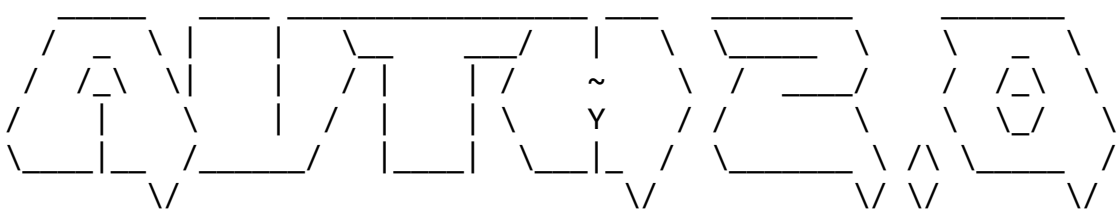


Passwords suck

- Good ones are hard to remember
- Difficult to have a different one for every site
- They can be intercepted, stolen from the client, stolen from the server, etc.
- What can we do to improve on passwords?

Factors of Authentication

1. Something you know
   - Password
2. Something you have
   - Phone, RSA token, etc
3. Something you are
   - Finger prints, Iris scan, DNA, etc.

Example

- When I log into my insurance site I am asked for a username and password
  - How many factors is that?
- After I log in I am asked to provide a “PIN”
  - Now how many factors have I used now?
- If I’m coming from an “unfamiliar computer” I am asked a secret question like: “What is the model of your first car?”
  - Now how many factors?

1 factor the whole time

- Only things I know

Example 2

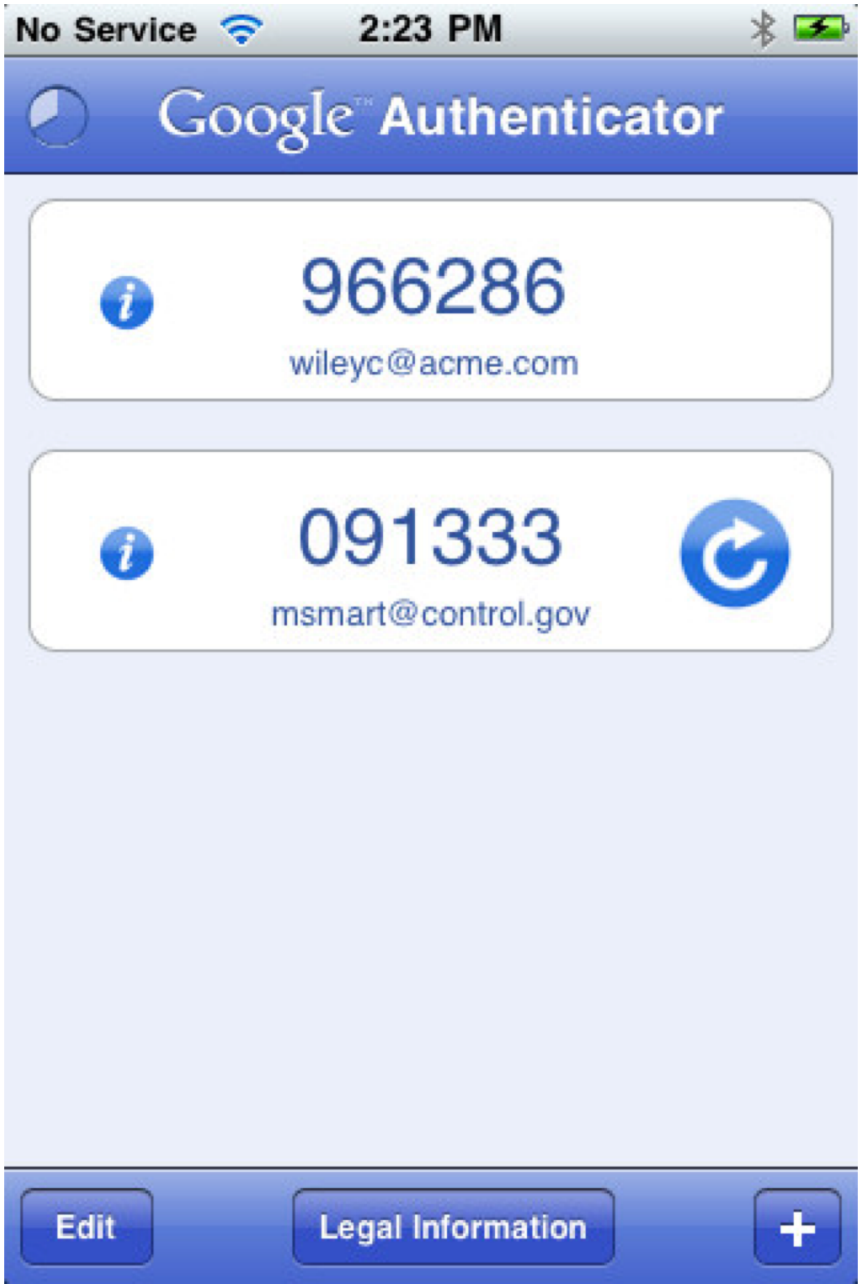
- When I log into Gmail I am asked for my username and password
- Then I am asked for my “Google Authenticator code”
- I pull out my phone, check the code which changes every 30 seconds, and enter that in
- How many factors have I used?

2 factors!

- Something I know (password)
- Something I have (phone)

How Google Authenticator Works

- Google Authenticator implements TOTP security tokens from RFC6238 in a mobile app
- Provides a 6 digit one-time password for login
- Requires a shared secret for setup
- Conforms to specification so other sites can use the Google Auth app
  - Coinbase
  - Lastpass


No Service
2:23 PM
Google™ Authenticator
966286
wileyc@acme.com
091333
msmart@control.gov
Edit
Legal Information
+

Under the Hood - Overview

- Web application generates an 80-bit secret key for each user
  - Served as 16 character base32 string or QR code
- Client Creates an HMAC-SHA1 using the secret key
  - The message HMAC-ed (hashed) can be:
    - The number of 30 second periods having elapsed since the Unix epoch (number of seconds since Thursday January 1, 1970)
    - The counter that is incremented with each new code
- A portion of the HMAC is converted to a 6 digit code

Under the Hood

Pseudocode for Time OTP [edit]

```
function GoogleAuthenticatorCode(string secret)
    key := base32decode(secret)
    message := floor(current Unix time / 30)
    hash := HMAC-SHA1(key, message)
    offset := value of last nibble of hash
    truncatedHash := hash[offset..offset+3]  //4 bytes starting at the offset
    Set the first bit of truncatedHash to zero  //remove the most significant bit
    code := truncatedHash mod 1000000
    pad code with 0 until length of code is 6
    return code
```

Can you think of any possible issues?

Possible Issues

- Shared Secret
  - Server could be compromised
  - Requires a cryptographically secure seed
- Clock Skew/Drift
  - Clock on phone and clock on server drift apart
  - To compensate the window for a valid code can be increased, but that makes the window of opportunity for compromise larger
  - More to keep track of and worry about server side
- Window of Opportunity
  - A dedicated attacker might still have ~30 seconds

A clever 2FA system from Twitter

Twitter's 2FA

- Consists of 2 parts
  - Online
    - Similar to other 2FA with slight twist
  - Backup codes
    - Inspired by S/KEY

Online 2FA

- 2 RSA 2048bit keys are generated (public and private)
- Public is given to Twitter and stored on their servers
- Private is stored on the phone
- When you log in the servers send a “challenge” (random secret) to the phone
- If the user approves the phone signs the random secret with the private key and sends it back
- The server then verifies the signature, and issues a session token to the client logging in
- No clock to worry about

Backup Codes

- A random key is generated by the phone
- The key is hashed 10,000 times
- The 10,000x hash is kept on the server
- The key is then hashed 9,999 times and written down
- If the user needs to log in they can provide the 9,999 iteration hash, which is then hashed once, and compared to the 10,000x hash on the server
- If they match then they must have the secret
- A new 9,998x hash is given to the user to be written down
- The 9,999x hash is stored on the server

What is a digital service that everyone here has access to?

Hint: it’s an old service that mimics a similar service in the outside world

Email

Square Cash

- When you sign up you are given a temporary URL sent to your email
- When you follow that link you enter your debit card
- On the back-end the debit card and your email are now “linked”
- Assumptions
  - Only someone with access to the email would be able to follow the URL
  - Only someone with the debit card would be able to fill out the form located at the URL
  - Therefore we can assume that the owner of the card and the owner of the email account are the same person

Square Cash

- When I want to send money I simply
  - send an email to the person I want to send money to
  - CC pay@square.com and put the amount in the subject line
- If they have an account the money is transferred (charge to me and refund to them)
- If they don’t have an account they are sent a signup link to link their email to their debit card

Square Cash

- No typical account signup
- No additional username and password
- If 2FA is setup on the email account then it is also setup on the Square account
- When you forget a password you use your access to the email to recover it, so why not just use access to your email as your password to begin with

Resources

- https://blog.twitter.com/2013/login-verification-on-twitter-for-iphone-and-android
- https://en.wikipedia.org/wiki/Google_Authenticator