Authenticate: An Open-Source TOTP Authenticator App

William Henderson

Tuesday 8th March 2022

1 Abstract

Two-factor authentication is already widely used as an additional line of defense to protect our data. However, there are few modern and open-source mobile applications to provide this functionality. This report details the research and development process which was undertaken to produce the Authenticate two-factor authentication mobile application, which is available for public use on the Google Play Store.

2 Introduction

Two-factor authentication is an increasingly important technology in today's digital society. Much of modern life is guarded by just a password, often badly chosen and susceptible to attacks, and with an average length of eight characters [1], most passwords could be cracked in an afternoon [2]. With this in mind, using just these eight bytes to protect our entire digital lives from malicious individuals seems reckless at best. It is therefore becoming commonplace to make use of a second authentication method to protect personal data even in the event of a password breach.

3 Background

3.1 Methods of Two-Factor Authentication

3.1.1 SMS Authentication

Two-factor authentication by SMS is one of the most common methods of two-factor authentication as it is considered one of the easiest for the user. When a user wants to authenticate with the server, the server generates a verification code and sends it via SMS to the user's mobile phone. The user then enters the code which is checked with the server, and if valid, the authentication is complete.

There are a number of drawbacks to this method. First and foremost, SMS messages are insecure and can be accessed by mobile network providers and governments. This causes more security holes in the method, as a malicious actor working for either organisation could intercept security codes. Secondly, not everywhere has mobile signal, so supporting this method exclusively could marginalise those in more rural areas as well as less developed countries. Sending an SMS message programmatically is also comparatively very expensive to alternative methods, so companies often try to avoid SMS verification where possible. Finally, SMS messages can take a short while to be delivered to the recipient, slowing down the authentication process.

3.1.2 Email Authentication

Two-factor authentication by email solves a lot of the problems with its SMS counterpart. It works in exactly the same way, except that codes are delivered by email instead of over the mobile network. This avoids the insecurity of SMS messaging as well as the problem of mobile coverage. However, email authentication can still be slow, and relies on the guaranteed security of the email account receiving the verification code. Furthermore, when used to protect an email account, another email account must be used, which is inconvenient for the user.

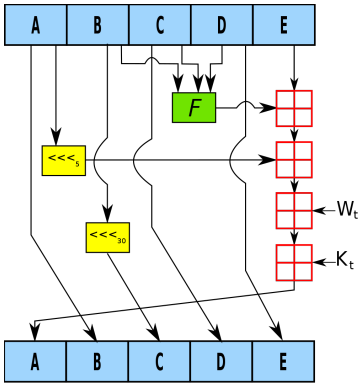
3.1.3 TOTP Authentication

Time-based one-time password (TOTP) authentication solves all of these problems. By making use of a shared secret between the server and user which is only disclosed when the account is set up, and a set of algorithms using the current time as an input, the server and client can independently generate matching security codes provided that their clocks are in sync. Conventionally, the time input into the algorithm is taken to be the UNIX timestamp modulus thirty seconds to allow for slight discrepancy in the clocks. The only way for a third-party to gain access to a TOTP-protected account would be through physical access to the mobile phone or physical key used to generate the codes. This makes TOTP the industry-standard for two-factor authentication thanks to its

high level of security and limited vulnerabilities.

3.2 Cryptography

At the core of the algorithm is the SHA-1 cryptographic hash function. A hash function takes a variable-length string of bytes, performs a number of one-way operations on them, often over a number of iterations, then produces a fixed-length string of bytes. A vital property of a hash function is that the same input will produce the same output, and that the probability of two different inputs producing the same output (a "collision") is almost impossible. The SHA-1 algorithm works by splitting the input into chunks, then performing a number of bitwise operations between the chunks and the ongoing result bytes.



The above diagram [3] illustrates one iteration within the SHA-1 hash function. The algorithm produces a 160-bit digest, so letters A through E represent the five 32-bit chunk words which are used to create this. The yellow boxes represent left rotation operations (left shift where instead of overflowing, bits wrap around), and the red boxes indicate addition, but since the algorithm is working with 32-bit integers, this is effectively addition modulo 32. The function F is different based on how far through the message the algorithm is, but all variants are comprised of bitwise operations.

3.2.1 Hash-Based Message Authentication Code

The hash-based message authentication code algorithm, or HMAC, builds upon a hash function H by introducing the use of a secret key K into the hash. The key is concatenated (k) with the message m (along with some padding and XOR masking) and then hashed, producing a fixed-length string of bytes. This differs from regular hashing, since to generate the same hash from the same input bytes, both parties must share the same secret key, allowing HMAC to be used to both verify the integrity and authenticity of a message.

The algorithm is defined in the internet standard RFC 2104 as the following: [4]

$$\mathrm{HMAC}(K;m) = \mathrm{H}\left(\left(K' \quad 0x5c\right) k\, \mathrm{H}\left(\left(K' \quad 0x36\right) k\, m\right)\right)$$

$$K' = \begin{cases} \mathrm{H}(K) & K \text{ is larger than block size} \\ K & \text{otherwise} \end{cases}$$

3.2.2 HMAC-Based One-Time Password

HMAC can be used to generate a one-time password with the HMAC-based one-time password algorithm, or HOTP. Before the algorithm can be used, both parties must have knowledge of a shared secret key. This is usually generated by the server and given to the client in the form of a QR code when the authentication method is set up.

The algorithm also uses a "counter", a variable known by both parties which is initialised to zero. The counter is incremented by one every time the algorithm is run, meaning that every generated password will be different. This can be a source of issues if the client and the server's counter values get out of sync, which can be resolved if the server tells the client what value to use when generating the password. However, this slows down the authentication process so is not ideal.

The two inputs are combined by using HMAC, taking the message to be the counter value and the key to be the shared secret. The resulting number modulus 10^n is taken as the final output password, where n is the number of digits. It is usually set to six, or occasionally eight, and this detail is also shared at the same time as the secret key.

3.2.3 Time-Based One-Time Password

The only major issue with HOTP is keeping the counter value in sync, so the time-based one-time password algorithm (TOTP) replaces it with the current UNIX timestamp divided by the "interval" with integer division, which is the number of seconds between the code changing.

The UNIX timestamp is the number of seconds which have passed since 1st January 1970. It is used instead of an arbitrary counter value since the time is one of the only things which is generally synchronised between most devices. The interval is usually set to 30 seconds to allow for slight discrepancies between the client and the server's clocks, but this can also be changed.

$$TOTP(K) = HOTP\left(K; \left\lfloor \frac{T}{T_X} \right\rfloor\right)$$

4 Methodology

4.1 Design (Figma)

The Authenticate mobile app was designed using the online design tool Figma. This allows the interface to be designed visually, while still providing a number of useful features for when it comes to converting the visual design into code.

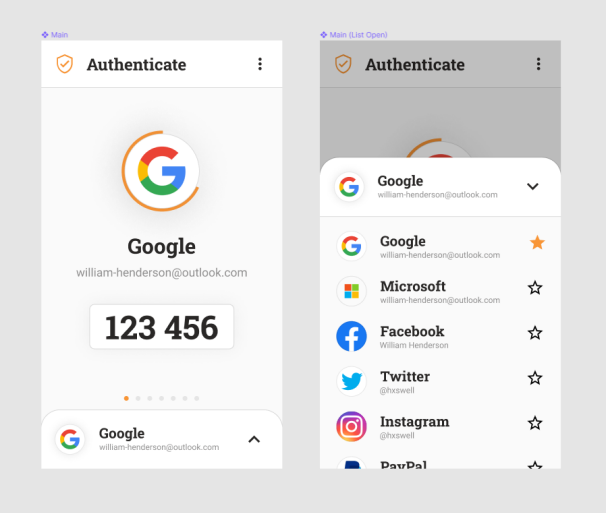


Figure 1: Screenshot of Figma Design

4.2 Programming

4.2.1 TypeScript

The Authenticate mobile app is written in TypeScript, a strongly-typed extension of JavaScript which catches a number of type errors at compile-time. Since it compiles down to JavaScript, it can be used on a number of platforms, including the web, mobile applications through React Native, and desktop applications through Node.js, Deno and/or Electron.

Used by 78% of JavaScript developers [7], TypeScript allows for quicker and more robust application development in many fields. In the Authenticate mobile app, it is used for everything, from the front-end design to the core cryptography.

4.2.2 React Native

Authenticate is built using the React Native mobile app framework, which is developed by Facebook in collaboration with the open-source community. It

allows for the creation of mobile apps using JavaScript or TypeScript paired with the React front-end framework.

Traditional mobile apps require two separate codebases, one for iOS written in Swift and one for Android written in Java or Kotlin. This makes development much slower and causes differences in design and operation between what are entirely separate apps. React Native changes this by allowing one JavaScript-based codebase to be used for both platforms, converting platform-agnostic components such as `View` to their native counterparts on each platform, in this case `UIView` on iOS and `android.view` on Android.

4.2.3 Expo

One drawback of React Native is that it requires the build systems for both iOS (Xcode) and Android (Android Studio) to be installed in order to compile and run the application. Expo is built upon React Native and allows users to securely compile and run applications in the cloud, without having any of the build systems installed.