Make the Cyber Safer with Multi-factor Authentication

Mr. Orange's Bad Day

You're fake news!

Mr. Orange is having a rough day.

He can't seem to access his offshore Cayman bank account. He has tried his secure passphrase ("You're fake news!") several times without any luck.

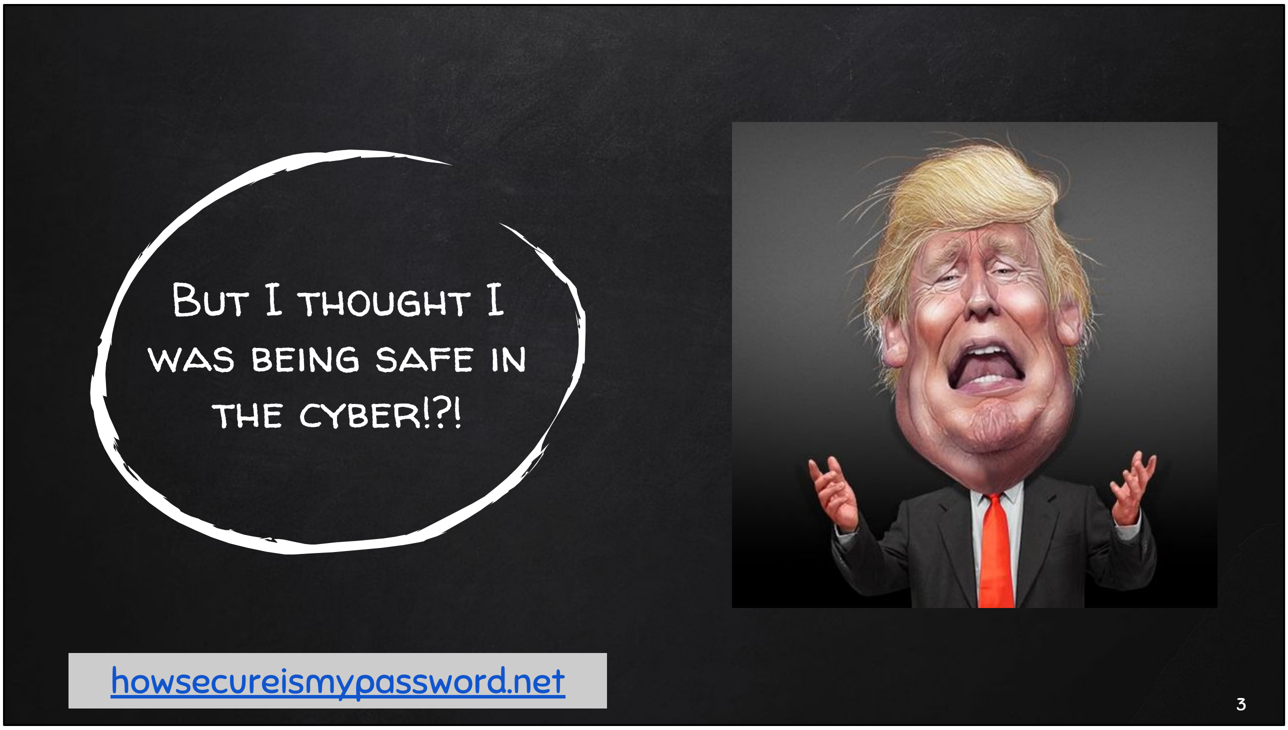
Time is running out to pay his handler, Mr. Red.

Desperate, he decides to raid the accounts used to front his fake charities, luckily he has additional SMS protection on those accounts.

Unfortunately, he has to wait for Air Force One to land before he can receive text verification codes, so he bides his time by border wall tweeting.

Upon landing he receives several delayed text messages notifying him of unexpected bank withdrawals. By the time he can get Sarah Huckabee Sanders to contact his mobile carrier and banks, it's too late, all of his Cayman and fake charity accounts have been drained dry.

Apparently a hacker has gotten the best of Mr. Orange.



But I thought I was being safe in the cyber!?!

howsecureismypassword.net

3

What did Mr. Orange do wrong?

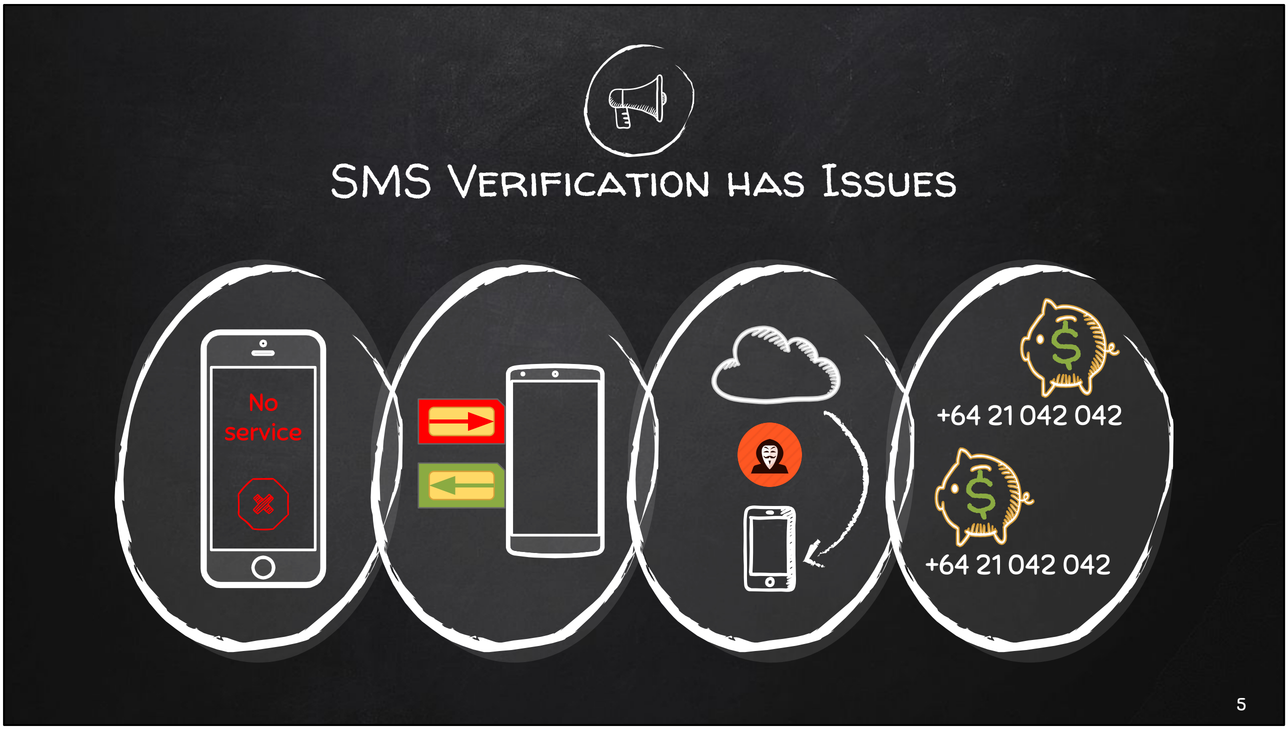
- He had a reasonably secure passphrase - according to https://howsecureismypassword.net/ it would take 610 trillion years to crack “You’re fake news!”.
- He had SMS text code verification.

How was his account compromised?

So what happened?

Combination of user mistakes and host mistakes resulted in Mr. Orange's very bad day.

- Account details contained in Ashley Madison leak - bank names, email address
- Pass phrases easily guessed from social media or hacked from Twitter log files, where passwords were recorded in plain text https://www.adweek.com/digital/twitter-corrected-a-bug-that-caused-passwords-to-be-stored-in-plain-text/
- SIM card swapped using social engineering by a skilled Trump impersonator
- Same pass phrase and SMS 2nd step used on all accounts



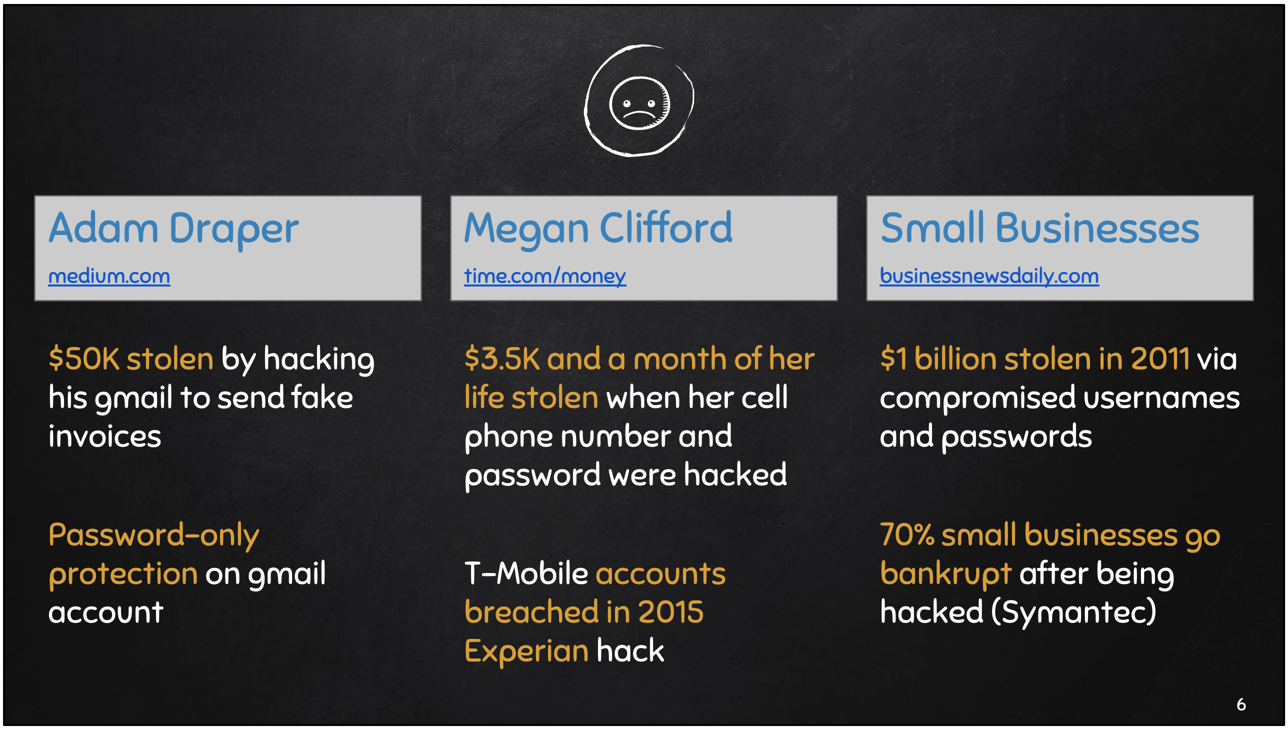
SMS Verification has Issues

Overall SMS codes are a flawed choice as a 2nd login verification

- SMS **messages not always available** (wifi but no cell coverage, foreign country, etc)
- **SIM Swap attacks** are completely out of the user's control
- **SMS messages can be intercepted** because of the dated and insecure **SS7** phone routing system
- SMS verification has the **same flaw as reused passwords**

References

- https://www.makeuseof.com/tag/two-factor-authentication-sms-apps/
- https://www.wired.com/2016/06/hey-stop-using-texts-two-factor-authentication/
- https://www.wired.com/story/sim-swap-attack-defend-phone/



Adam Draper

medium.com

$50K stolen by hacking his gmail to send fake invoices

Password-only protection on gmail account

Megan Clifford

time.com/money

$3.5K and a month of her life stolen when her cell phone number and password were hacked

T-Mobile accounts breached in 2015 Experian hack

Small Businesses

businessnewsdaily.com

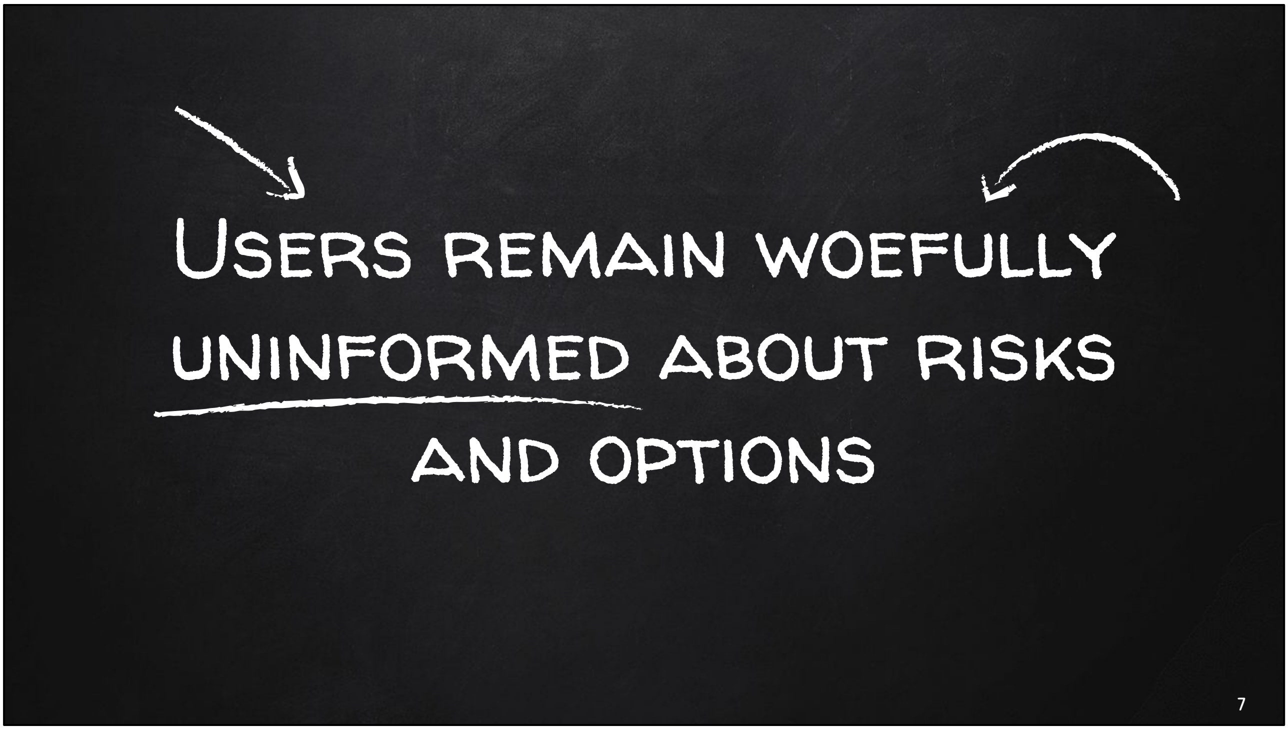
$1 billion stolen in 2011 via compromised usernames and passwords

70% small businesses go bankrupt after being hacked (Symantec)

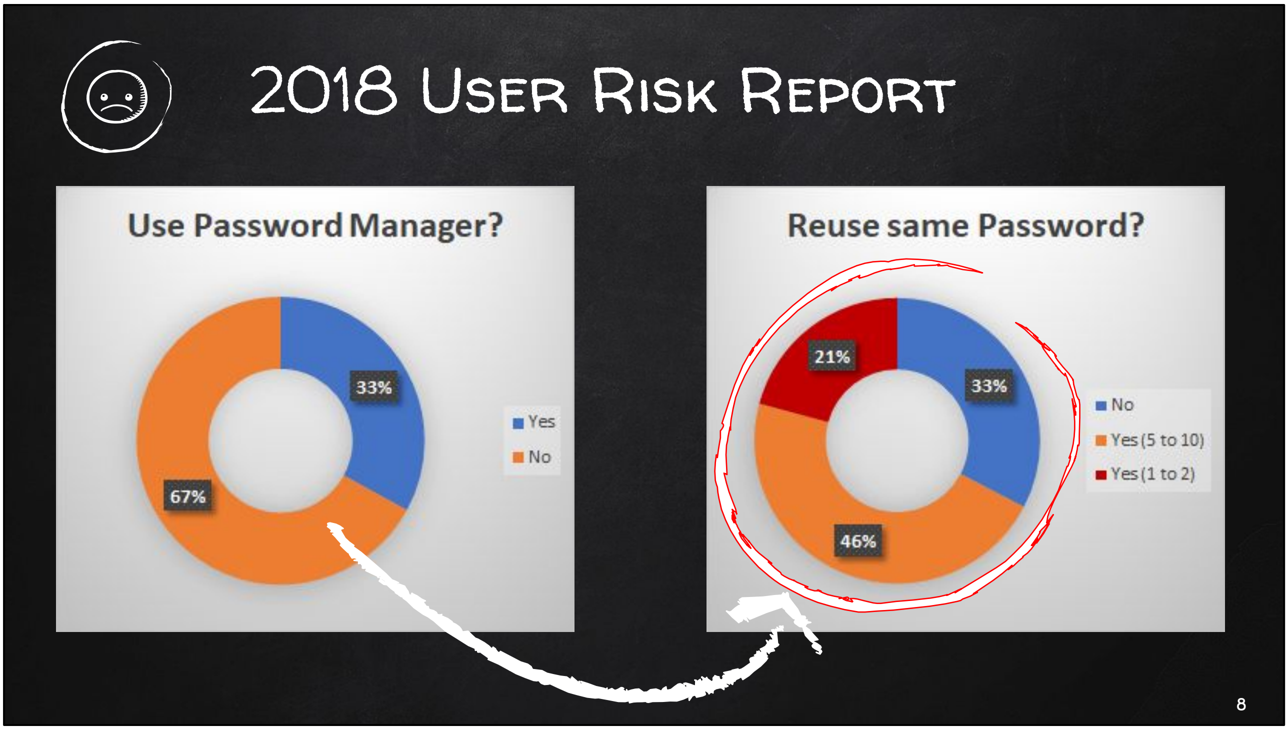
Adam Draper
https://medium.com/@adamdraper/a-hacker-stole-50k-from-my-bank-account-388822389671

Megan Clifford
http://time.com/money/5245878/cell-phone-porting-scam-t-mobile/

Small Businesses
https://www.businessnewsdaily.com/5855-why-your-bank-account-might-not-be-as-safe-as-you-think.html



Users remain woefully uninformed about risks and options



2018 User Risk Report

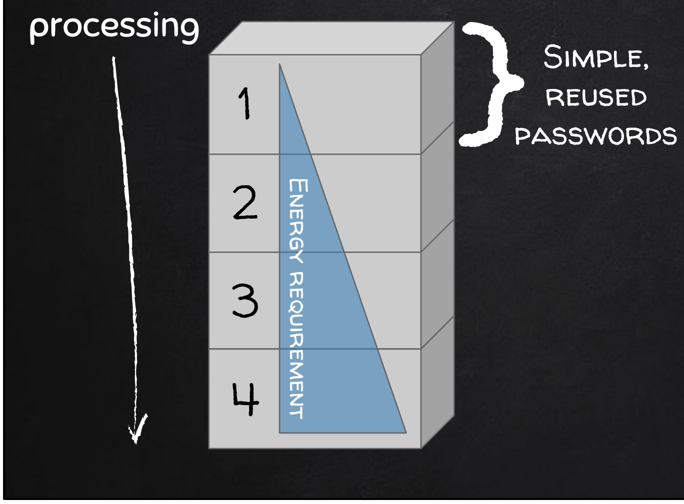
2018 User Risk Report performed by Wombat Security

An international cybersecurity awareness survey of 6000+ working adults with mean age 45 in 6 countries (USA, Europe, Australia)

https://www.wombatsecurity.com/user-risk-report

Users are biologically wired to make predictable password choices

Top-down Processing



Human brain attempts to conserve energy by reverting to known patterns

Dr. Chantal Hofstee

"Mindfulness on the Run: Quick, effective mindfulness techniques for busy people"

2016

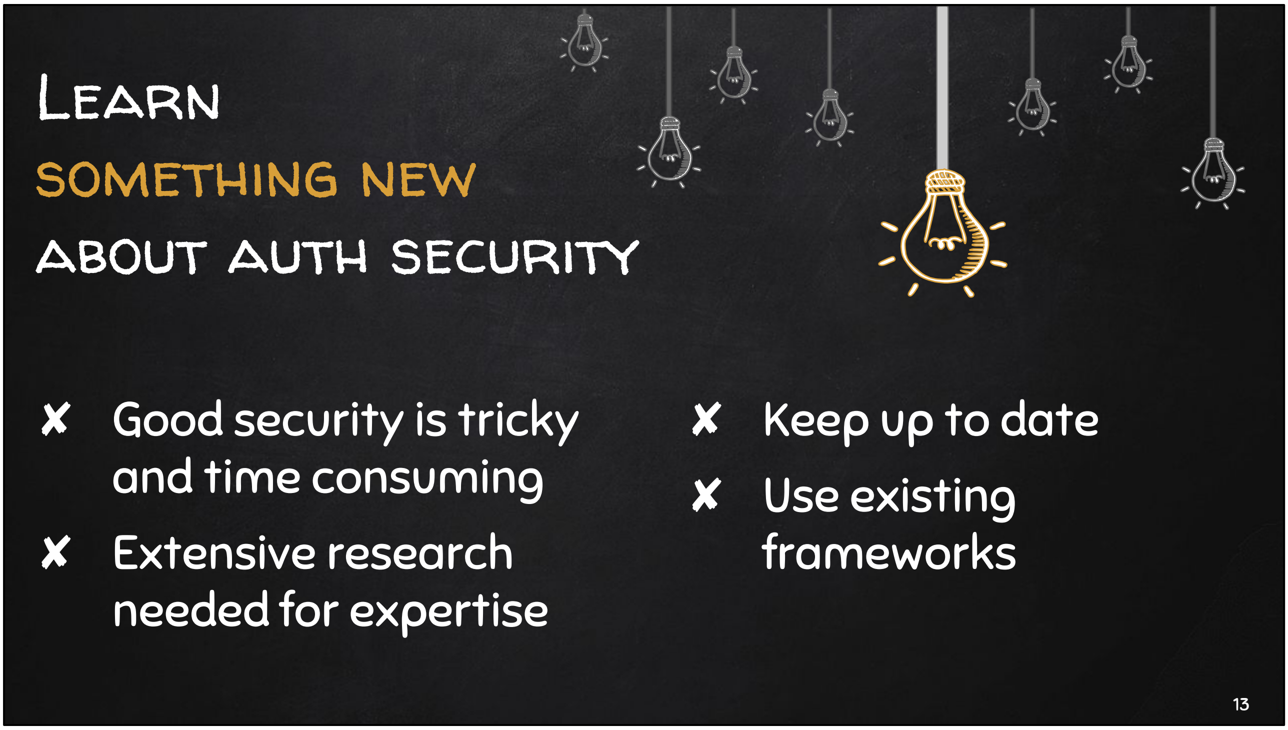
Additional protection exists but is not offered often enough

https://twofactorauth.org/#banking

Make it challenging for malicious parties

Honour trust placed in us as developers, by implementing protections to secure user data, and to limit damage when breached

Hackers are people (until the robot overlords take over) with limited resources



Learn
something new
about auth security

✗ Good security is tricky and time consuming

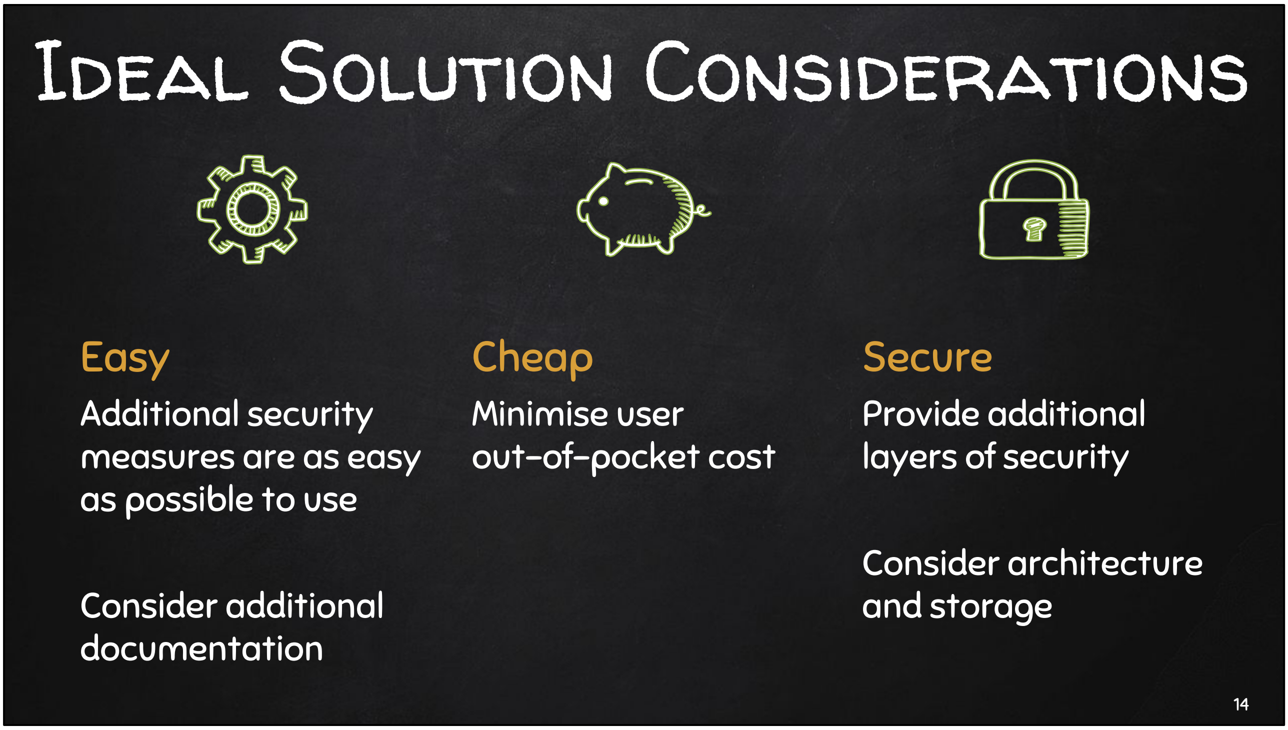
✗ Extensive research needed for expertise

✗ Keep up to date

✗ Use existing frameworks

Primary RFC’s for this talk:

- HOTP 4226 https://tools.ietf.org/html/rfc4226
- TOTP 6238 https://tools.ietf.org/html/rfc6238
- HMAC 2104 https://tools.ietf.org/html/rfc2104
- Randomness for Security 4086 https://tools.ietf.org/html/rfc4086
- BaseXX Data Encodings 4648 https://tools.ietf.org/html/rfc4648



Ideal Solution Considerations

Easy

Additional security measures are as easy as possible to use

Consider additional documentation

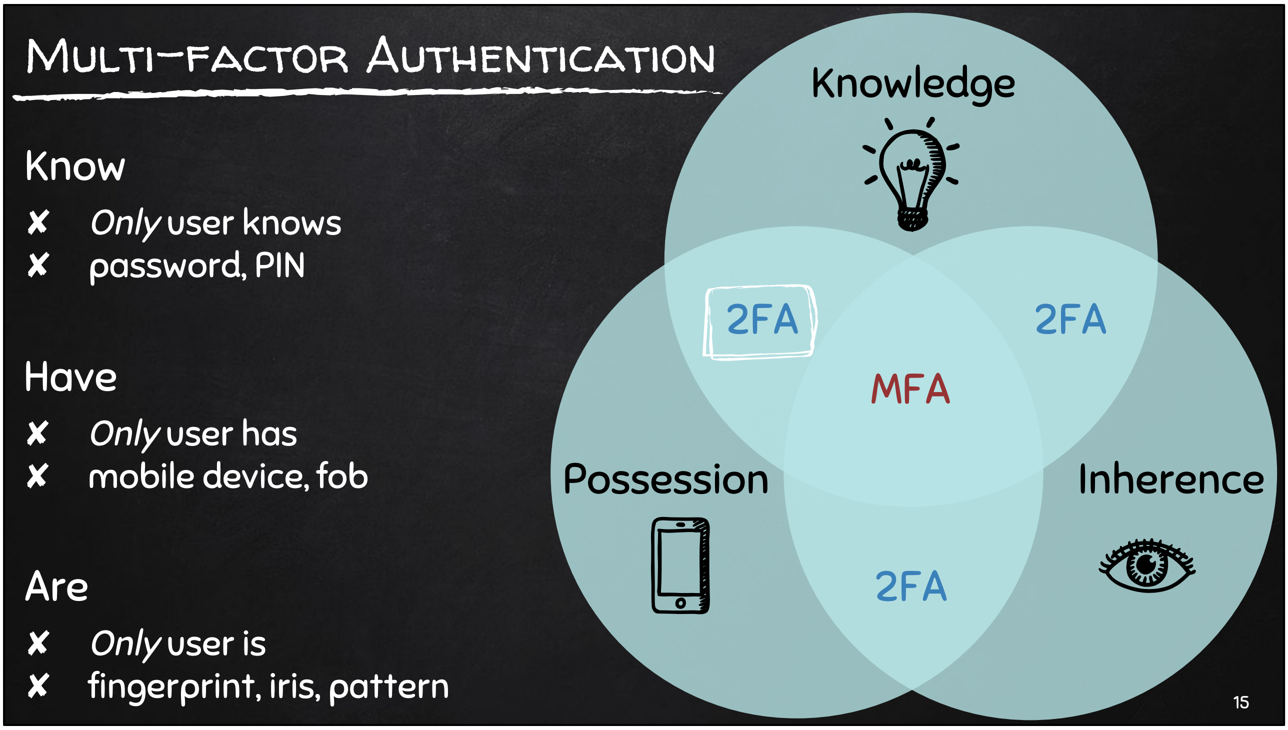
Cheap

Minimise user out-of-pocket cost

Secure

Provide additional layers of security

Consider architecture and storage



Multi-factor Authentication

Know

- ✘ *Only* user knows
- ✘ password, PIN

Have

- ✘ *Only* user has
- ✘ mobile device, fob

Are

- ✘ *Only* user is
- ✘ fingerprint, iris, pattern

Knowledge

2FA

2FA

MFA

Possession

Inherence

2FA

Multi-factor authentication (MFA) is a security mechanism in which a user is authenticated through more than one required security and validation procedure. It is a combination of 2 or more factors:

- knowledge (something **only** the user knows) - password
- possession (something **only** the user has) - device
- inherence (something **only** the user is) - fingerprint

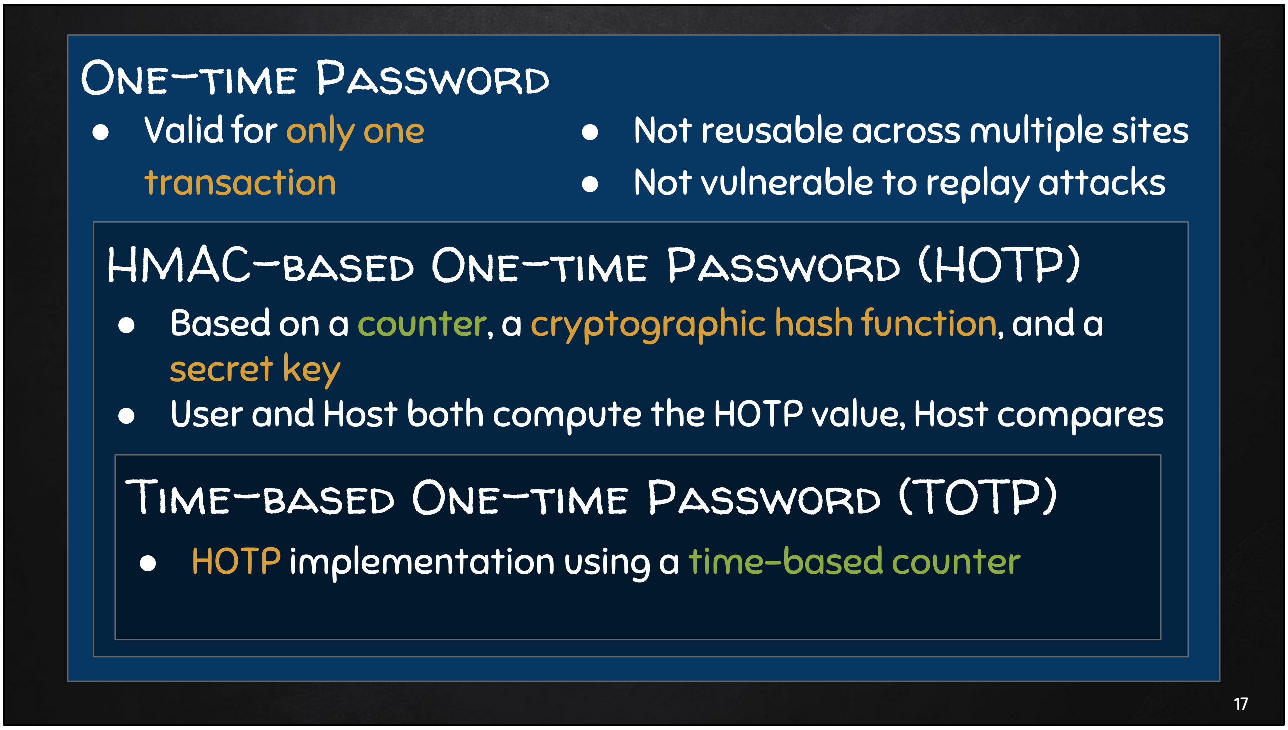
Two-factor authentication (2FA) is a subset of MFA although the 2 terms are often used interchangeably. 2FA requires only 2 of the above factors, and most often is a combination of knowledge and possession. We'll be focusing on a **possession 2nd factor**

Multi-factor is not the same as multi-step. Multi-step typically remains within the category of **Knowledge**:

- Security question
- reCAPTCHA
- SMS verification

Provide a 2nd possession factor using a mobile device (or hardware key) that only the user has

Mechanism is a time-based one-time password (TOTP) software token



One-time Password

- Valid for only one transaction
- Not reusable across multiple sites
- Not vulnerable to replay attacks

HMAC-based One-time Password (HOTP)

- Based on a counter, a cryptographic hash function, and a secret key
- User and Host both compute the HOTP value, Host compares

Time-based One-time Password (TOTP)

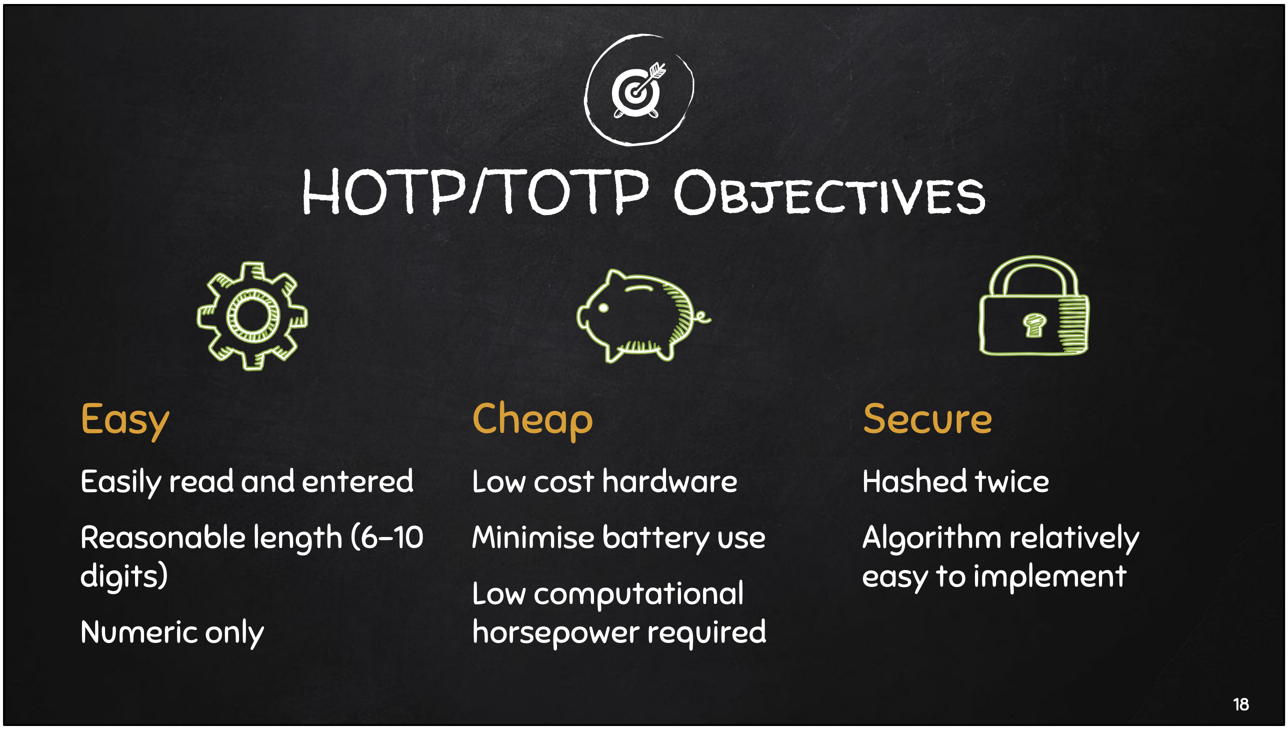
- HOTP implementation using a time-based counter

Possession factor generates a specific type of one-time password that is unique to the device

- HOTP = IETF RFC 4226 (December 2005)
- HMAC (hash-based message authentication code) = a specific type of message authentication code (MAC) involving a cryptographic hash function and a secret cryptographic key.
- TOTP = IETF RFC 6238 (May 2011)

References:

- https://en.wikipedia.org/wiki/One-time_password
- https://en.wikipedia.org/wiki/HMAC
- https://en.wikipedia.org/wiki/HMAC-based_One-time_Password_algorithm
- https://tools.ietf.org/html/rfc4226
- https://en.wikipedia.org/wiki/Time-based_One-time_Password_algorithm
- https://tools.ietf.org/html/rfc6238



HOTP/TOTP Objectives

Easy

Easily read and entered

Reasonable length (6-10 digits)

Numeric only

Cheap

Low cost hardware

Minimise battery use

Low computational horsepower required

Secure

Hashed twice

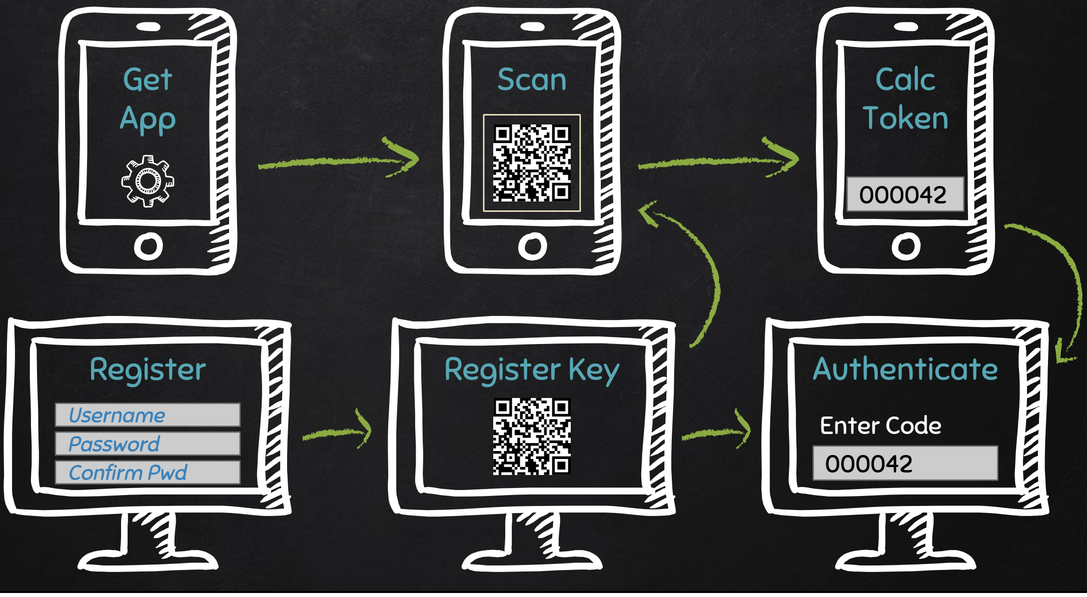
Algorithm relatively easy to implement

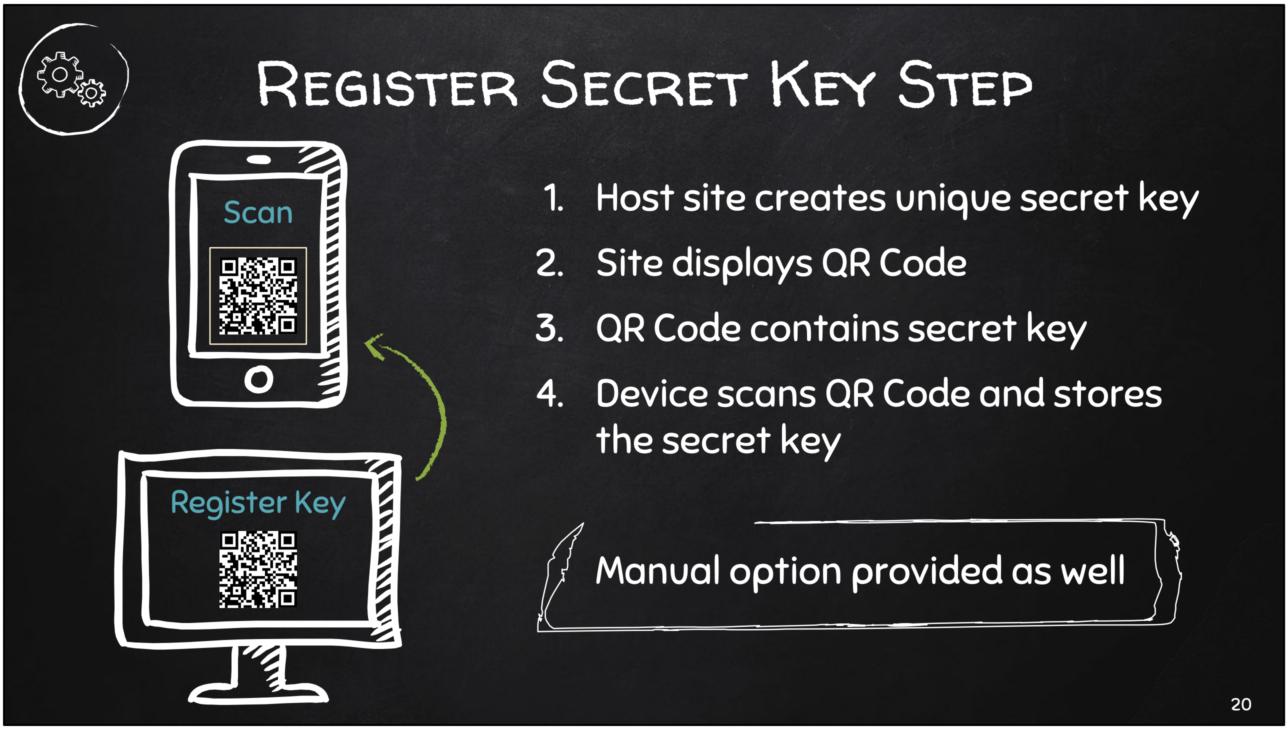
Low cost hardware =

- existing mobile device or inexpensive hardware key/fob
- Minimise battery consumption
- Low computational horsepower

Simple, secure algorithm = relatively straightforward for developers to implement

TOTP Registration Process





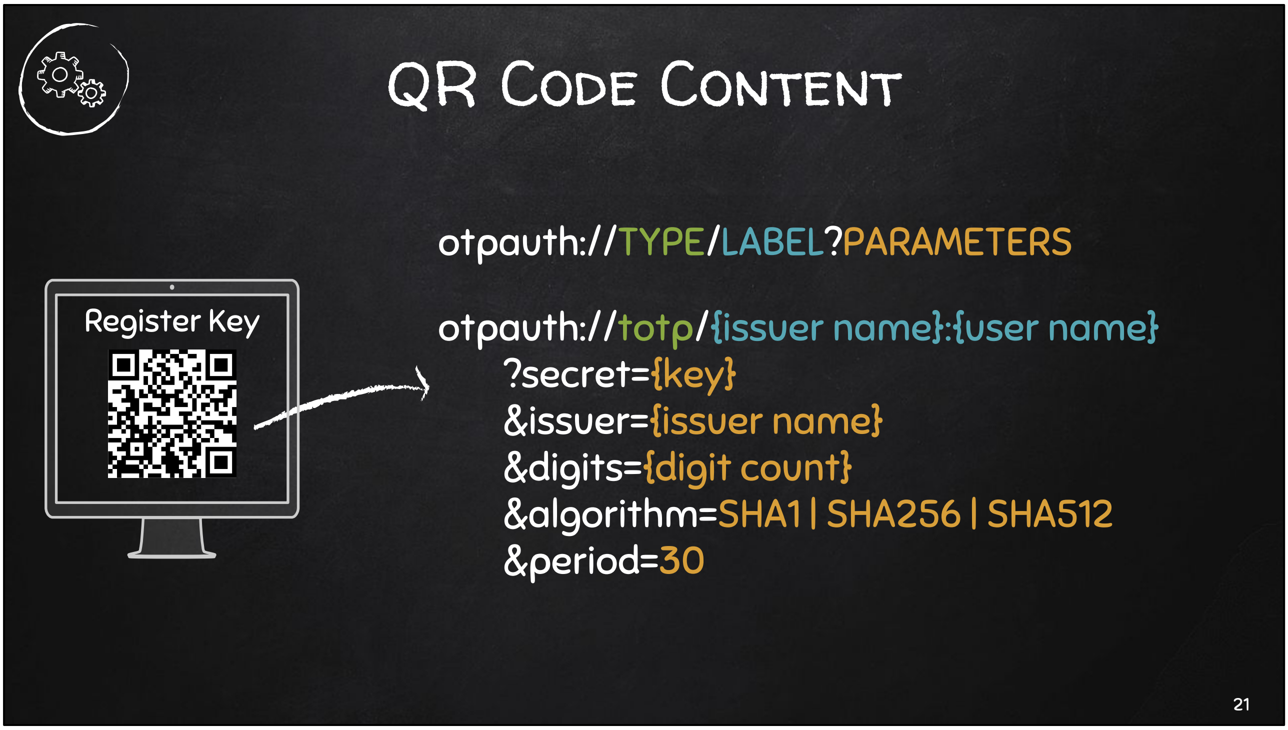
Register Secret Key Step

1. Host site creates unique secret key
2. Site displays QR Code
3. QR Code contains secret key
4. Device scans QR Code and stores the secret key

Manual option provided as well

Manual option for

- devices without a QR reader
- if registration is being performed on a mobile device to begin with



QR Code Content

Register Key

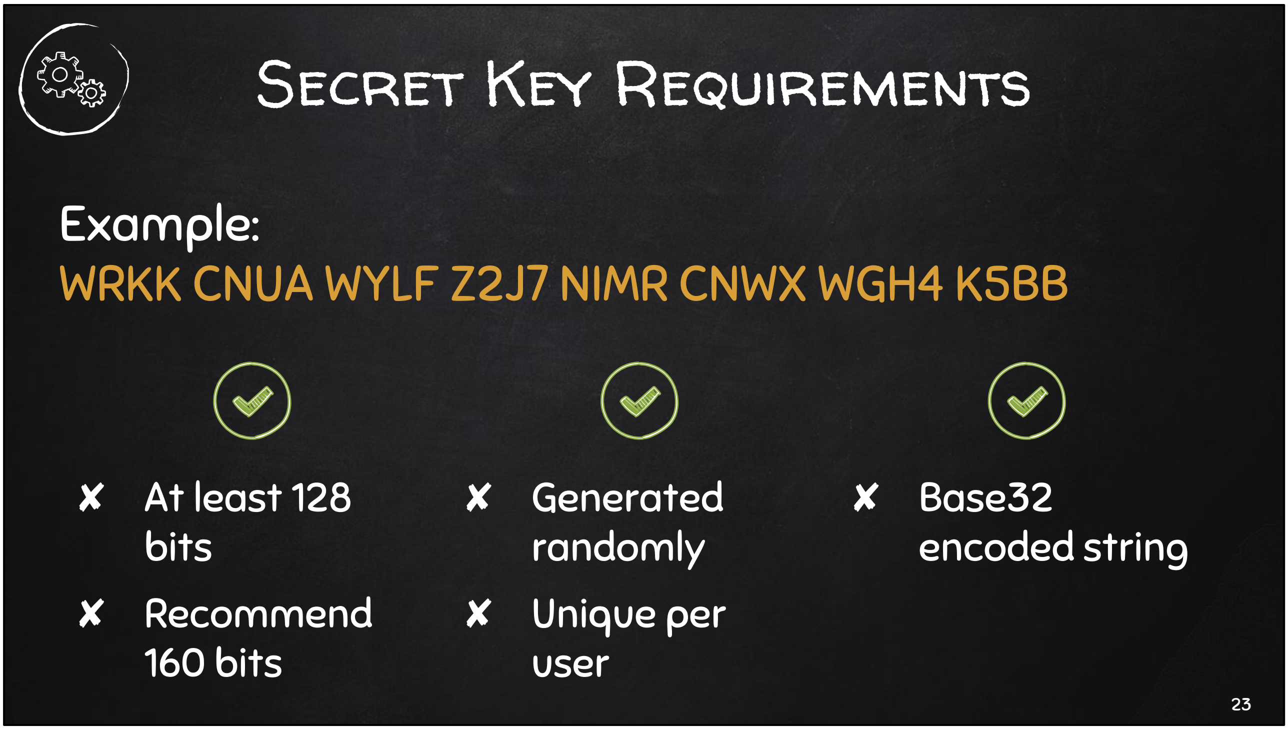
```
otpauth://TYPE/LABEL?PARAMETERS

otpauth://totp/{issuer name}:{user name}
     ?secret={key}
     &issuer={issuer name}
     &digits={digit count}
     &algorithm=SHA1 | SHA256 | SHA512
     &period=30
```

https://github.com/google/google-authenticator/wiki/Key-Uri-Format

QR Code Example

```
otpauth://totp/GitHub:kevin.thomas@equinox.co.nz
      ?secret=WRKKCNUAWYLFZ2J7NIMRCNWXWGH4K5BB
      &issuer=GitHub
      &digits=6
```



Secret Key Requirements

Example:
WRKK CNUA WYLF Z2J7 NIMR CNWX WGH4 K5BB

- ✘ At least 128 bits
- ✘ Recommend 160 bits

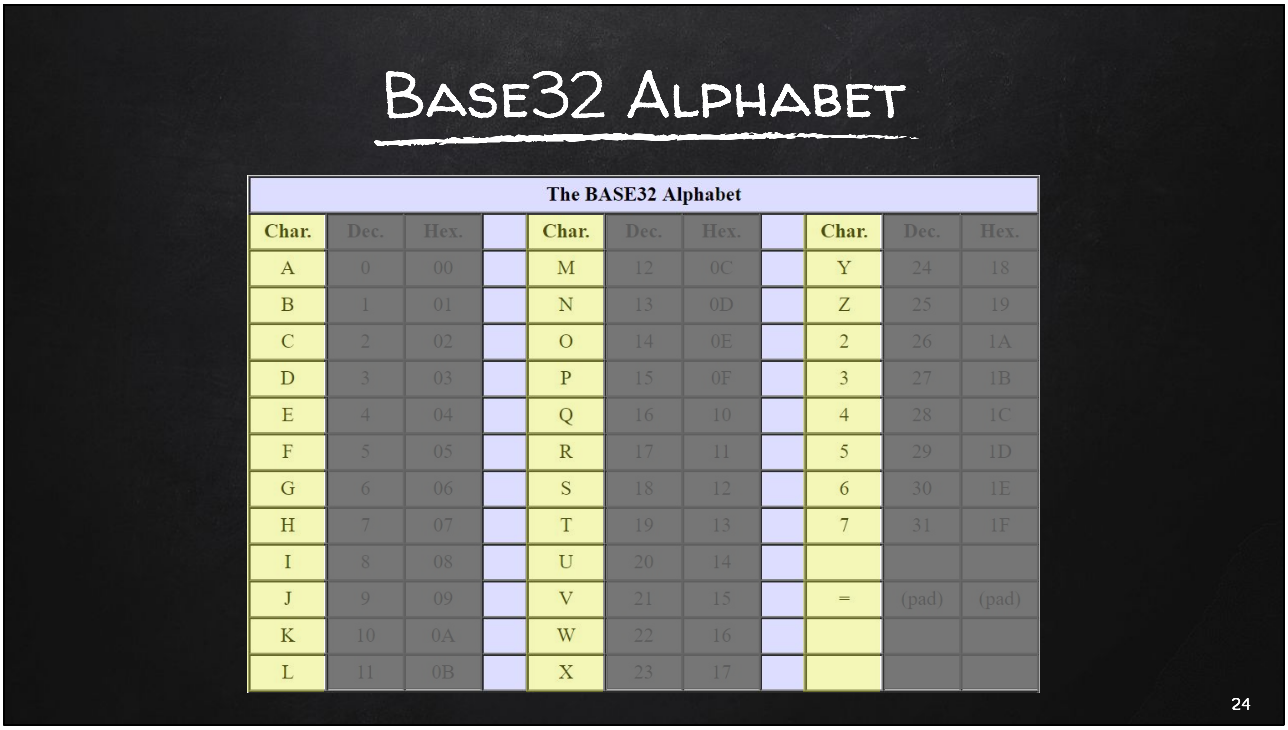
- ✘ Generated randomly
- ✘ Unique per user

- ✘ Base32 encoded string

RFC 4226 = HOTP
https://tools.ietf.org/html/rfc4226

RFC 4086 = Randomness requirements for security
https://tools.ietf.org/html/rfc4086

RFC 4648 = The Base16, Base32, and Base64 Data Encodings
https://tools.ietf.org/html/rfc4648



Base32 Alphabet

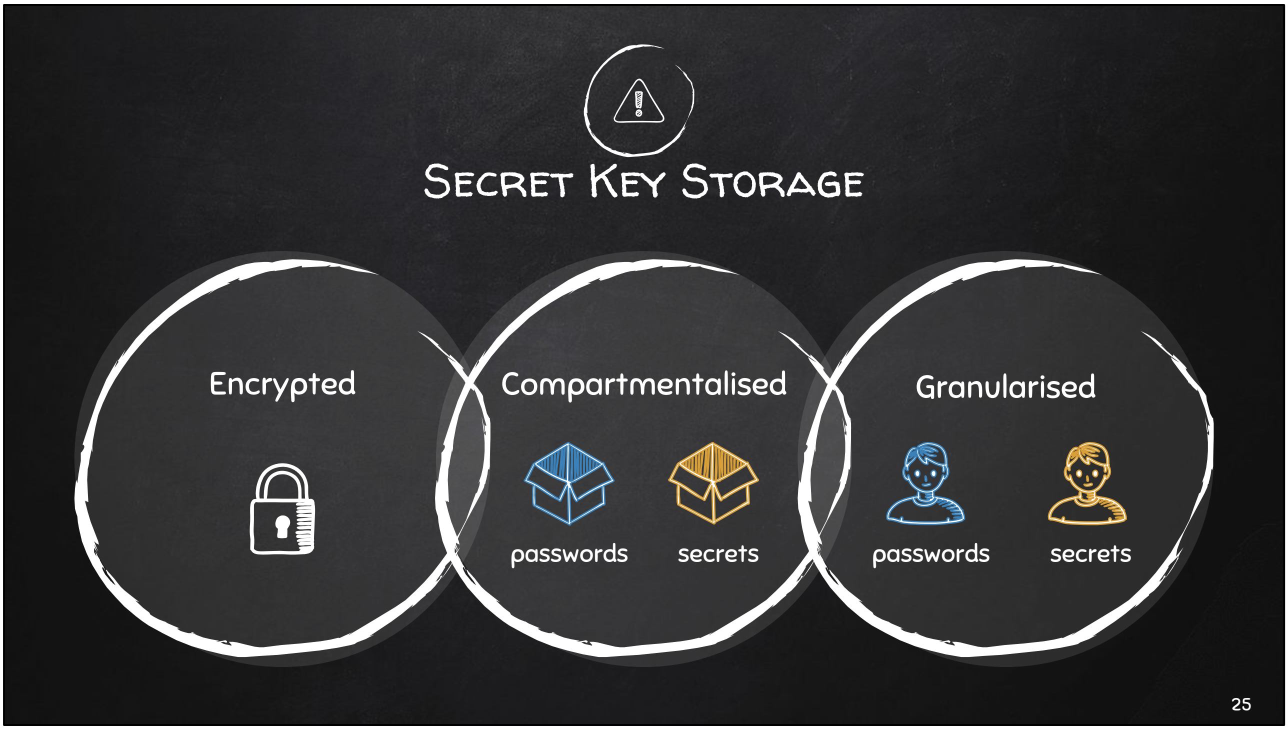
The BASE32 Alphabet

Char.	Dec.	Hex.	Char.	Dec.	Hex.	Char.	Dec.	Hex.
A	0	00	M	12	0C	Y	24	18
B	1	01	N	13	0D	Z	25	19
C	2	02	O	14	0E	2	26	1A
D	3	03	P	15	0F	3	27	1B
E	4	04	Q	16	10	4	28	1C
F	5	05	R	17	11	5	29	1D
G	6	06	S	18	12	6	30	1E
H	7	07	T	19	13	7	31	1F
I	8	08	U	20	14			
J	9	09	V	21	15	=	(pad)	(pad)
K	10	0A	W	22	16			
L	11	0B	X	23	17			

Base 32:

- Characters are all upper-case
- Similar-looking symbols are omitted (1, 8, 9, 0)
- Useful for the scenario where the secret key is manually entered into the possession factor device
- https://en.wikipedia.org/wiki/Base32
- https://tools.ietf.org/html/rfc4648

Image from: https://www.garykessler.net/library/base64.html



Secret Key Storage

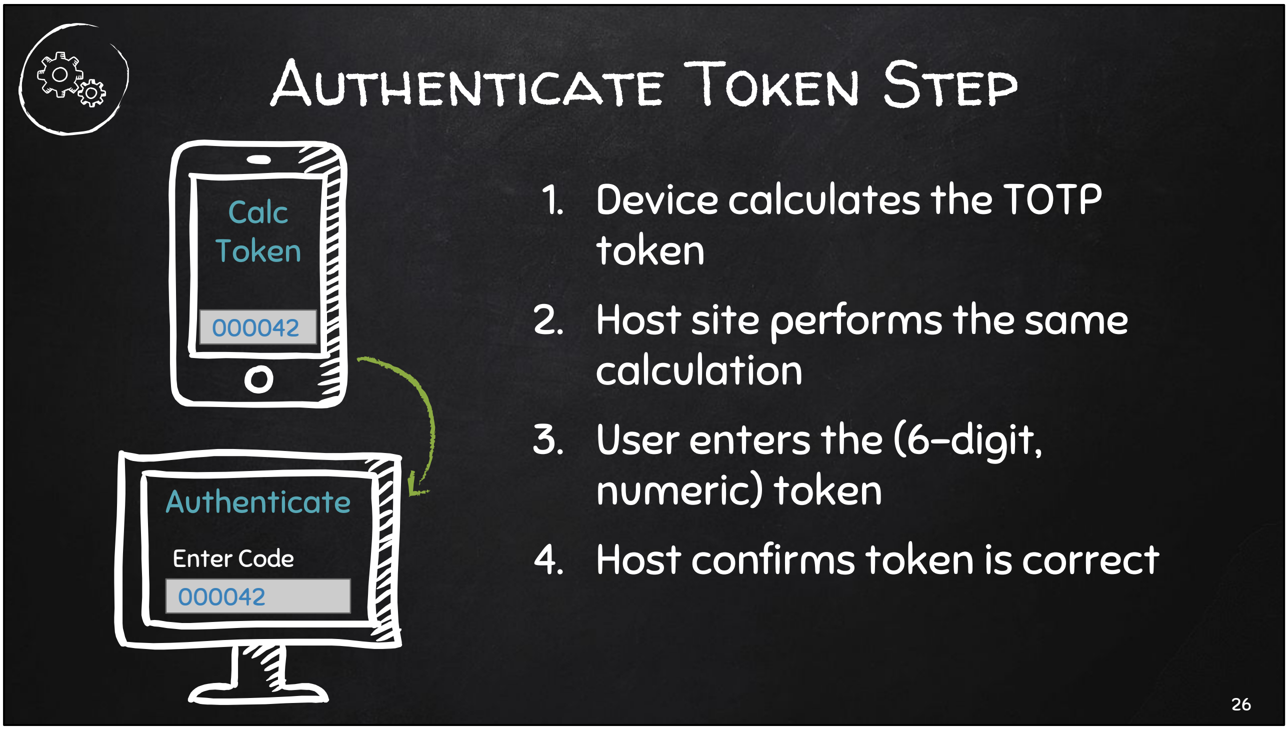
TOTP (RFC 6238): We also RECOMMEND storing the keys securely in the validation system, and, more specifically, encrypting them using tamper-resistant hardware encryption and exposing them only when required

Encrypted: e.g. using Rijndael/AES symmetric key algorithm with Cipher Block Chaining (CBC) mode.

- https://en.wikipedia.org/wiki/Advanced_Encryption_Standard
- https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation#Cipher_Block_Chaining_.28CBC.29
- Must use **encryption** rather than **hashing** because secret key must be available in plain text

TOTP (RFC 6238): The key store MUST be in a secure area, to avoid, as much as possible, direct attack on the validation system and secrets database. Particularly, access to the key material should be limited to programs and processes required by the validation system only.

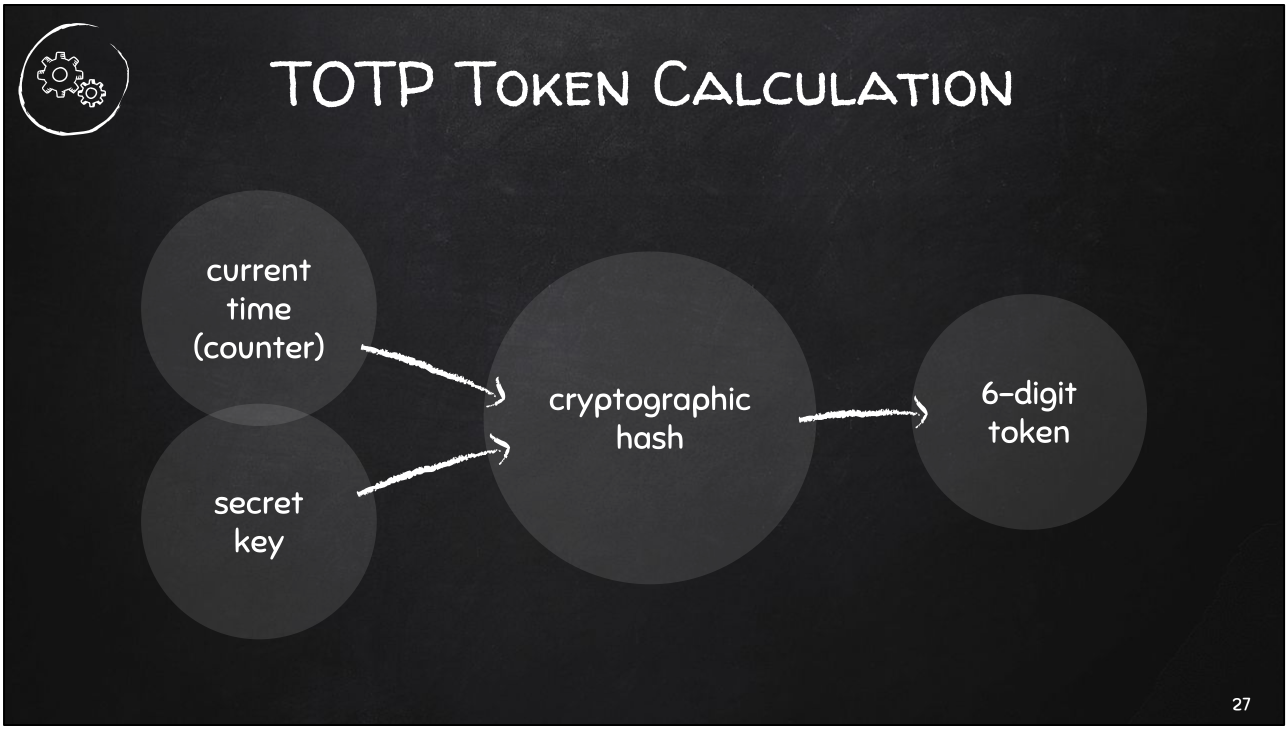
Compartmentalised: consider storing passwords and TOTP/HOTP secret keys in different locations (e.g., different databases)
Granularised: consider granting different roles/accounts to access passwords and secret keys (this is the default approach used by Microsoft ASP.NET Core applications - UserName/Password stored in ***AspNetUsers*** table, secrete keys stored in ***AspNetUserTokens*** table (same database); can grant permissions on these 2 tables to different users



Authenticate Token Step

Calc
Token
000042
Authenticate
Enter Code
000042

1. Device calculates the TOTP token
2. Host site performs the same calculation
3. User enters the (6-digit, numeric) token
4. Host confirms token is correct



TOTP Token Calculation

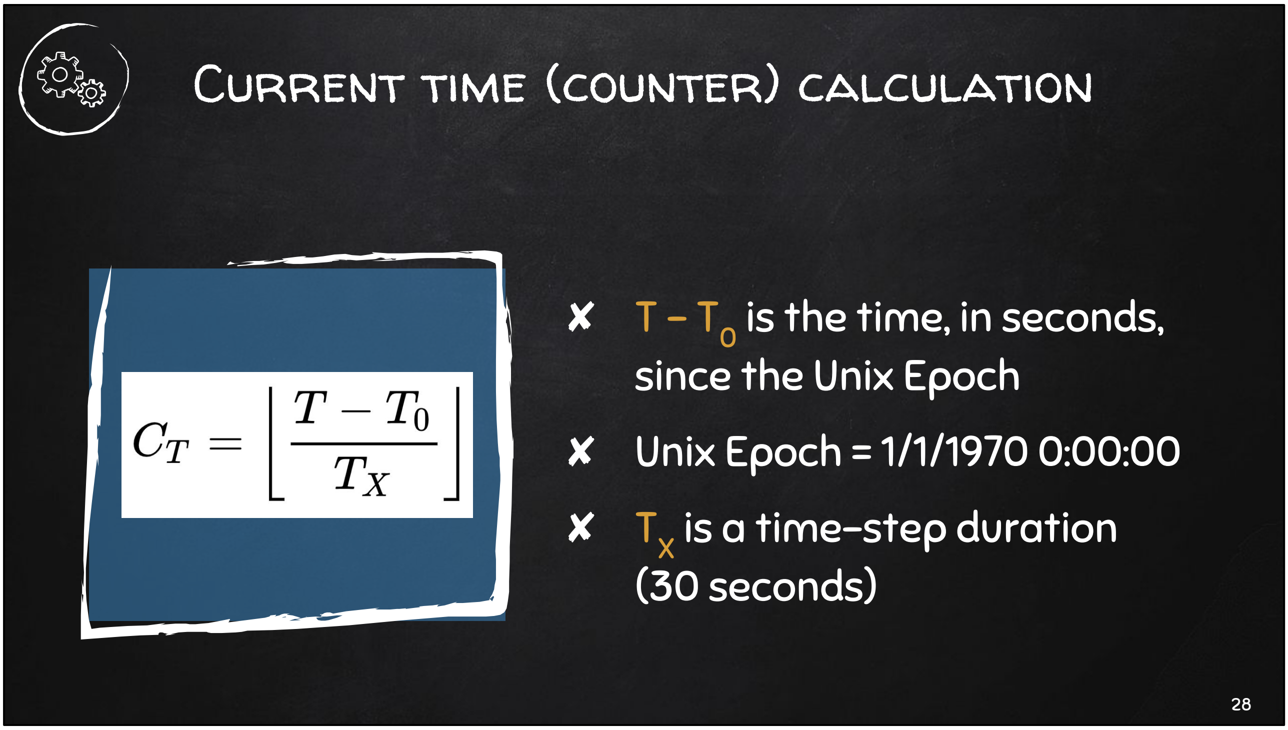
current time (counter)

secret key

cryptographic hash

6-digit token

Same calculation performed on device (possession factor) and on host



Current time (counter) calculation

$$C_T = \left\lfloor \frac{T - T_0}{T_X} \right\rfloor$$

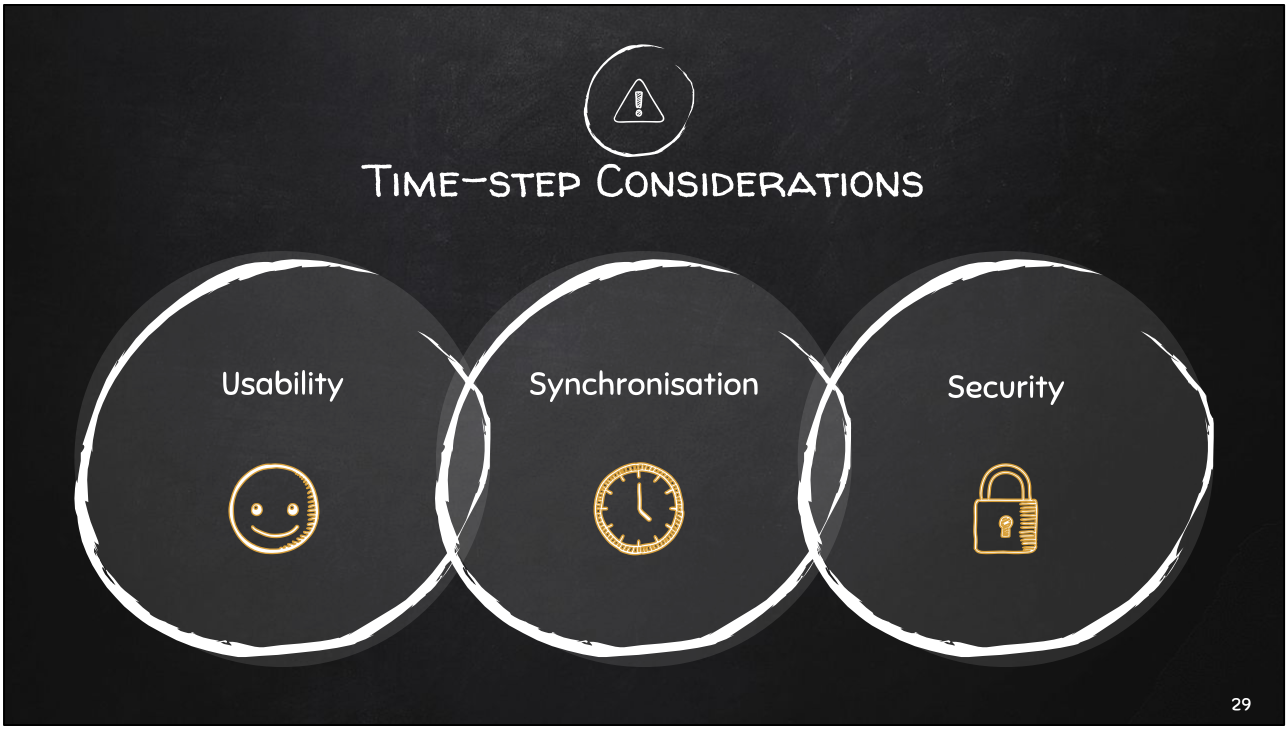
- ✘ $T - T_0$ is the time, in seconds, since the Unix Epoch
- ✘ Unix Epoch = 1/1/1970 0:00:00
- ✘ T_X is a time-step duration (30 seconds)

Epoch converter: https://www.epochconverter.com/

30 seconds is the default time-step, selected as a balance between security and usability

Token value will be unchanged for the duration of the time-step

RFC 6238 https://tools.ietf.org/html/rfc6238



TIME-STEP CONSIDERATIONS

Usability

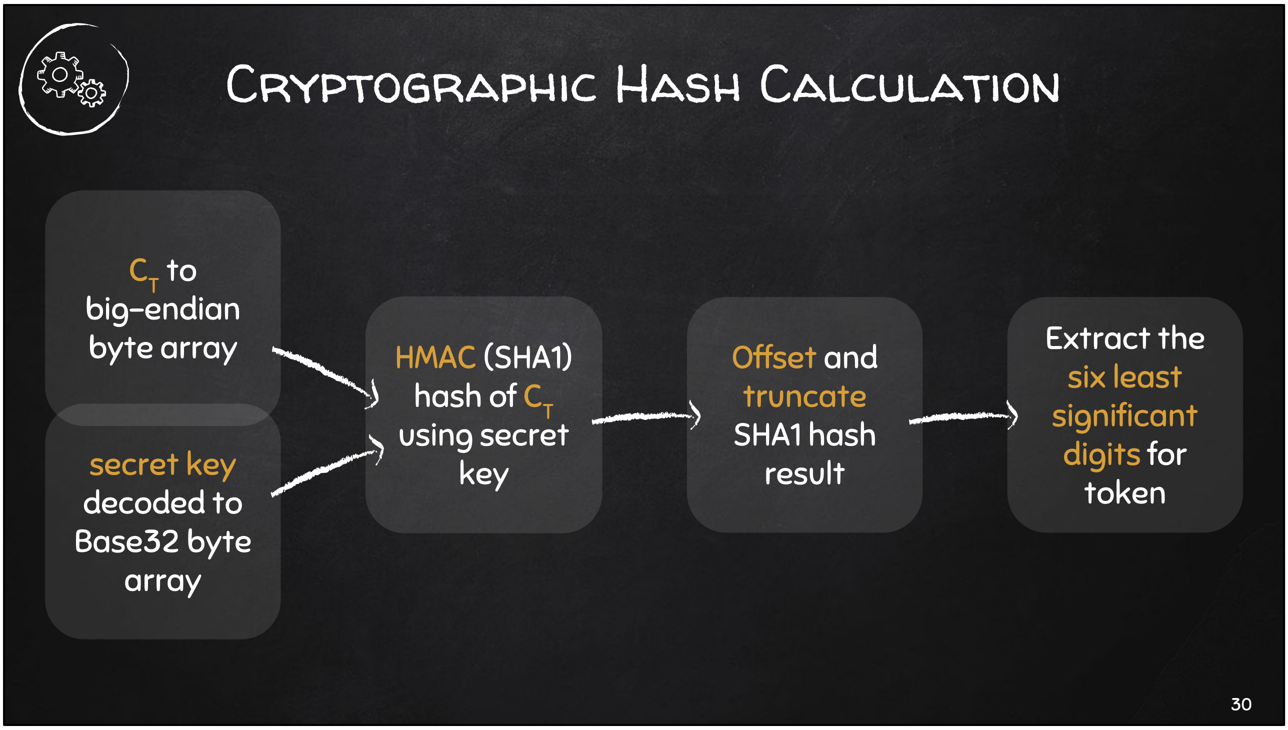
- Need to provide enough time for user to type in 6 digits

Synchronisation

- Account for clock drift, network latency, etc
- Typically allow for several time steps (1 in each direction from current)

Security

- Brute force attack is only viable option for attacker
- Critical to have a throttling parameter and/or lockout scheme



Cryptographic Hash Calculation

C_T to big-endian byte array

secret key decoded to Base32 byte array

HMAC (SHA1) hash of C_T using secret key

Offset and truncate SHA1 hash result

Extract the six least significant digits for token

Current time to **big-endian** byte array

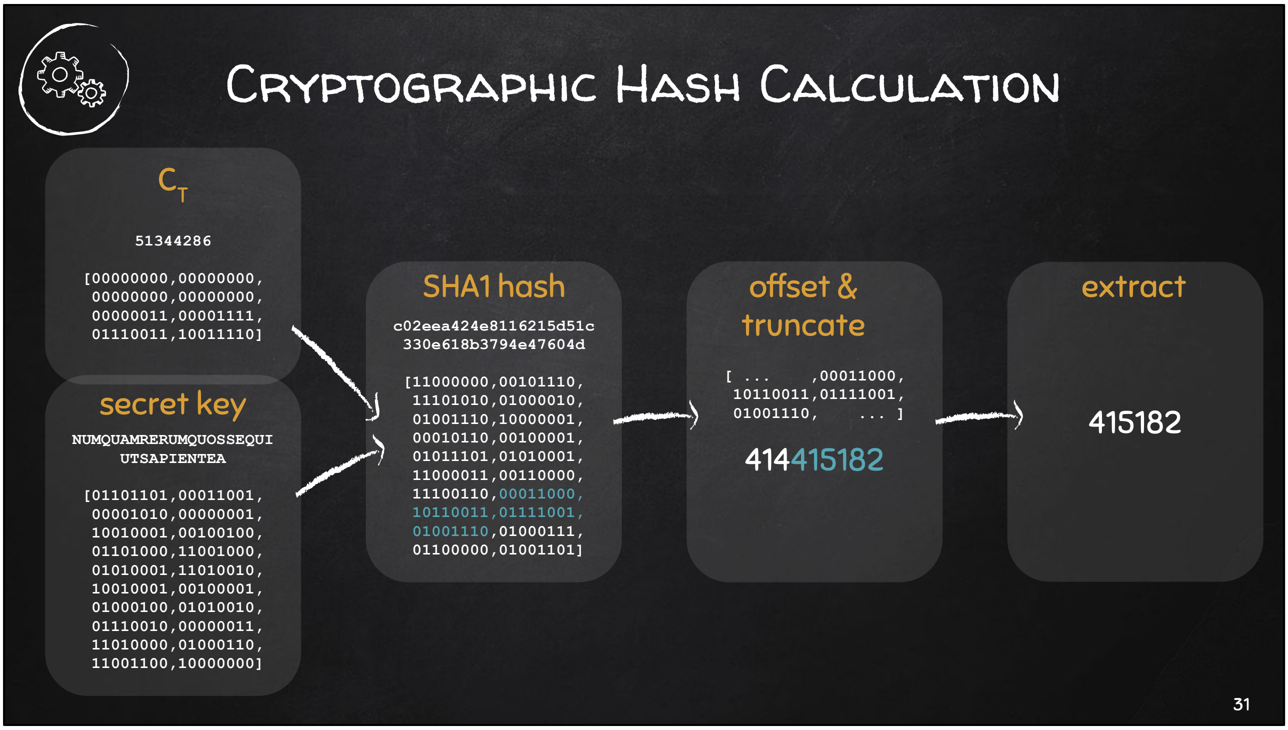
- Big-endian means the most significant byte is stored first (index 0 in byte array)
- Causes sign of the number to be dropped
- Standardised/predictable sequence of bytes
- https://www.mathsisfun.com/binary-decimal-hexadecimal-converter.html

Offset and truncate:

- Essentially performing a 2nd hash that will result in a number
- Sufficient randomness to make it impossible to reverse-engineer

Extract digits:

- Take the 6 least-significant (= right-most) digits
- Left-pad with 0's if necessary



Cryptographic Hash Calculation

C_T

```
51344286

[00000000,00000000,
 00000000,00000000,
 00000011,00001111,
 01110011,10011110]
```

secret key

```
NUMQUAMRERUMQUOSSEQUI
UTSAPIENTEA

[01101101,00011001,
 00001010,00000001,
 10010001,00100100,
 01101000,11001000,
 01010001,11010010,
 10010001,00100001,
 01000100,01010010,
 01110010,00000011,
 11010000,01000110,
 11001100,10000000]
```

SHA1 hash

```
c02eea424e8116215d51c
330e618b3794e47604d

[11000000,00101110,
 11101010,01000010,
 01001110,10000001,
 00010110,00100001,
 01011101,01010001,
 11000011,00110000,
 11100110,00011000,
 10110011,01111001,
 01001110,01000111,
 01100000,01001101]
```

offset & truncate

```
[ ...     ,00011000,
 10110011,01111001,
 01001110,    ... ]
```

414415182

extract

415182

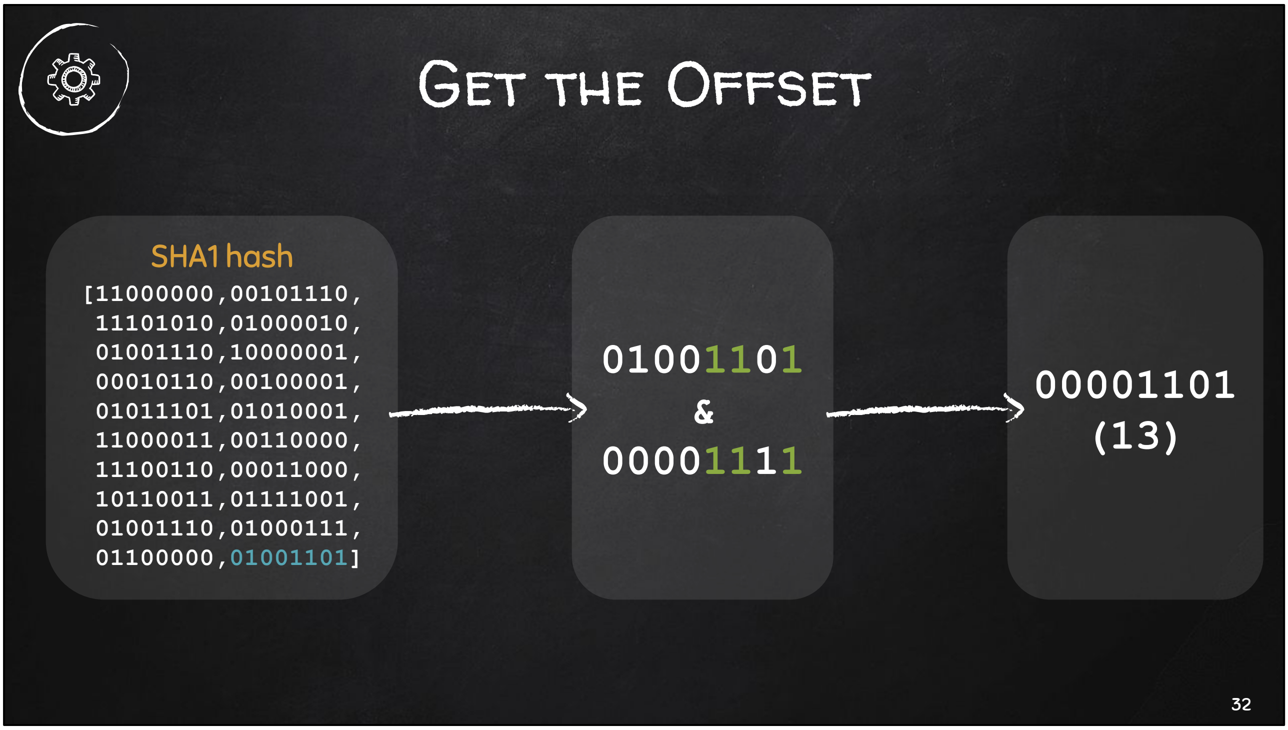
HMAC-SHA-1 hash from RFC 2104
https://tools.ietf.org/html/rfc2104

SHA1 hash explanation:

- https://deadhacker.com/2006/02/21/sha-1-illustrated/
- https://www.cryptocompare.com/coins/guides/how-does-a-hashing-algorithm-work/

HMAC-SHA-256 or HMAC-SHA-512 are both supported by RFC 6238

SHA1 has been broken, however given the multi-hash generation and time duration of the token, SHA1 is still safe for TOTP
https://www.quora.com/Why-is-the-SHA1-algorithm-still-being-used-with-2FA-codes-instead-of-SHA2



Get the Offset

SHA1 hash

```
[11000000,00101110,
 11101010,01000010,
 01001110,10000001,
 00010110,00100001,
 01011101,01010001,
 11000011,00110000,
 11100110,00011000,
 10110011,01111001,
 01001110,01000111,
 01100000,01001101]
```

```
01001101
&
00001111
```

```
00001101
(13)
```

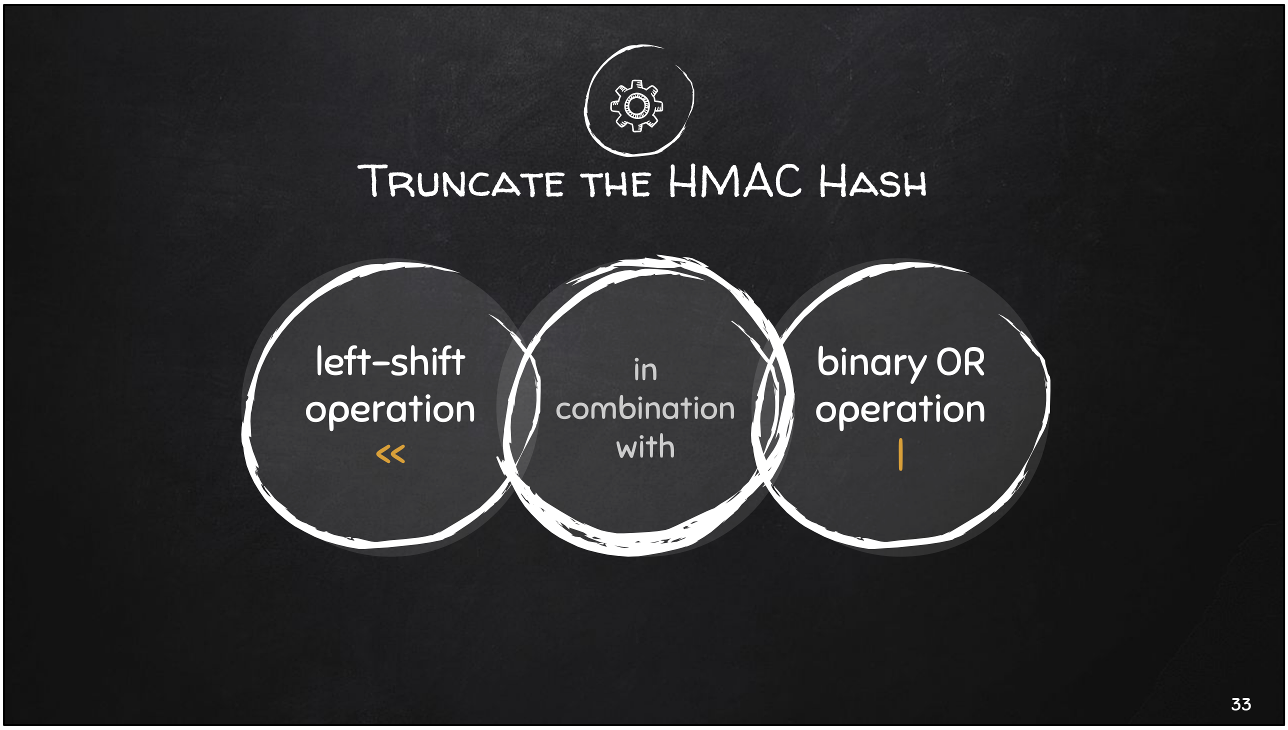
Perform binary AND operation on the least significant byte & 15

The offset is used as a starting point index on the HMAC byte array

The binary AND will always return a value between 0 and 15, which ensures the offset index will always be valid for the HMAC array (length 20)

& (binary AND) operator references:

- https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/and-operator
- https://www.tutorialspoint.com/csharp/csharp_bitwise_operators.htm

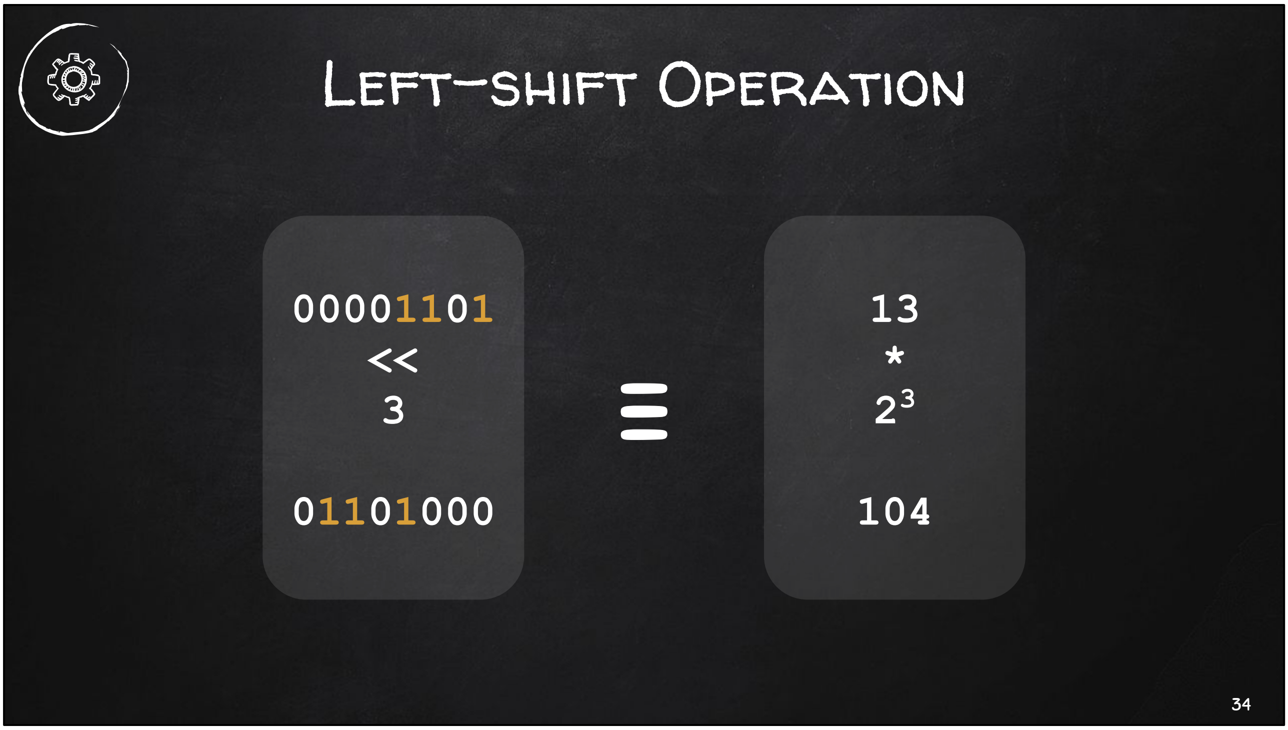


Truncate the HMAC Hash

left-shift operation
<<

in combination with

binary OR operation
|



Left-shift Operation

```
00001101
   <<
    3

01101000
```

≡

```
 13
  *
```

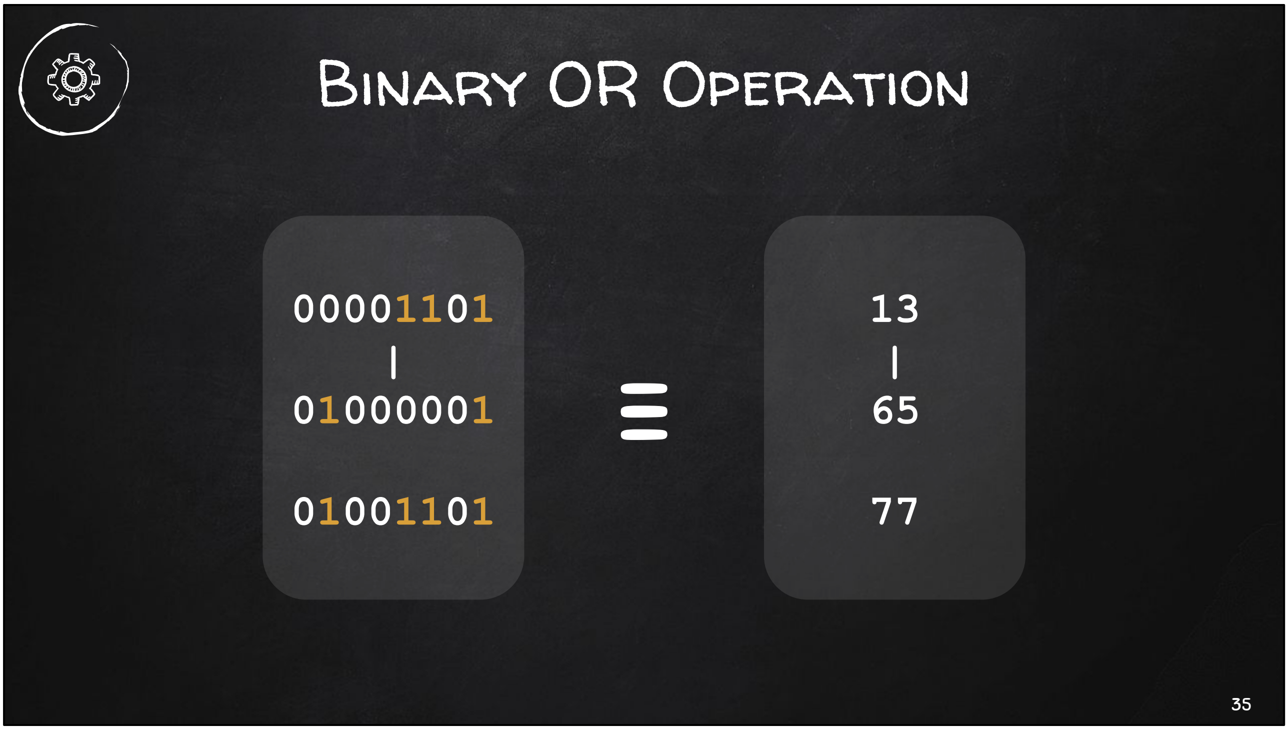
2^3

```
104
```

- Bit **left shift** operation
- Comparable to multiplying by 2 to the nth power
- A good optimizing compiler will replace multiplications with shifts when possible
  (TOTP goal is low computational horsepower)

<< (left shift) operator references:

- https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/left-shift-operator
- https://stackoverflow.com/questions/141525/what-are-bitwise-shift-bit-shift-operators-and-how-do-they-work



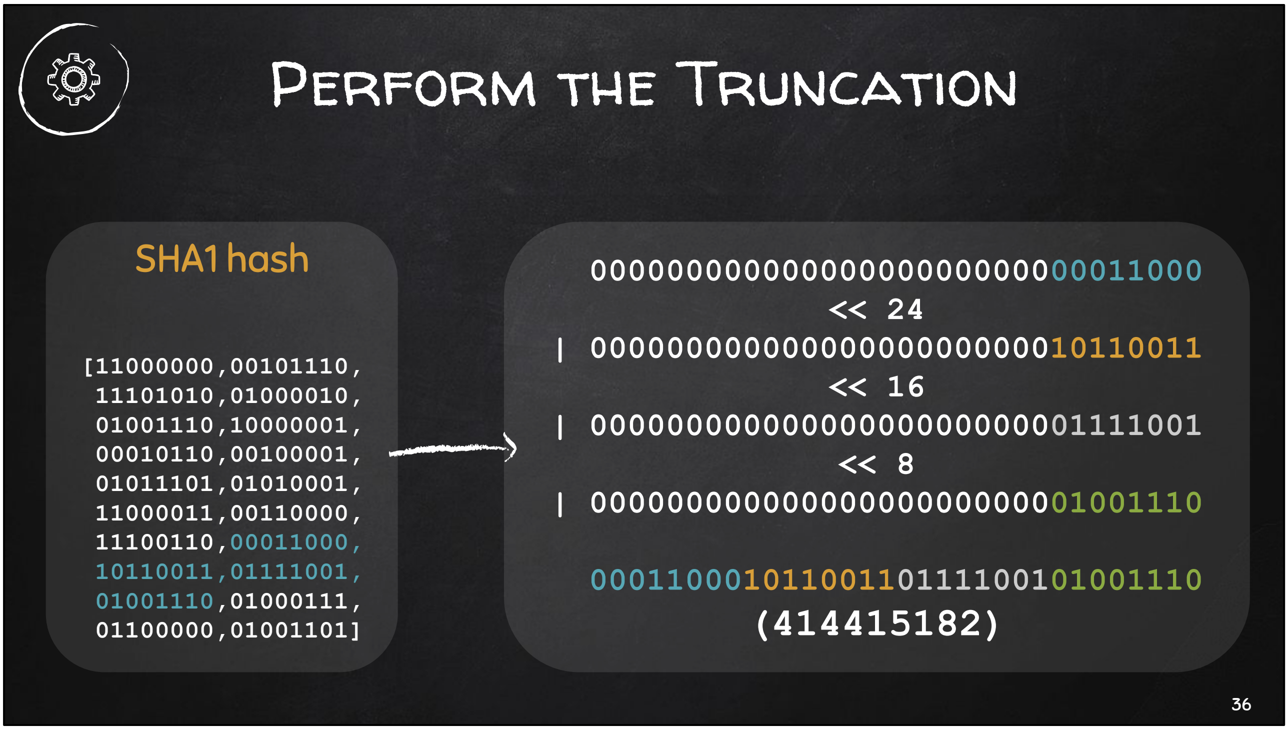
Binary OR Operation

```
00001101        13
    |            |
01000001   =    65

01001101        77
```

| (binary OR) operator references:

- https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/or-operator
- https://www.tutorialspoint.com/csharp/csharp_bitwise_operators.htm



Perform the Truncation

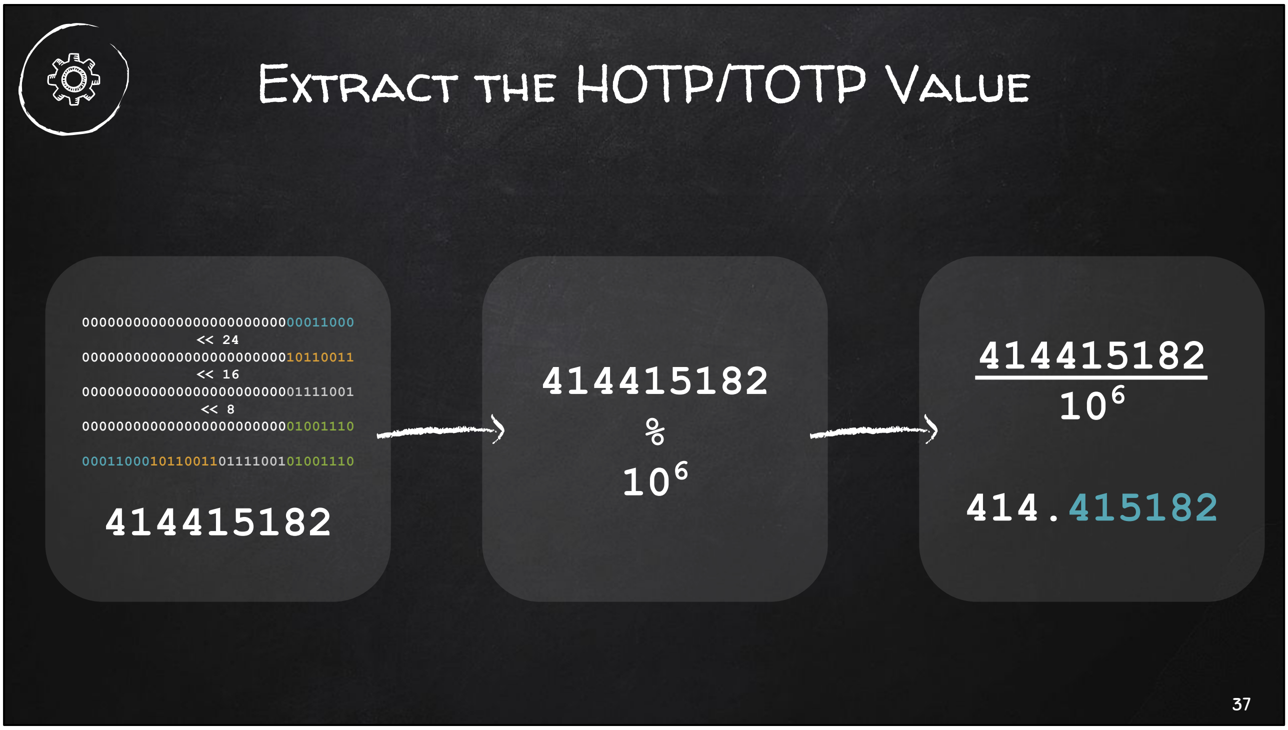
SHA1 hash

```
[11000000,00101110,
 11101010,01000010,
 01001110,10000001,
 00010110,00100001,
 01011101,01010001,
 11000011,00110000,
 11100110,00011000,
 10110011,01111001,
 01001110,01000111,
 01100000,01001101]
```

```
  00000000000000000000000000011000
               << 24
| 00000000000000000000000010110011
               << 16
| 00000000000000000000000001111001
                << 8
| 00000000000000000000000001001110

  00011000101100110111100101001110
            (414415182)
```

1. Take the 4 least significant bytes starting at the offset (index position 13).
2. Perform left-shift and binary OR operations to combine into a single number.



Extract the HOTP/TOTP Value

```
00000000000000000000000000011000
              << 24
00000000000000000000000010110011
              << 16
00000000000000000000000001111001
              << 8
00000000000000000000000001001110

00011000101100110111100101001110
```

414415182

$$414415182 \; \% \; 10^6$$

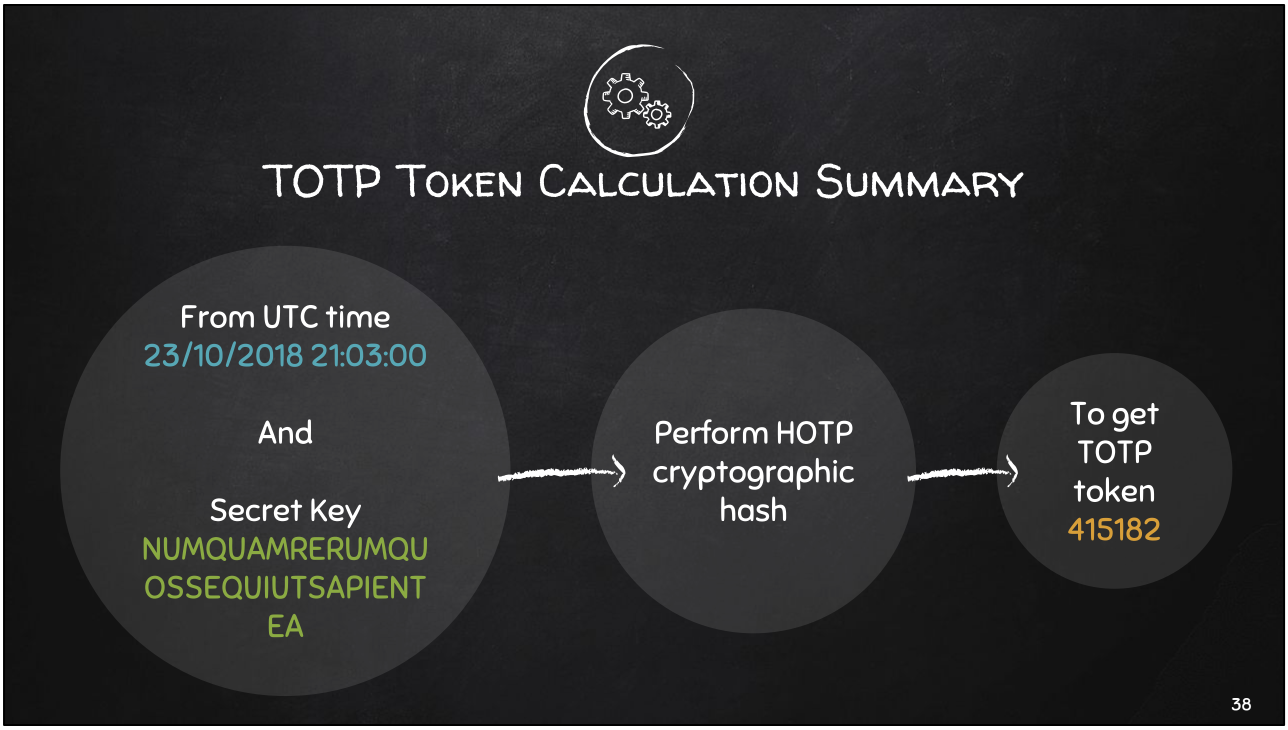
$$\frac{414415182}{10^6}$$

414.415182

1. Perform a modulo operation on the truncated value
2. Truncated value modulo 10 to the x power, where x is the desired number of digits
3. Retain any leading zeros for the final TOTP/HOTP value.

% (modulo) operator references:

- https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/remainder-operator
- https://www.computerhope.com/jargon/m/modulo.htm
- https://www.khanacademy.org/computing/computer-science/cryptography/modarithmetic/a/what-is-modular-arithmetic



TOTP Token Calculation Summary

From UTC time
23/10/2018 21:03:00

And

Secret Key
NUMQUAMRERUMQUOSSEQUIUTSAPIENTEA

Perform HOTP cryptographic hash

To get TOTP token
415182

Cryptographic Hash

1. **SHA1** hash of **time-counter** using **secret key** generates byte array
2. Get offset from **least significant byte** using **binary AND**
3. Generate a single random number by truncating the hash using **left shift** combined with **binary OR**
4. Perform **modulo** operation on that number to get the final 6-digit token

Additional TOTP/HOTP Considerations

TOTP (RFC 6238): All the communications SHOULD take place over a secure channel, e.g., Secure Socket Layer/Transport Layer Security (SSL/TLS) [RFC5246] or IPsec connections [RFC4301].

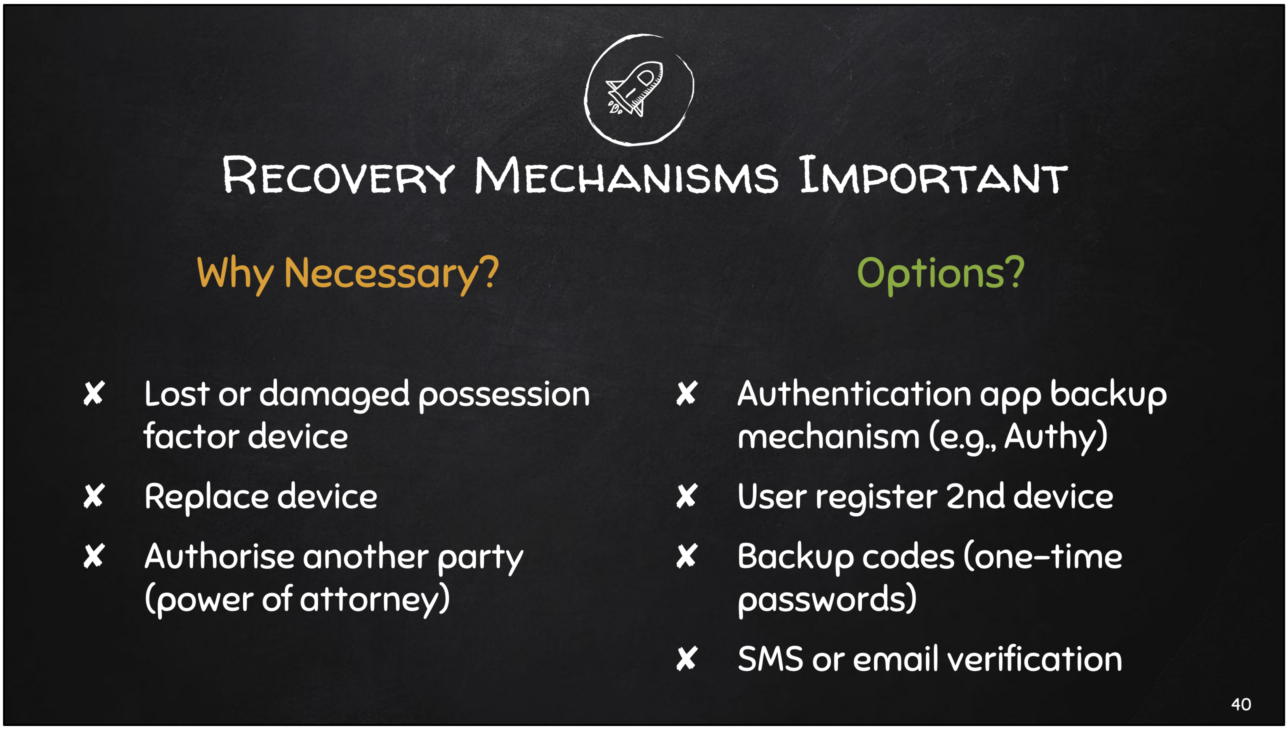
HOTP (RFC 4226): RP1 - verifier MUST support two-factor authentication. The secret code is known only to the user and usually entered with the One-Time Password value for authentication purpose.

HOTP (RFC 4226): RP2 - verifier SHOULD NOT be vulnerable to brute force attacks. This implies that a throttling/lockout scheme is RECOMMENDED on the validation server site.

TOTP (RFC 6238): Because of possible clock drifts between a client and a validation server, we RECOMMEND that the validator be set with a specific limit to the number of time steps a prover can be "out of synch" before being rejected. This limit can be set both forward and backward from the calculated time step on receipt of the OTP value. If the time step is 30 seconds as recommended, and the validator is set to only accept two time steps backward, then the maximum elapsed time drift would be around 89 seconds, i.e., 29 seconds in the calculated time step and 60 seconds for two backward time steps.

TOTP (RFC 6238): R7: The keys MAY be stored in a tamper-resistant device and SHOULD be protected against unauthorized access and usage.

Consider a “trust this computer” for better usability, with user choice on lesser security. Recommend good instructions to accompany.



Recovery Mechanisms Important

Why Necessary?

✘ Lost or damaged possession factor device

✘ Replace device

✘ Authorise another party (power of attorney)

Options?

✘ Authentication app backup mechanism (e.g., Authy)

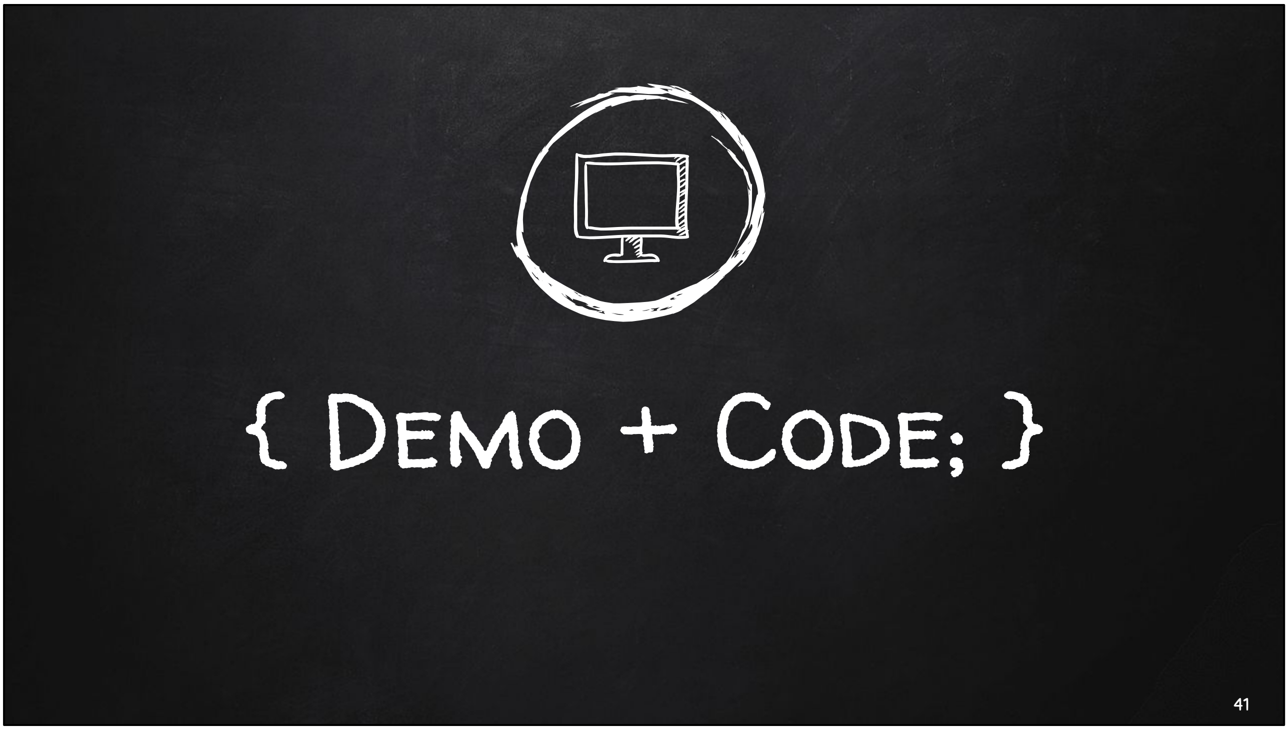
✘ User register 2nd device

✘ Backup codes (one-time passwords)

✘ SMS or email verification

Possibly a combination of several recovery options.

Authy: https://authy.com/



{ DEMO + CODE; }

Setup:

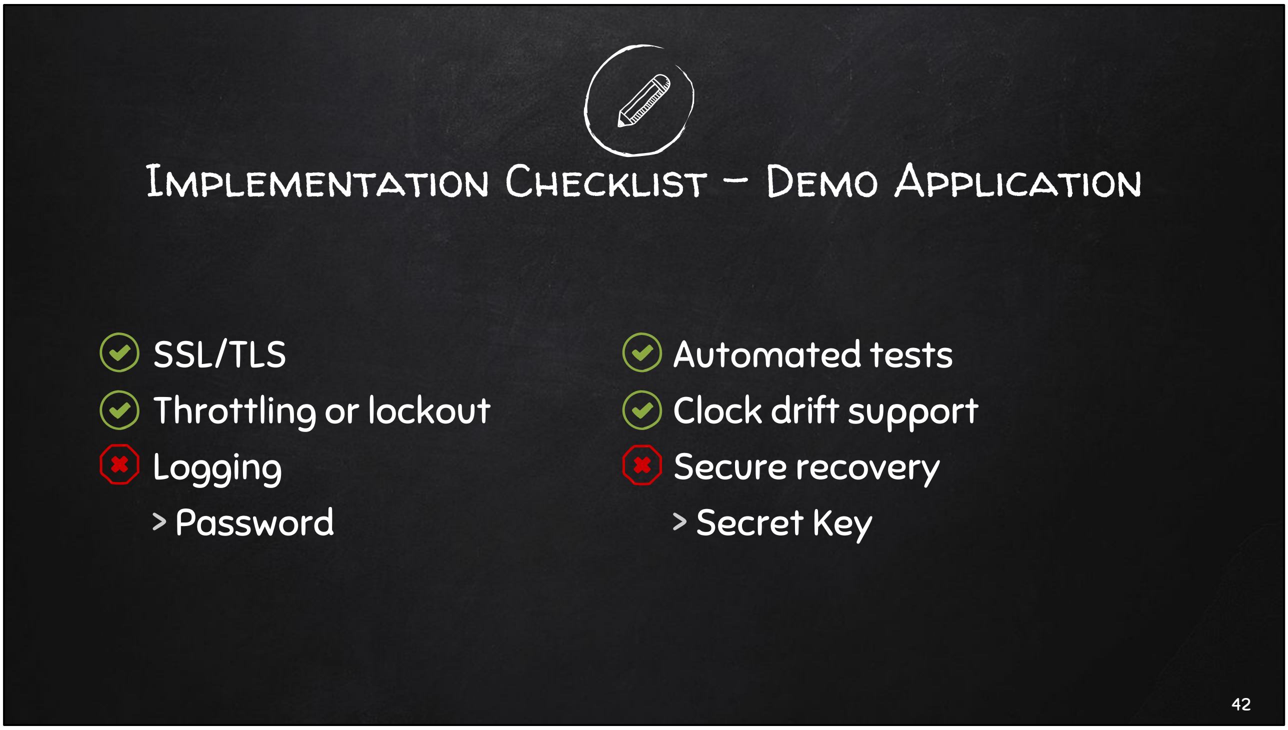
- Launch site and console app with ***dotnet run*** from project root
- https://www.isunshare.com/windows-10/change-font-and-font-size-in-windows-10-command-prompt.html to change console text size

Demo Talking Points

- Demo with 2FA
- Default to MFA and instructions

Code Talking Points

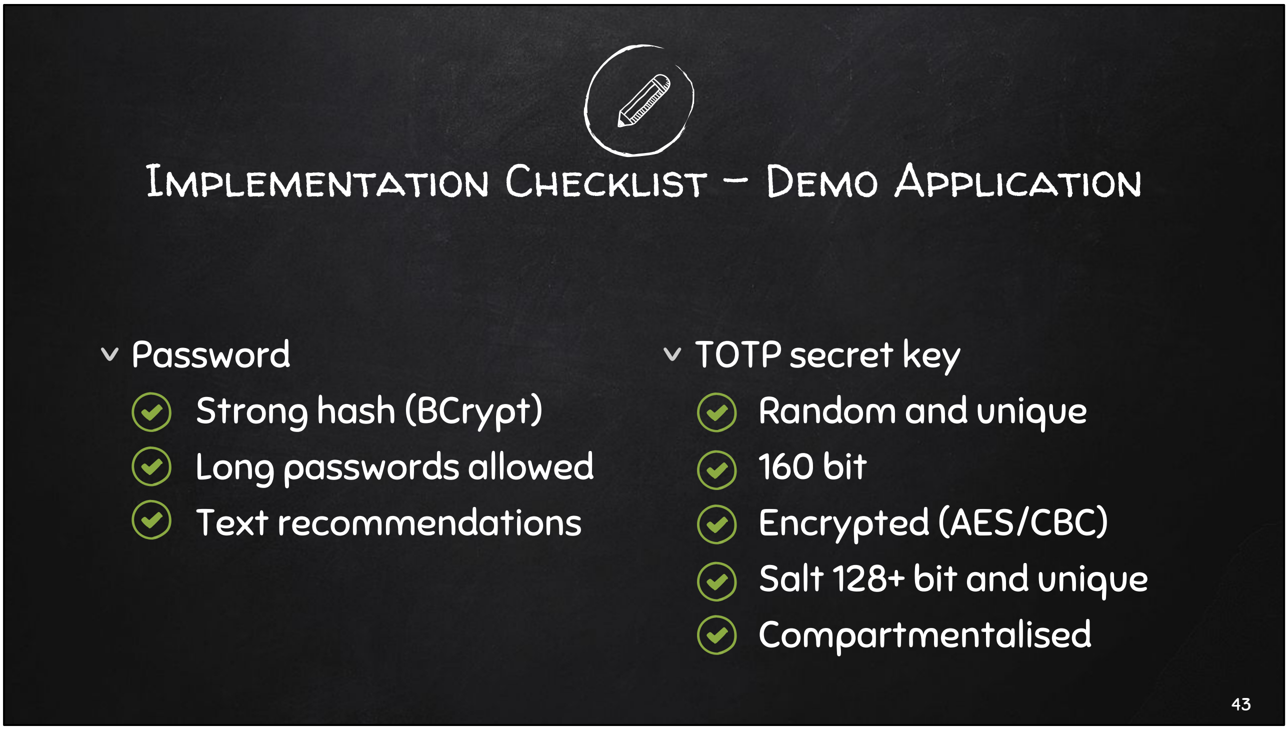
- MultiFactorAuthentication.ManualTotpTokenProvider class
- MultiFactorAuthentication.TotpTokenBuilder class
- TotpTokenGenerationTests.Rfc6238SpecificationTestVectors



Implementation Checklist – Demo Application

- ✓ SSL/TLS
- ✓ Throttling or lockout
- ✗ Logging
  - > Password
- ✓ Automated tests
- ✓ Clock drift support
- ✗ Secure recovery
  - > Secret Key

The Basics of Web Application Security: https://martinfowler.com/articles/web-security-basics.html



Implementation Checklist – Demo Application

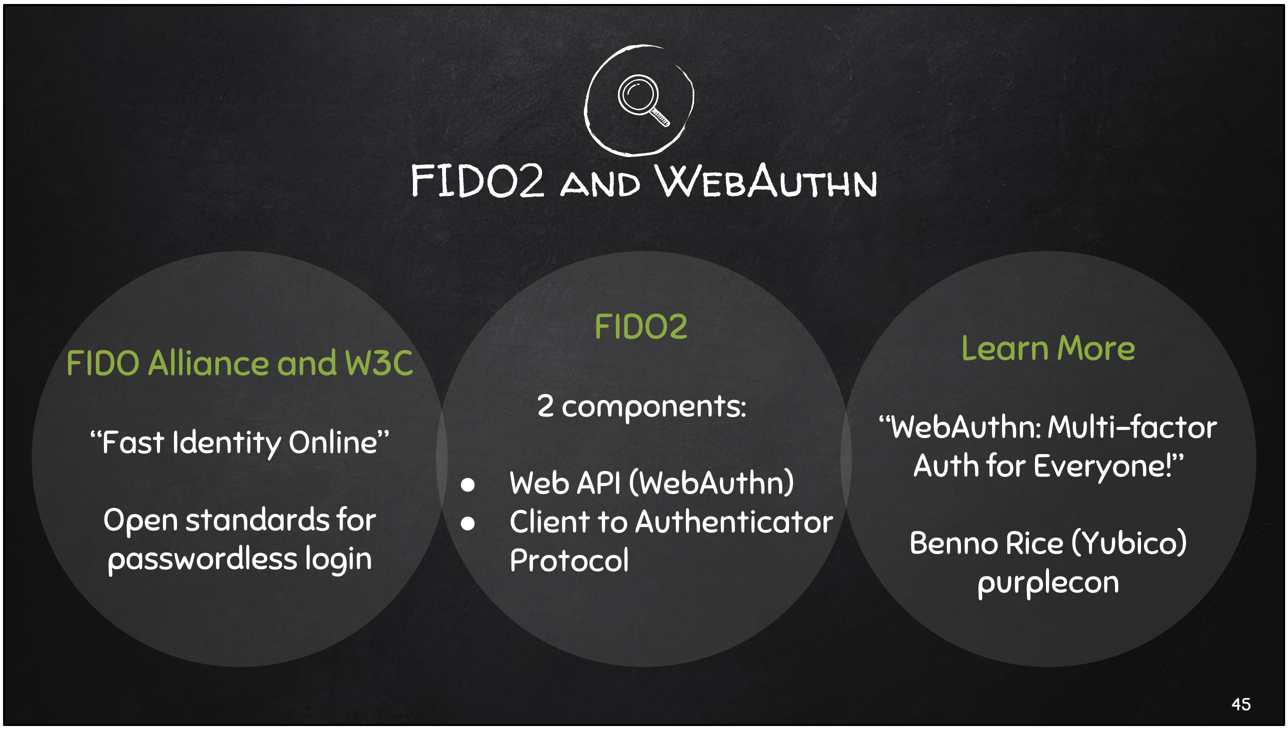
- Password
  - ✓ Strong hash (BCrypt)
  - ✓ Long passwords allowed
  - ✓ Text recommendations
- TOTP secret key
  - ✓ Random and unique
  - ✓ 160 bit
  - ✓ Encrypted (AES/CBC)
  - ✓ Salt 128+ bit and unique
  - ✓ Compartmentalised

- BCrypt hash: https://en.wikipedia.org/wiki/Bcrypt
- AES symmetric key algorithm with Cipher Block Chaining (CBC) mode: https://tools.ietf.org/html/rfc3602

On the Horizon

✘ FIDO2 and WebAuthn Standard

✘ Behaviour Recognition



FIDO2 and WebAuthn

FIDO Alliance and W3C

"Fast Identity Online"

Open standards for passwordless login

FIDO2

2 components:

- Web API (WebAuthn)
- Client to Authenticator Protocol

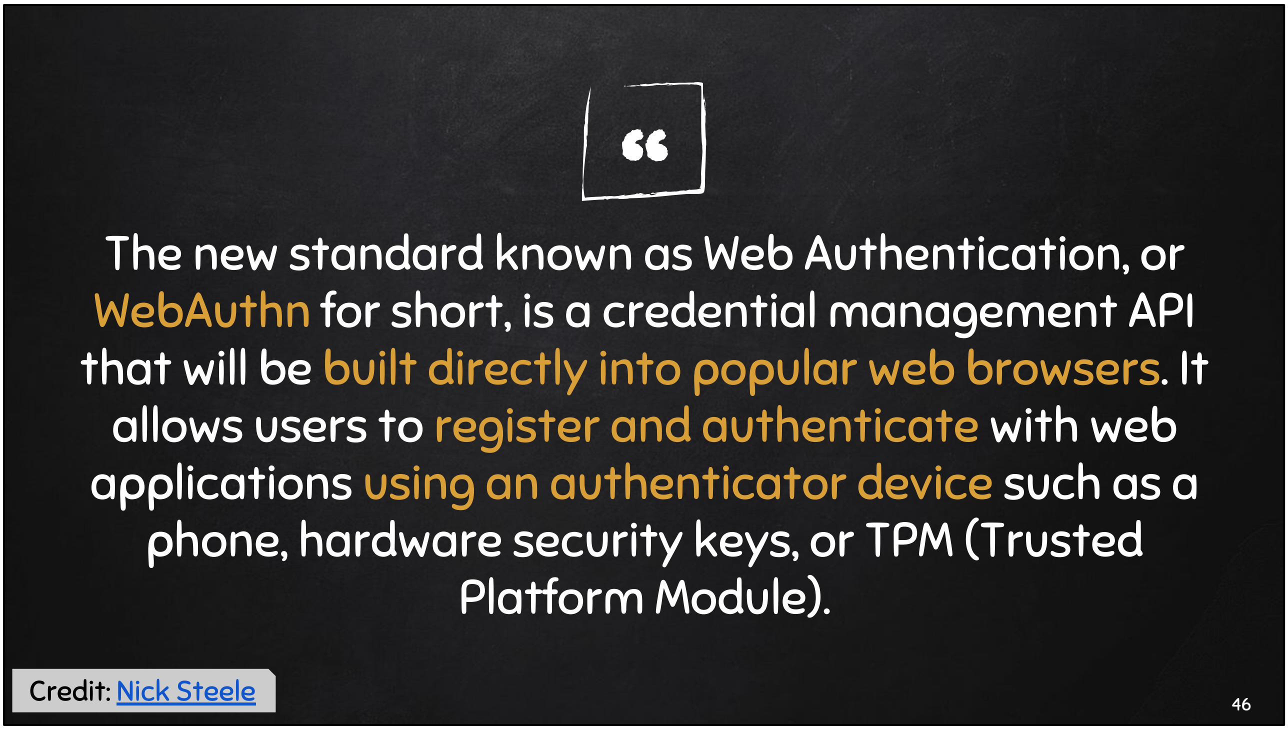
Learn More

"WebAuthn: Multi-factor Auth for Everyone!"

Benno Rice (Yubico) purplecon

References:

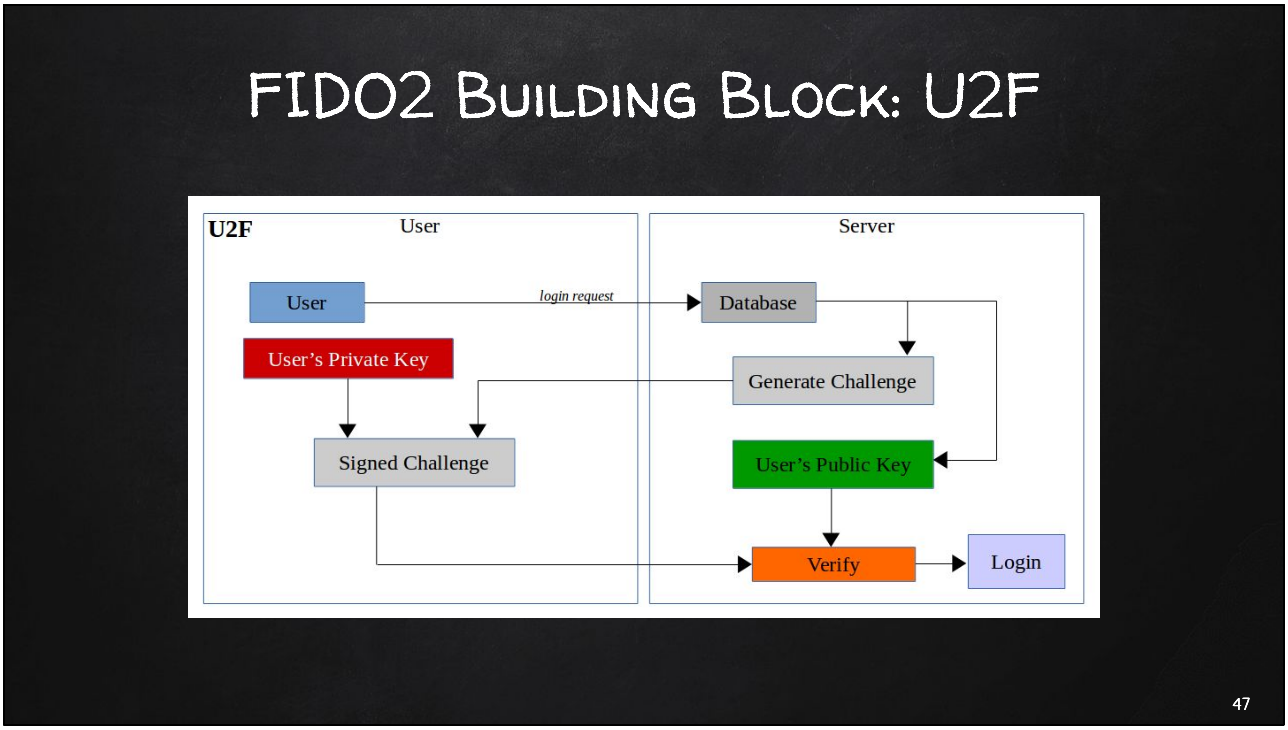
- https://www.wired.com/story/webauthn-in-browsers/
- https://www.yubico.com/2018/08/10-things-youve-been-wondering-about-fido2-webauthn-and-a-passwordless-world/
- https://duo.com/blog/web-authentication-what-it-is-and-what-it-means-for-passwords



> The new standard known as Web Authentication, or WebAuthn for short, is a credential management API that will be built directly into popular web browsers. It allows users to register and authenticate with web applications using an authenticator device such as a phone, hardware security keys, or TPM (Trusted Platform Module).

Credit: Nick Steele

WebAuthn Standard Specification:
https://www.w3.org/TR/webauthn/



FIDO2 Building Block: U2F

U2F

User

Server

User

login request

Database

User's Private Key

Generate Challenge

Signed Challenge

User's Public Key

Verify

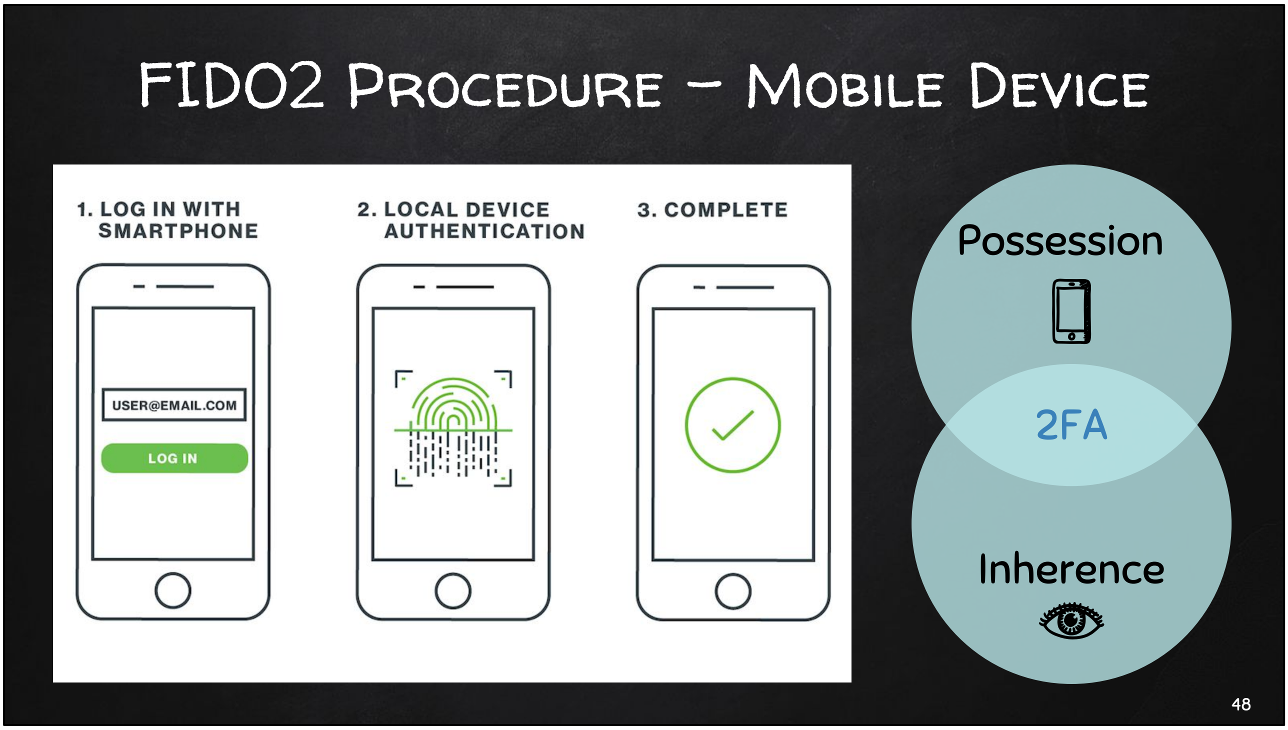
Login

U2F serves as a 2nd factor (possession) that is more secure than TOTP

Benefits of U2F:

- Relies on public-key cryptography
- No shared key
- No typing of TOTP codes

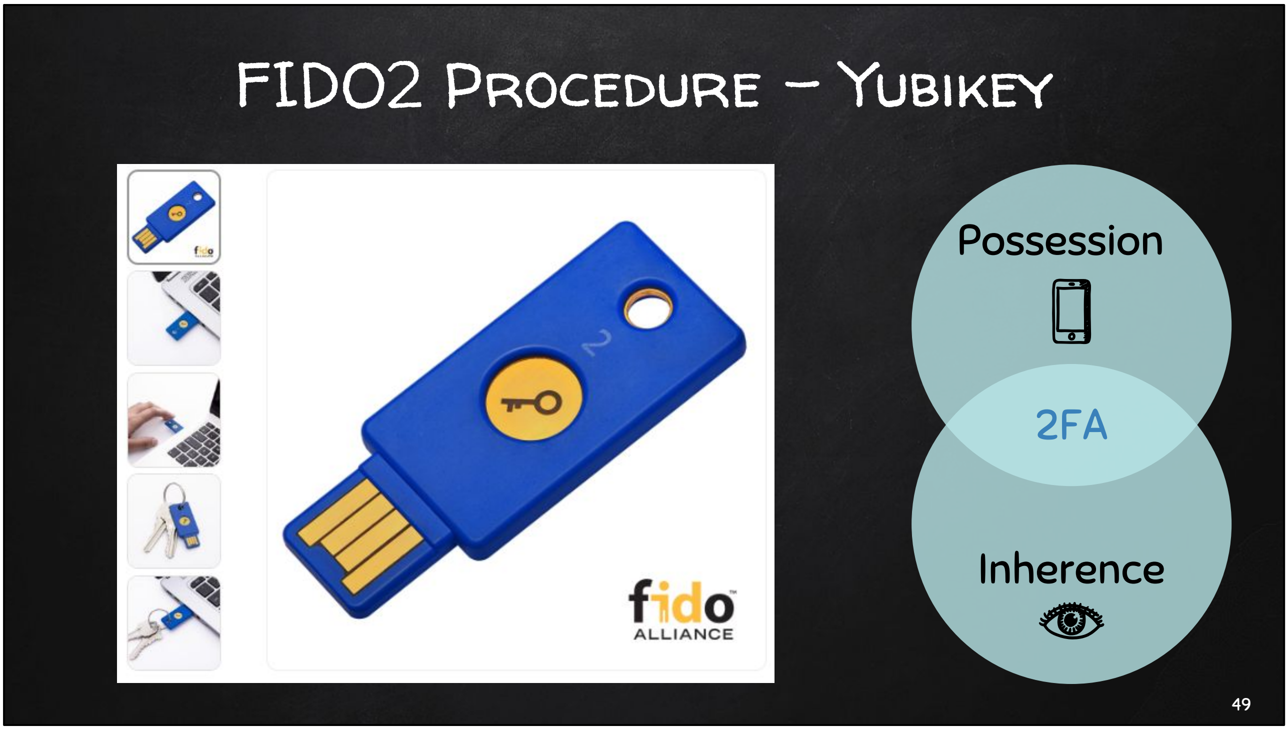
Reference:
https://blog.trezor.io/why-you-should-never-use-google-authenticator-again-e166d09d4324



FIDO2 Procedure – Mobile Device

1. Authenticate on local device with biometric (**inherence factor**)
2. Public-key challenge to host (**possession factor**)

Reference:
https://duo.com/blog/web-authentication-what-it-is-and-what-it-means-for-passwords



FIDO2 Procedure – Yubikey

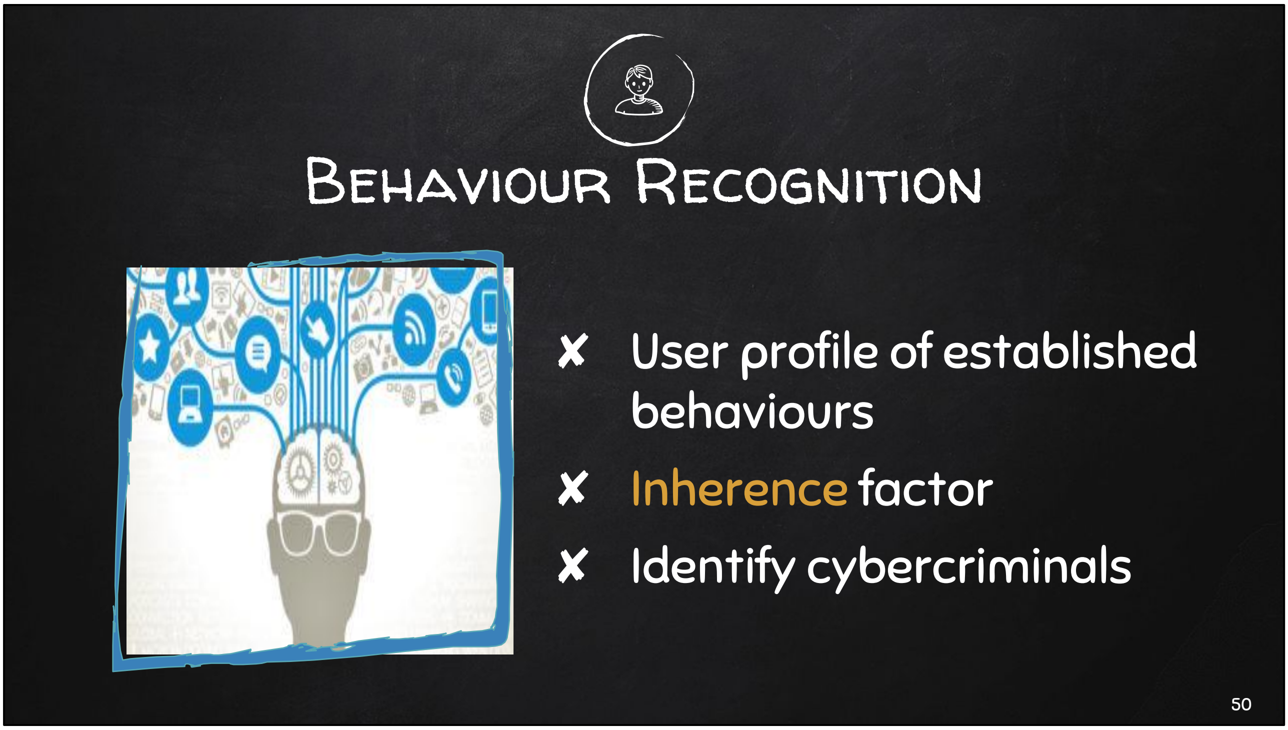
Possession

2FA

Inherence

Another option is the Yubikey:

- https://www.yubico.com/2018/04/new-security-key-fido2/
- https://www.yubico.com/product/security-key-by-yubico/

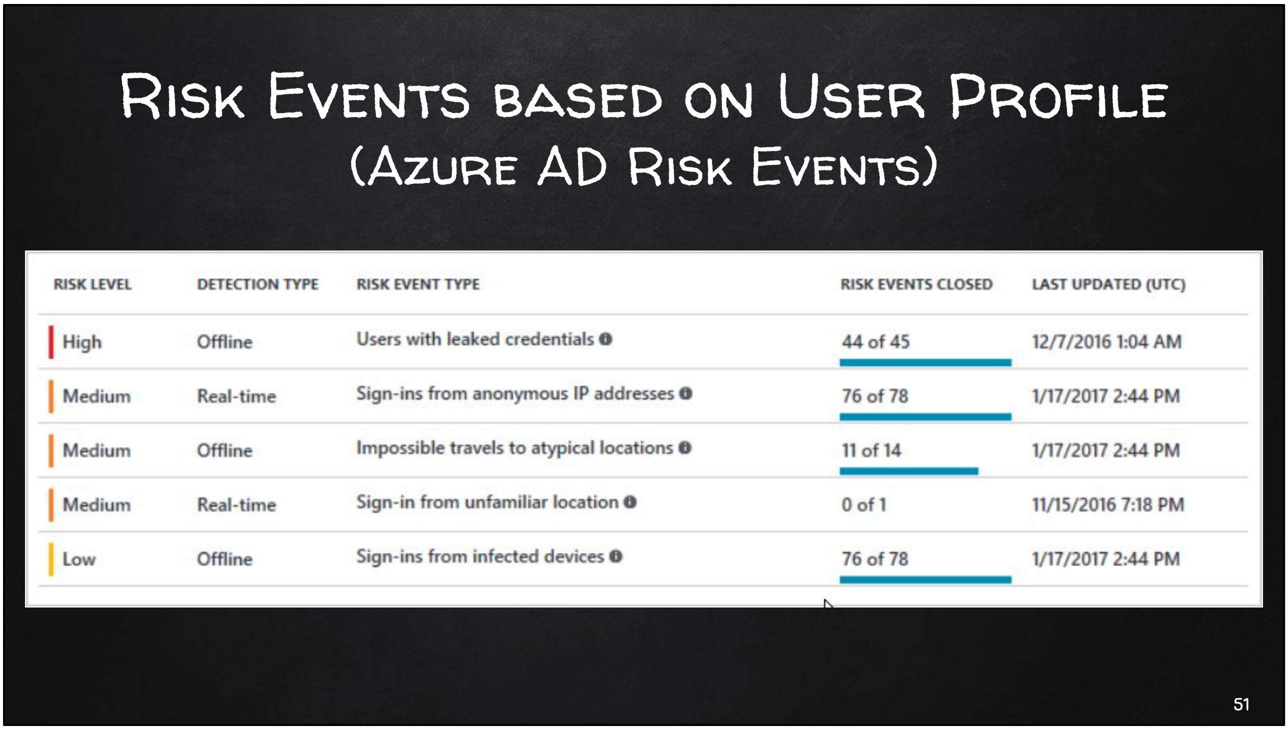


Behaviour Recognition

- ✘ User profile of established behaviours
- ✘ Inherence factor
- ✘ Identify cybercriminals

Resources:

- https://techcrunch.com/2015/08/23/next-gen-cybersecurity-is-all-about-behavior-recognition/
- https://docs.microsoft.com/en-us/azure-advanced-threat-protection/what-is-atp
- https://docs.microsoft.com/en-us/azure/active-directory/identity-protection/overview



Risk Events based on User Profile (Azure AD Risk Events)

RISK LEVEL	DETECTION TYPE	RISK EVENT TYPE	RISK EVENTS CLOSED	LAST UPDATED (UTC)
High	Offline	Users with leaked credentials	44 of 45	12/7/2016 1:04 AM
Medium	Real-time	Sign-ins from anonymous IP addresses	76 of 78	1/17/2017 2:44 PM
Medium	Offline	Impossible travels to atypical locations	11 of 14	1/17/2017 2:44 PM
Medium	Real-time	Sign-in from unfamiliar location	0 of 1	11/15/2016 7:18 PM
Low	Offline	Sign-ins from infected devices	76 of 78	1/17/2017 2:44 PM

Resources:

- https://docs.microsoft.com/en-us/azure/active-directory/reports-monitoring/concept-risk-events
- https://dotnetrocks.com/?show=1520

.NET Rocks Podcast “Security for Non-Profits” (story at 18:30 mark)

- Attack on school
- Hacker was password-spraying school
- Able to distinguish hacker’s login attempts
- Blocked hacker while allowing students to login even though hacker had some valid credentials
- Neither students nor hacker had any idea threat was detected and mitigated
- Relied on **login profile** of students

THANKS!

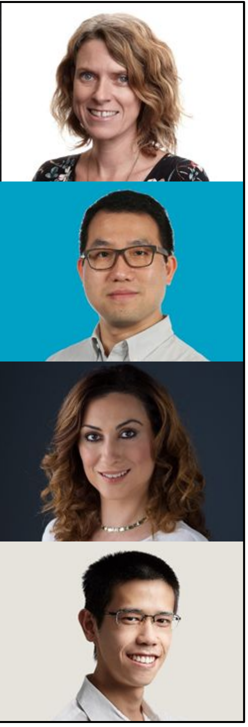
Kevin Thomas

Senior Consultant

Equinox IT

- linkedin @kevpthomas
- github @kevpthomas
- ✉ kevin.thomas@equinox.co.nz
- www.equinox.co.nz/about/kevin-thomas

EQUINOX IT



We believe solving tough business problems **starts with people**

We formed Equinox IT to do things better based on a fundamental belief in the power of people to achieve this. We aspired to build a company with 'balance' – where technology serves the business and where family, personal growth, social responsibility, diversity and sustainability are equal partners with profit, growth and commercial success.

- We aspire to enhance lives, accelerate careers, and see people flourish
- We seek to understand our clients first and strive to make a real difference for them and the communities they serve
- We call 'bullshit' on technology and practices that promise wonderful and magical things but inevitably fail to deliver.