Yubiduino: TOTP on an Arduino

Jessica Fleck, Brandon Mills, Paul Tela
Department of Computer Science and Engineering
The Ohio State University
Columbus, OH 43210
{fleck.48, mills.511, tela.3}@osu.edu

***Abstract*—Two-factor authorization (2FA) is a more secure method for authenticating users during login. Many 2FA systems, including the one used by the popular Gmail service, follow the Time-based One Time Password (TOTP) algorithm standardized in RFC 6238. Per the standard, the client and server agree on a shared secret key and starting timestamp at setup. Then, each time the user attempts to log in, the algorithm generates an ephemeral numeric passcode by running a cryptographic function with the shared secret key and the number of elapsed time intervals since the starting timestamp as inputs. However, this system requires a user to obtain and manually enter another unique password every time they log in. Inspired by the commercial YubiKey multi-factor authorization key, we designed and built a device, called a Yubiduino, that calculates and automatically enters the one-time password on the user's behalf. The device consists of an Arduino Due microcontroller, a DS3231 real time clock with a battery backup, a microSD card for persistent storage, and a button that the user presses to activate the algorithm. After setup, the user has only to plug the Yubiduino into a USB port, select the field in the login form for the one-time password, and press the button, and it will calculate and type the password automatically.**

I. Introduction

Internet services permeate every facet of modern life. Communication, finances, and even our very identities all live online. Through the power of data, Google, Facebook, and dating comany OkCupid know more about their users than their users know about themselves. This trove of personal and financial information is an enticing treasure to malfeasants, who contrive ever more sophisticated attacks through backdoors and social engineering. Multi-factor authentication stands in the way of an attempted account hijacking by verifying identity through a combination of things only the user *knows*, *has*, or *is*. In the case of the Yubiduino, the user's standard login password is something only the user *knows*, and the one-time password is generated from the secret key that nobody but the user *has*. Without both, no attacker can gain illegitimate access to an account. The Yubiduino lets users add this second layer of security to any service that supports Two Factor Authentication via the Time-based One Time Password standard, including Gmail, GitHub, and Facebook.

II. System Design

Hardware assembly was necessarily the first step in building the Yubiduino. The part list was fairly short:

- Arduino Due microcontroller
- Breadboard
- Arduino-compatible Ethernet shield
- MicroSD card
- DS3231 real-time clock module with backup battery
- Momentary push-button switch
- 10 kΩ resistor
- Various wires

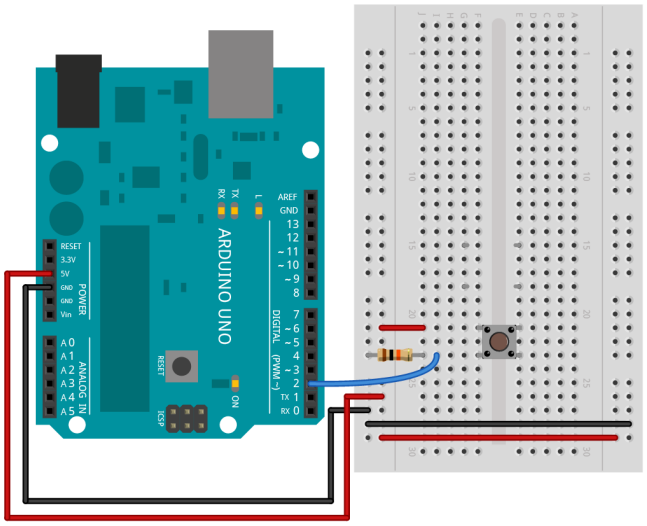

Fig. 1. Button Circuit

Assembly consisted of connecting the button as shown in Figure 1 and the real-time clock (RTC) module as shown in Figure 2, then stacking the Ethernet shield directly on top of the Arduino.

As part of two-factor authorization setup, Gmail and Facebook will make available a base32-encoded secret key. To set up the Yubiduino, this key should be copied to a file named "key.txt" on the Yubiduino's FAT-formatted microSD card. This must only be done once. When the user logs in online, plugs in the Yubiduino, and activates it by pressing the button, it emulates a USB keyboard and "types" the digits of the one-time password into the currently-focused form field. The code itself was determined by running the TOTP algorithm using the shared secret key from the microSD card and the timestamp from the RTC.

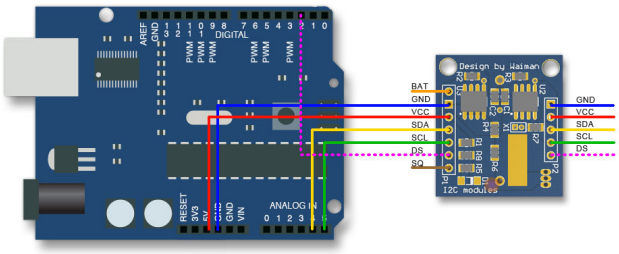

Fig. 2. RTC Circuit

III. Implementation and Evaluation

A. Implementation

Yubiduino's software was implemented in C++ using the AVR-GCC compiler to target the Arduino Due board. This board uses a 32 bit ARM core micro controller running at 84Mhz. This micro controller allows for 4 byte wide data operations to occur in one clock cycle, which provides significant performance benefits for the cryptographic hashing code.

Several libraries were used in order to build the Yubiduino software package. Standard libraries used were `Serial`, `Keyboard`, `SD`, and `SPI`. In addition to these standard libraries, two third party libraries were used. These were `DS3231` and `Sha`.

The `Serial` library provides a way for the Arduino to communicate via a serial interface. This is primarily used for interacting with the Arduino using the Arduino IDE's built in Serial Monitor. The `Keyboard` library is used to allow the Arduino to send keyboard input to a connected computer. The `SD` library is used to read and write files on an SD card and supports both the FAT16 and FAT32 file systems. An ethernet shield was used in order to provide an SD card input. The `SPI` library allows for access to the Serial Peripheral Interface. This interface allows for short distance communication between micro controllers. This library was used to communicate with the Real Time Clock (RTC) over I^2C.

Third party libraries were used when standard library functionality was not available. The `DS3231` library was used to communicate with the RTC. It provided convenience methods for reading the current time and converting the time between different formats. The `Sha` library provides SHA1 and HMAC-SHA1 cryptographic hashing capabilities. In addition to this, a `base32_decode` function was used from the open source Google Authenticator project. This implementation of `base32_decode` was chosen to ensure that the secret key would be decoded in the exact way intended.

The Arduino API specifies two methods that can be used as program entry points. The `setup` method is run once any time the Arduino is powered on or reset. It is useful for performing setup tasks. Yubiduino uses this method to perform general setup including enabling input and output pins, initializing third party libraries, and several other actions outlined below. The other method is `loop`. This method is run continuously as long as the Arduino is powered on. It is generally used to check for specific conditions and run event handling routines when they are met. For Yubiduino, this means listening for a button press and generating a token. If a button press is not detected, the Yubiduino will sleep for a short amount of time in order to conserve power and clock cycles.

The first step in implementing TOTP on the Arduino was to set the time on the RTC. This is necessary to make sure that the interval offset calculated matches what the server expects. Setting the time on the RTC is an operation that only needed to be performed once. In order to make this operation as user friendly as possible, a serial utility was created. The utility is invoked by entering `set time` at the serial monitor while the Arduino is running. The Arduino will recognize this as a valid command and prompt for the current UTC year, month, day, hour, minute, and second. Once this information is entered the Arduino will use the `DS3231` library to communicate the correct time to the RTC. The serial utility can also be used to read the current time by entering `time` at the serial monitor.

Once the time is set, the next piece of information needed is the secret key. This is determined by the service the user is setting up 2FA with, and will be a 32 character string. This string is base 32 encoded in order to prevent confusion resulting from similar looking letters and digits. Yubiduino will look for this key in a file on the SD card called key.txt. The key can be entered with or without spaces. On startup, the key is read from the file and decoded into a 20 byte string. If the key file cannot be found, Yubiduino will log an error to the serial monitor and refuse to start. The serial utility can also be used to view the key stored on the Yubiduino by entering `key` at the serial monitor prompt.

After the `setup` method has completed, the `loop` method is invoked. This method continuously reads the `BUTTON_PIN`. When the signal from the pin changes, it means the button has been pressed. At this point it allocates memory for a new token, which is stored in a 6 byte char array. It also gets the current time from the RTC and converts it into a UNIX timestamp. Yubiduino then passes the secret key, key length, current time, and token buffer into the `totp` method.

The `totp` method performs the necessary calculations to turn the current time and secret key into a token. The first step in this process is to calculate the time interval, `C`. This is done by dividing the current time by the interval value, which by default is 30 seconds. `C` is returned as a `long`, which is a 32 bit signed integer. It must be converted into a 64 bit unsigned integer before it can be hashed. This is done by creating an 8 byte array, setting the 4 most significant bytes to `0x00`, and setting the remaining 4 bytes to the value stored in the `long`. The next step is to hash this value using HMAC-SHA1. The `Sha` library provides a convenient way of doing

this. The secret key used for the hash is the same secret key set previously when setting up the Yubiduino. The resulting hash is returned as a 20 byte array, `H`. The next step in the process is to truncate `H` in such a way that the resulting value is cryptographically sound. This is done by taking the 4 least significant bits of `H`, called `O`. 4 bytes are taken from `H`, starting at `O`. The most significant bit is dropped, and the remaining bits are stored as an unsigned 32 bit integer, `I`. The token is the lowest 6 digits of `I` represented in base 10. If `I` is fewer than 6 digits, it is left padded with 0s.

Once the token is calculated, it is sent to the user through the `Keyboard` interface. This allows the Yubiduino to fill in the correct token without the user needing to type it themselves, preventing errors from typos and timing.

B. Evaluation

For the Yubiduino's final test, we set up two-factor authentication on a Gmail account, saved the shared secret key to the microSD card, and successfully logged into the account on multiple occasions using the Yubiduino to generate and input the authentication code automatically. By the very real-world measure of using it on a popular production system, we consider the Yubiduino a success. Hashing speed was a particular concern going into the project; fortunately, it generates authentication codes apparently instantaneously. However, the Yubiduino is not without room for improvement. At its current size, it is impractical for everyday use, though we believe we have proven the concept and that miniaturization of the hardware is an achievable solution.

IV. Conclusion

Users trust Internet companies to protect more private information than ever before. Two-factor authentication is a field-tested method that provides some much-needed additional account security. With the Yubiduino, we successfully designed and built an assistive device that implements the open, standardized Time-based One-Time Password algorithm, and we proved that success by using it to log into a real-world commercial system over the Internet.

Additional references
https://www.authy.com/add-2-factor-authentication-facebook
https://help.github.com/articles/about-two-factor-authentication/
https://code.google.com/p/google-authenticator/source/browse/libpam/base32.c

References

[1] *Time-based One-time Password Algorithm*, 2014 (accessed December 3, 2014). [Online]. Available: http://en.wikipedia.org/wiki/Time-based_One-time_Password_Algorithm

[2] *Multi-factor authentication*, 2014 (accessed December 3, 2014). [Online]. Available: http://en.wikipedia.org/wiki/Multi-factor_authentication

[3] D. M'Raihi, *TOTP: Time-Based One-Time Password Algorithm*, 2011 (accessed December 3, 2014). [Online]. Available: http://tools.ietf.org/html/rfc6238

[4] Arduino, *Arduino Due*, 2014 (accessed December 3, 2014). [Online]. Available: http://arduino.cc/en/Main/ArduinoBoardDue

[5] ——, *Serial*, 2014 (accessed December 3, 2014). [Online]. Available: http://arduino.cc/en/reference/serial

[6] ——, *Mouse and Keyboard libraries*, 2014 (accessed December 3, 2014). [Online]. Available: http://arduino.cc/en/Reference/MouseKeyboard

[7] Yubico, *STRONG MULTI-FACTOR AUTHENTICATION*, 2014 (accessed December 3, 2014). [Online]. Available: https://www.yubico.com/about/intro/yubikey/

[8] Arduino, *Keyboard Message*, 2014 (accessed December 3, 2014). [Online]. Available: http://www.arduino.cc/en/Tutorial/KeyboardMessage

[9] ——, *SD Library*, 2014 (accessed December 3, 2014). [Online]. Available: http://www.arduino.cc/en/Reference/SD

[10] ——, *SPI Library*, 2014 (accessed December 3, 2014). [Online]. Available: http://www.arduino.cc/en/Reference/SPI

[11] D. M'Raihi, S. Machani, M. Pei, and J. Rydell, *TOTP: Time-Based One-Time Password Algorithm*, 2014 (accessed December 3, 2014). [Online]. Available: http://tools.ietf.org/html/rfc6238

[12] Authy, *Setup Facebook Two Factor Authentication*, 2014 (accessed December 3, 2014). [Online]. Available: https://www.authy.com/add-2-factor-authentication-facebook

[13] Github, *About Two-Factor Authentication*, 2014 (accessed December 3, 2014). [Online]. Available: https://help.github.com/articles/about-two-factor-authentication/