker may use:

We recommend using the maximum amount allowed by your system's hardware, in order to speed up the build time.

Step 2. Confirm which version you have installed on your Android device.

You can find the version/build number at the bottom of the Settings page. Note that Telegram supports reproducible builds starting with version 5.13 .

The commit tag to checkout source code for the example above will be release-5.13.0_1821 .

Please make sure that you're using the correct version and build number of the version you want to check (and not the one from this example).

The part after the version number will help you know in which folder to look for the correct APK when you've finished building the app (Step 4):

“Direct” after version number means that the APK will be inside the “afat/standalone” folder. “Universal” after version number means that the APK will be inside the “afat/release” folder. If you have Android Version 6.0 or greater, your APK folder will have the “_SDK23” suffix. “arm64-v8a” - folder name will start with “arm64”. “armeabi-v7” - folder name will start with “armv7”. “x86” - folder name will start with “x86”. “x86_64” - folder name will start with “x64”.

Step 3. Obtain the source code.

Open Terminal, run the commands: git clone https://github.com/DrKLO/Telegram.git $HOME/telegram-android cd $HOME/telegram-android git checkout release-

For our example, the command would be: git checkout release-5.13.0_1821.

Step 4. Build the app.

Open Terminal, run the commands: cd $HOME/telegram-android docker build -t telegram-build .

docker run --rm -v "$PWD":/home/source telegram-build.

These commands will produce 9 different APKs for different target SDK versions and CPU ABIs.

These APKs can be found in: $HOME/telegram-android/TMessagesProj/build/outputs/apk/

Use the folder name from Step 2 to find the correct folder that holds the same APK as installed on your device. For example, for non-universal Android 9.0 arm64-v8a, the path to your APK will be: $HOME/telegram-android/TMessagesProj/build/outputs/apk/arm64_SDK23/release/app.apk Copy this APK to the root source directory by running this command in Terminal: cp $HOME/telegram-android/TMessagesProj/build/outputs/apk/arm64_SDK23/release/app.apk $HOME/telegram-android/telegram_built.apk.

Step 5. The Telegram APK installed on your device.

You will need adb for this step.

If you downloaded your APK directly from Telegram's website , use the package name org.telegram.messenger.web in this step. To verify the Google Play APK , use org.telegram.messenger .

Connect your device to the computer, open Terminal, run the commands: adb shell pm path org.telegram.messenger.

The output will look something like this: package:/data/app/org.telegram.messenger-_zOSURFEx2GpHM8UDF_PVg==/base.apk By using this information, pull the APK from your device to $HOME/telegram-android using command: adb pull /data/app/org.telegram.messenger-_zOSURFEx2GpHM8UDF_PVg==/base.apk $HOME/telegram-android/telegram_store.apk.

Step 6. Compare the APKs.

Open Terminal, run the commands: cd $HOME/telegram-android python apkdiff.py telegram_store.apk telegram_built.apk If the APKs are the same, you will see APKs are the same!

If your APKs don't match, please make sure that you chose the correct code version and the right SDK.

Check out the Troubleshooting section first in case you run into trouble.

Reproducible Builds for iOS.

The verification process for iOS builds is, unfortunately, a lot more complex than for Android. The two main issues with Apple's current policies and infrastructure are as follows:

Apple insists on using FairPlay encryption to “protect” even free apps from “app pirates” which makes obtaining the executable code of apps impossible without a jailbroken device. To solve this issue, Apple would simply need to allow submitting unencryptable binaries to the App Store. This would not affect security since the code would still be signed – and would enable anyone to check the integrity of apps supporting reproducible builds without endangering the integrity and security of their devices.

Building your own reproducible binaries is difficult because macOS doesn't support containers like Docker. If Apple followed in the footsteps of Linux (and even Microsoft!) and added container support, it would eliminate the need for steps 1-3 in the guide below.

As things stand now, you'll need a jailbroken device , at least 1,5 hours and approximately 90GB of free space to properly set up a virtual machine for the verification process.

To provide a stable and easily reproducible environment, Telegram iOS builds are compiled on a virtual machine. Parallels is used to verify the builds.

Step 1. Install the Parallels virtual machine.

Parallels can be obtained here, it features a fully-functional trial version.

Step 2. Install the latest version of macOS Catalina.

To download an image that can be installed on the virtual machine, open the App Store, search for “Catalina” and click “View”.

Search for macOS Catalina on App Store > View.

macOS Catalina > Get.

This will open a system pop-up offering to download the OS:

Select 'Install Windows or another OS' > Continue.

Select a file. > Applications (All files) > Install macOS Catalina.

Before starting the installation, configure the virtual machine:

Checkbox 'Customize settings before installation'

Change the name of the virtual machine to macos10_15_Xcode12_2.

Name VM as macos10_15_Xcode12_2.

Hardware > Processors: 2-4 Memory > 4GB may suffice but 8GB is recommended.

At least 2 CPUs + 4 (8 recommended) GB Memory.

You will get something like this:

Parallels may request access to your microphone and camera, this is not required – just press Close .

Install macOS > Continue.

Your Apple ID is also not required, you can choose Set Up Later .

Skip Apple ID with 'Set Up Later'

Use “telegram” for both the account name and password.

Do not ever use the password “telegram” for anything else, it's cursed.

Create a computer account with 'telegram' set both as account name and password.

Now install Parallels tools from the menu bar:

Install Parallels Tools using menu bar > Parallels icon > Actions > Install (Reinstall) Parallels Tools.

After the system restarts, log in. Open Terminal and run: sudo visudo Enter the password “telegram”

Find this line at the end of the file: %admin ALL=(ALL) ALL Press “i” on your keyboard, add “NOPASSWD:” %admin ALL=(ALL) NOPASSWD: ALL Press Escape. Type in “:wq” Press Enter.

Press i to edit the highlighted string.

Enter :wq > press Enter.

In the terminal, run: sudo systemsetup -setcomputersleep Never.

sudo systemsetup -setcomputersleep Never > press Enter.

Step 3. Install SSH keys on the virtual machine.

In the virtual machine, open System Settings > Sharing and enable Remote Login .

In the virtual machine, open Terminal and run: mkdir -p .ssh; nano .ssh/authorized_keys.

In your main OS, open Terminal and run: if [! -e.

/.ssh/id_rsa.pub]; then ssh-keygen -t rsa -b 4096; fi && cat.

If you see the line “Enter file in which to save the key (/Users/…/.ssh/id_rsa):”, press Enter In the virtual machine, press CMD+V Then Ctrl+O , Ctrl+X.

Step 4. Install Xcode version 12.2.

In the virtual machine, open Safari and go to https://developer.apple.com Sign in to your Account:

developer.apple.com > Account > sign in with your Apple ID.

Go to Downloads > More Enter Xcode in the search field and find the version 12.2.

Downloads > More > Xcode 12.2.

Once the installation is complete, open the file Xcode 12.2.xip. The system will unarchive the app into the same folder. Move it to the Applications folder using Finder.

Unarchive Xcode > drag the app to Applications folder.

On the virtual machine, run this command from the terminal: sudo xcode-select -s /Applications/Xcode.app/Contents/Developer.

Shut down the virtual machine.

Shut down the virtual machine.

Step 5. Obtaining the source code.

git clone --recursive https://github.com/TelegramMessenger/telegram-ios.git $HOME/telegram-ios cd $HOME/telegram-ios git checkout release-$

E.g., git checkout release-7.3 . Please note that you need to check out the whole git history as the build version depends on the number of commits in the repository.

Step 6. Downloading Bazel 3.7.0 to $HOME/bazel/bazel.

mkdir -p $HOME/bazel && cd $HOME/bazel curl -O -L https://github.com/bazelbuild/bazel/releases/download/3.7.0/bazel-3.7.0-darwin-x86_64 mv bazel-3.7.0-darwin-x86_64 bazel.

Check that you have downloaded the correct version: chmod +x bazel ./bazel --version.

Step 7. Building the app.

Open Terminal, run the commands: cd $HOME/telegram-ios BAZEL="$HOME/bazel/bazel" sh buildbox/build-telegram.sh verify.

If the environment has been set up correctly, this will start the building process. Note that this step can easily take 30-40 minutes . The average build time on a MacBook Pro (i9 6 core) is 35 minutes.

Once the process is complete the resulting IPA file can be found in build/artifacts/Telegram.ipa All the following steps will be made via Terminal on your main system.

Step 8. Downloading a decrypted version of the app from the App Store.

This step requires a jailbroken device equipped with tools for decrypting apps. We‘d love to make this process more simple but that’s what you get for using Apple tech.

Step 9. Comparing the AppStore build and the version built in the virtual machine.

Install the necessary tools: if ! type brew > /dev/null; then /usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"; fi && brew install python3.

Run python3 tools/ipadiff.py build/artifacts/Telegram.ipa PATH-TO-THE-IPA-FILE-FROM-STEP-9.

cd telegram-ios > Enter python3 tools/ipadiff.py build/artifacts/Telegram.ipa /path/ > Enter.

In case of a successful comparison, you will get a text along these lines:

In case of any mismatches, you'll get a detailed report.

iOS: Notes.

You will get a warning if the archive created in Step 7 contains encrypted files. If all these files are in the PlugIns subfolder, they represent various system extensions (e.g. external sharing, Siri, 3D touch). Decrypting such files using existing ways of receiving app archives via Jailbreak is non-trivial (but we're working on resolving this issue). If you do manage to decrypt them, e.g. on iOS 8, they will be matched.

You will be notified if the archive includes an Apple Watch app . The watch app will soon no longer be included in the archive.

Files with the .car extension are app resource archives (images, sounds) which were compiled and processed specifically for the target device. The App Store processes them in non-trivial ways, we're planning on getting rid of them in future versions.

The LaunchScreen.nib file is an empty file containing a description of the interface which is displayed by the system before the app is launched. It is processed by the App Store in a non-trivial way but doesn't contain any code and therefore may be ignored.

Troubleshooting.

If you encounter any issues with obtaining the code, building and comparing the apps, please contact us at @botsupport and include the hashtag #reproducibleBuilds with your message describing the problem.

Troubleshooting: Android.

Make sure that you checkout the correct version of the code.

Make sure that you build the app using the right SDK.

If the gradle version used in the Dockerfile is not available anymore and building of the Docker image fails, wait for a Dockerfile update or update manually to lastest available version.

We will update this section with information on overcoming other common issues.

smart-tv-telegram Docker.

Q: How do I get an app_id and app_hash?

Q: The video keeps freezing.

A: Check the video bitrate, this bot supports maximum.

Project details.

Project links.

Statistics.

Stars: Forks: Open issues/PRs:

View statistics for this project via Libraries.io, or by using our public dataset on Google BigQuery.

License: GNU Affero General Public License v3 or later (AGPLv3+)

Author: andrew-ld.

Requires: Python >=3.6.

Maintainers.

Classifiers.

License OSI Approved :: GNU Affero General Public License v3 or later (AGPLv3+) OS Independent Python :: 3.

Release history Release notifications | RSS feed.

Download files.

Download the file for your platform. If you're not sure which to choose, learn more about installing packages.