OTP Authenticator v1.2

Intro

Want to have something special? How about cool security? Using any TOTP (Open TOTP) smartphone application **(See Software section)** or hardware TOTP device **(See Hardware section)**, your clients will never again have to remember password or worry about security. Just one look at smartphone or keychain device and they are in. Use your existing smartphone or tablet running any application that supports the open TOTP standard, or one of many cheap and stylish hardware TOTP Devices (ex. keychain devices). Always near you, and best of all - passwords can't be stolen, they change every 30 seconds.

TOTP Authentication process information

For this to work, correct time is essential. Since nowadays most devices and computers are internet time synchronized, you probably won't need to worry about that. But... better to check twice, than be sorry ;) Also, since Unity runs on client side, client's clock should be internet time synchronized. Other solution would be to pull time from server that is time synced, and with delay time of data transfer included, you should not be more than 1 sec off. This is perfectly acceptable. I will work on such time pull system in near future.

Files included in package

Lib/OTPAuthenticator.dll

- Main assembly file, containing logic for handling TOTP authentication requests. You can expand it in Project window, and use shown item as regular Unity C# script.

Scenes/OTPTest

- Test scene, showing data related in process of authentication, and giving you abillity to play around with (few of) them. **(See Testing section)**

Scenes/OTPSample

- Sample scene, showing proposed usage in production builds. Includes Login dialog and Registration dialog. **(See Testing section)**

Scripts/OTPTest.cs

- Unity written C# script used in OTPTest scene, handles instancing and communicating with main assembly (OTPAuthenticator.dll), GUI events, and some helpers needed for this sample (Md5 hashing, downloading, etc.). Has lots of debug calls, and parameters not shown in production builds.

Scripts/OTPSample.cs

- Unity written C# script used in OTPSample scene, handles instancing and communicating with main assembly (OTPAuthenticator.dll), GUI events, and some helpers needed for this sample (Md5 hashing, downloading, etc.). It shows proposed production builds usage.

Testing (usage)

OTPTest scene

- As mentioned before, this scene has lots of debug calls, and shows parameters not shown in production builds.

Identity is field for entering general user identification, most likely username, email, or customer number. Field is editable, and you can play around with it. It's not essential in authentication process, but is included in QR code generation. QR reader devices will use this as title for related OTP generation. After each change you will get a new QR code.

Secret is field containing essential information for authentication. It is also editable, and contains 20 chars. They are actually 10 bytes in hex format. You can play around with it, but keep it 20 chars long (for compatibility with applications and devices), and use A-F and 0-9. For each change you will get a new QR code. **IMPORTANT:** For purpose of this sample/demo, generation of secret is done following way: take Identity field, make Md5 hash of it, and use first 10 bytes for secret. Thats why secret changes after you change Identity. In your production builds, you should use some other method for generating secrets, this was just an idea for this sample/demo. Is not mandatory that secret generation is Identity related. Other ideas are: PC hardware fingerprint, just random numbers, etc.

OneTimePassword shows current OTP (read: password). Will change every 30 seconds.

QRCodeUrl shows assembled QR Code image retrieval Url. Just for info, image downloaded from that url is shown on right side of the dialog.

ValidTime is time in seconds, that current OTP is still valid. Once timer reaches 0, new OTP is generated, and countdown is set to 30 seconds. (may be nice info in production)

Timestamp is last timestamp used for OTP generation (Geeks only)

Hmac shows HMAC-SHA-1 Keyed Hash used in OTP generation (also Geeks only)

HOW TO TEST

- Enter any username, email or number in Identity field, scan the given QR code with an smartphone OTP application (ex. Google authenticator). Enter the application, go to Menu > Add account > Scan barcode, and scan QR right off your screen. You can also manually enter account data, go to Menu > Add account >> Manually add account, Identity (or whatever, but preferably same to avoid misunderstandings) in Account field, Secret to Key field, and choose Time Based. If Secret was correctly entered (that's why I prefer QR scanning), you will see your account name, current one time password, and small timer counting down. They should match Identity, OneTimePassword and ValidTime fields displayed in Unity. It does not matter what you do (close app, restart phone, computer), they will match while time is synchronized and secret is not changed. If time was off (not synchronized), and OTP's didn't match, they will after you synchronize time on side that is off (preferably on both), no need for QR rescan. Hint: if working, do not delete that entry from your smartphone, will be useful in OTPSample scene.

OTPSample scene

- This scene is proposed usage of main assembly (OTPAuthenticator.dll) in production builds. It has 2 dialogs: Login and Registration. They simulate real life scenario of user registering/authenticating on a system.

Login dialog has 2 fields and 2 buttons. If you present yourself as an existing user, you will be entering your username or email (identity), then look at your smartphone OTP application (ex. Google authenticator) or your keychain device and type in current OTP into password field. After hitting Login, system will reply with Auth Success ot Auth Fail. In production builds, at this point you should check in your database if user with that identity exist, and if does, fetch secret for that user. Then provide both values to main assembly via provided functions. In this sample, check and fetch from database are not implemented, and to generate secret, we use first 10 bytes from MD5 hash of entered Identity. **See TESTING - OTPTest Scene - Secret field.** Now, if you scanned QR from OTPTest scene, and still have it on your smartphone, you can enter that identity and current OTP from phone to Unity and you will be authenticated. In case you present yourself as new user, click on Register button. Dialog will appear asking your identity (username/email) and secret. After you enter some identity, you will see a generated

secret (above mentioned method), but you can put in something you generated (from code, 20 chars, A-F, 0-9, and disable editing), and freshly generated QR code. Prompt user to scan QR or enter secret manually into an smartphone OTP app (so should you now, can use also same identity used in OTPTest scene), and press Done. In production builds, at this moment you should save entered identity and secret to your database. This is not implemented in this sample. Now you are again at Login screen, with username already entered, just need to type current OTP from your smartphone.
Note: If you will use hardware devices (keychain devices with small lcd), at register screen you should enable user to enter it's own secret that came on a sticker with that specific device. HW devices are already time synchronized.

Software & Hardware

- Google authenticator (Android, iPhone, iPodTouch, iPad, BlackBerry) - works flawlessly http://support.google.com/accounts/bin/answer.py?hl=en&answer=1066447

- RCDevs (to be Checked), apps and devices for almost any platform http://www.rcdevs.com/products/openotp/
-

References

https://tools.ietf.org/html/rfc4226
https://code.google.com/p/google-authenticator/wiki/KeyUriFormat

Version history

1.0 - Initial release
1.1 - Added display/editing of secret, for manual entry into/from devices that don't support QR scanning. (Older phones with J2ME OTP app, or keychain devices)
1.2 - Added documentation, and small improvements in sample/test scene