

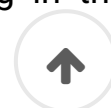
Leto Bukarica

Tech Lead @ Deploy

August 21, 2017 · 10 minutes read

Intro

Here, we will present the concepts and a use case for adding two-factor authentication in a web application. We have used Java for some reference links, but nothing in this tutorial is Java-specific.



Two-factor authentication (2FA) is a subset of **multi-factor authentication** (which requires authentication from at least two of these categories: (1) something a user knows, (2)

something he has and/or (3) something “he is” (a physical characteristic). We are going to use two factors in this example: the user’s knowledge (user/pass combination) and something in the user’s possession (mobile smart phone).


Mobile phone 2FA has many different implementations – SMS, dedicated service login, token generation with reference number, etc. We are going to use the detached implementation offered by **Google Authenticator** (GA) application. Detached in this context means that the phone holder doesn’t have to be online to use the app nor that Google has or needs any information from the user.


As a showcase, we are considering an already existing simple web application that has a security system in place (user/pass-based). We want to present 2FA on the login screen and require it for some users. Depending on the use case, it can be added just on some features or sub-services or mixed with OAuth. The underlying mechanisms are similar.

User experience

Since the mechanism was not implemented on the first release of the web application, old users will see a new field (verification code) on the login screen. We could employ a two-step login process but this one simplifies implementation (and stumbles brute-force attack on either verification code or password).

Please Sign In

Email Address
 

Password
 

Verification Code


The Login Screen

After successful login, users flagged with 2FA required need to scan a QR-code using the GA application. The user should now have a new entry in GA with the app name, username and a login code that is refreshed every 30 seconds.

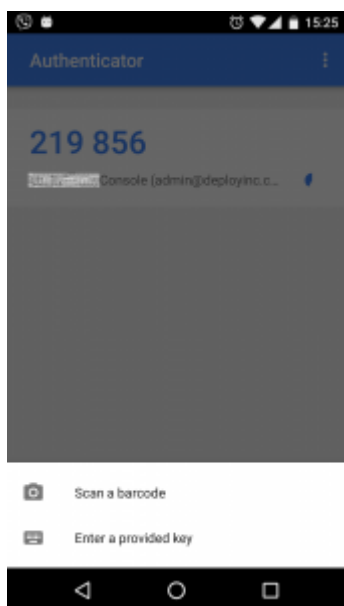
Please Sign In

Hi *admin@deployinc.com*

Scan this Barcode using Google Authenticator app on your phone to use it during login



2FA



GA

Tip: The user now needs to log in with username/password/verification code to gain access to the system.

Note: Since GA is detached from the system, the user has to click the OK button to be able to continue. Clicking that button is a signal to the system that all further logins from that user will require a verification code. Clicking an OK button but failing to scan the code will disable that user from further logins to the system. Some manual action by the web-app administrator is needed to remedy this.

Background

In the basic username/password combination, **the user and the system** share a secret password (or hashed/salted password) used in the login process. In this case, **the system and the mobile device** share a randomly-generated key. That key is generated by the system, stored for that user and presented in the QR-code. After reading it, the device has the key stored in its GA.

Since the verification code used in the login is 6 digits long, it can be brute-forced pretty easily (presuming that the attacker knows the username/password). By limiting the code to six digits, the realm of valid codes is effectively squeezed to 999,999 different values (which could be brute-forced within a matter of hours).

That's one of the reasons the time factor is introduced. The other reason is that the original shared key is never used in the communication. On the other hand, it isn't realistic to expect a user to type in something like 16 alpha-numeric characters in a 30s time frame and then also type in a username and password.

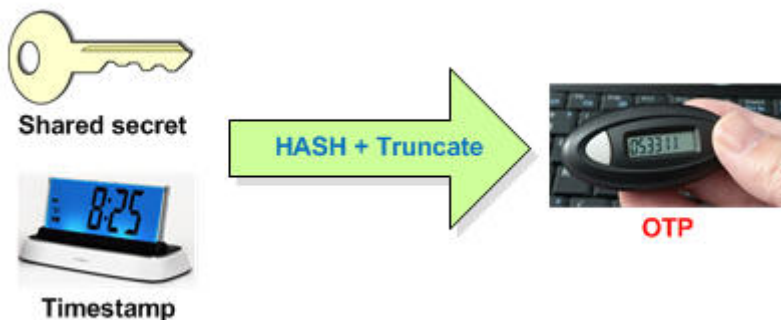


Image Source

A current timestamp (in milliseconds) is rounded down to a 30s period and is used to generate a 6 digit code. It introduces a pattern of change and also reduces effectiveness of sniffing and replaying since the token is valid only for a set period of time.

The algorithm that is used by GA is **TOTP**. More information can be found [here](#). For implementation libraries, a simple [Google search](#) should be enough.

To generate a verification code, both the server and client use:

- Secret key
- Current time
- TOTP algorithm

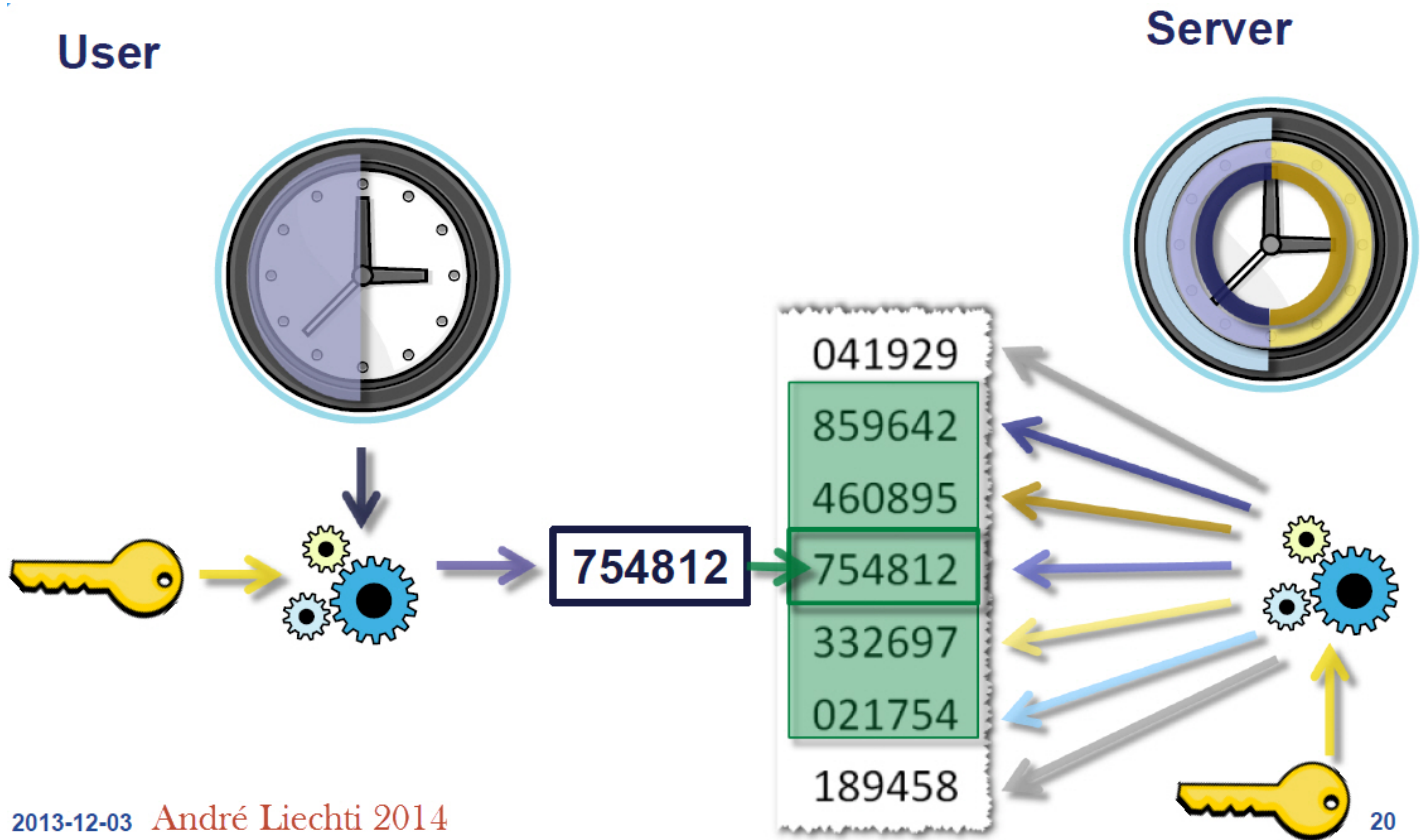


Image Source

Note: Client time desynchronization will disable the user from logging in. Server time desynchronization disables all of the users from logging in.

Implementation

Spring (+ Spring Security) was used as a basis in our example, but the details won't be presented here. There is a [useful tutorial](#) of this process.

New dependencies:

- Library that can transform secret key + timestamp => validation code – TOTP reference library.
- Random code generator – usually exists in TOTP library.

- QR code image generator – isn't necessary. **Google chart services** can be used to create the QR code or the plaintext key can be presented to the user and manually entered in the GA. The GA QR code has a specific **format**.

The database layer only needs two new fields in the user table:

Field	Type	Description
secret	String	Shared key for the specific user
using2fs	Boolean	2FA is used for that user

using2fa is a three-state logic field which isn't the best solution but will suffice for this usage:

- Null – no 2FA for the user
- 0 – present 2FA on next login
- 1 – is using 2FA

Note: If the secret key should not be stored in the DB as plain text, it can be encrypted using symmetric encryption salted with non-changing field (username) but **must not be hashed**. The system needs to have an original secret key at validation point.

Business layer tasks:

1. Detect that the user is marked for 2FA but still doesn't have the secret key issued.
2. Present the user with the shared key, store it, and flag the user as using 2FA.
3. Detect that the user is logged in but did not present the 2FA code for requested functionality (not fully authenticated).

4. Check the verification code using the secret key for that user/timestamp and verify or deny the user.

Resolutions:

1. Check if `using2fs==0`
2. Generate secret code using the library, present it to the UI (optionally, using QR code), store it in the secret field, and set `using2fs=1`
3. Implementation-specific – Check the user session for authentication details
4. Use the secret field and current timestamp to create verification code and compare it with the user-entered code

Conclusion

The purpose of this tutorial is to present concepts and implementation guidelines in a concise and simple manner. Finding references which weren't cluttered with a detailed algorithm description and specific implementation details was hard, so writing this tutorial came naturally.

2FA is a useful security concept besides being fancy. We can see how easy is to set it up even on an existing application. One of the more relaxed usages would be to add it to only some parts of the system or some functionalities so the user doesn't perceive it as a hassle but as a much-needed security feature.

I hope this will be as useful to you as the exploration of the subject was to me. Happy coding!

Tags: 2FA, authentication, google authenticator, tutorial, two-factor authentication



Share



Print page



2 Likes

Recent Posts

KRACKing the WPA2

Better Code Reviews

Two-factor authentication (Google Authenticator)

Email Evolution

YOU'RE NEVER ALONE

Categories

Design

General

Platforms

Software Development

Tips N Tricks

Web

Tags

2016 authentication avocode big data branching strategy **CSS** custom elements cyber security data cleaning data manipulation designs eDm template email template html git hawtio helios how to open sketch file on windows html best practices html imports lua mysql newsletter html newsletter template no-js examples perl

photoshop privacy pros and cons **python** r regex ruby scripting language security shadow dom **shell** shell
tools sketch sketch file templates terminal tools text search uber web components web console

We're

Engineers

Leads

Producers

Analysts

Technologists

Deploy



Austin Los Angeles Belgrade Seattle

© Copyright 2009-2017 Deploy Inc. | 1 888 685 7640 | info@deployinc.com