

Security keys workshop

Sylvain Pelissier
Nils Amiet



November 24, 2023



Who are we?

- Nils Amiet
 - Security researcher
 - Authentication
 - Data processing at scale
 - Linux enthusiast
- Sylvain Pelissier
 - Security researcher
 - Applied Cryptography
 - Hardware attacks
 - CTF player
 - @ipolit@mastodon.social



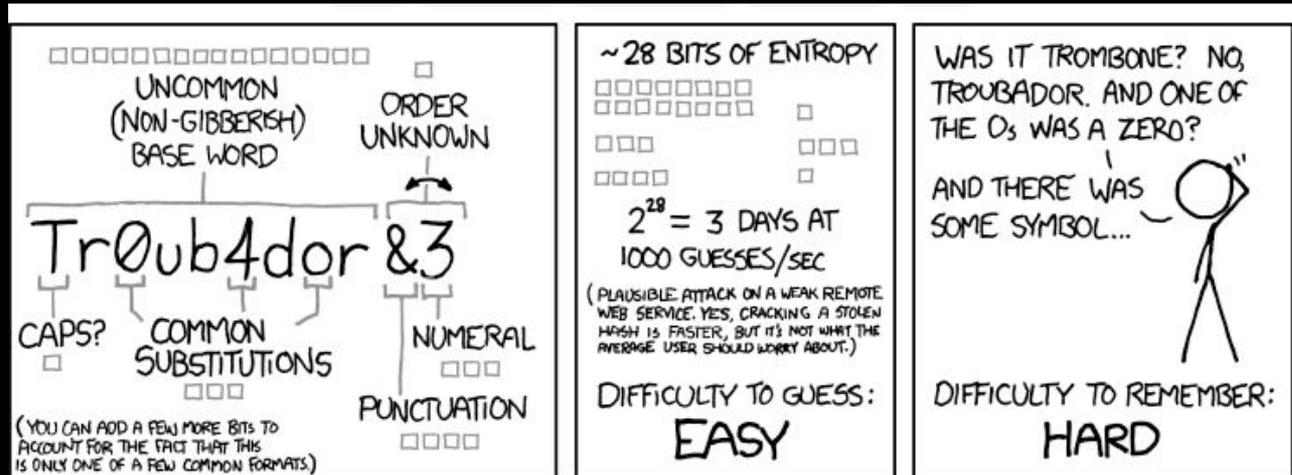
Introduction

Workshop goals

- Why should I use a security key?
- What is a secure key used for?
- Which security key should I use?

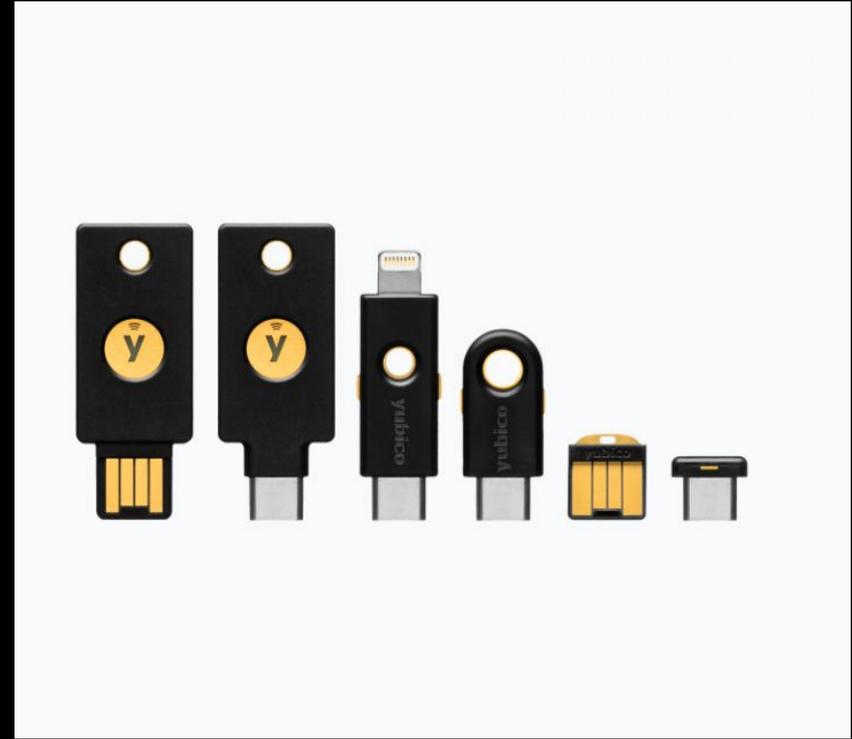
The password problem

- Difficult to remember
- Re-use across services leads to “credential stuffing”
- Phishing
- Computer or password manager breach



What's a security key?

- An isolated secure element or microcontroller performing only Cryptography operations
- Security keys can be used in a variety of uses cases to mitigate the password problem
 - Let's go through some of these



Set-up

Set-up

- Materials available at: <https://t.ly/FeAwy>
- A virtual machine is available there:
 - username: *user*
 - password: *changeme*
- Slides
- All the commands are in a VM file in folder `~/workshop` or at the link above.

Use case: 2FA

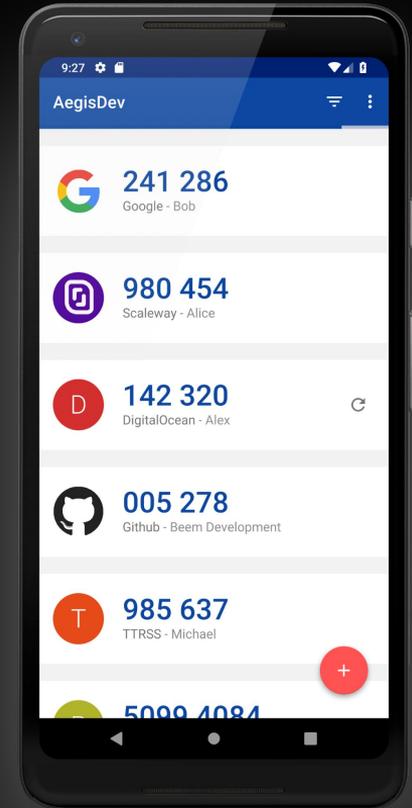
Security key as a second factor

Two-factor authentication (2FA)

- Generally in addition to a password to avoid a single point of failure.
- It can be a SMS on a mobile phone, an authenticator application, a TPM, or a **security key**.
- The advantage of security key is to have a separated device less prone to be breached.

OATH, HOTP, TOTP

- OATH (Initiative for Open Authentication)
 - HOTP: HMAC-based
 - TOTP: Time-based
- A one time code generated by another device.
- Used as a second factor together with a password.
- It is unlikely that both devices will be compromised at the same time.
- Recommended Android application: [Aegis Authenticator](#)
 - Open-source and easy backups.
- Other solutions: Google Authenticator, FreeOTP, Authy, etc.



HOTP: HMAC-Based One-Time Password Algorithm

- Defined in **RFC 4226**
- Less used in practice
- Both the client and the server share a secret key or seed.
- Server sends a challenge value **C** (for example a counter)
- Client answer by:

$$\text{HOTP}(\text{Key}, \mathbf{C}) = \text{Truncate}(\text{HMAC-SHA-1}(\text{Key}, \mathbf{C}))$$

- The result of truncate is a for example a 6-digit number.

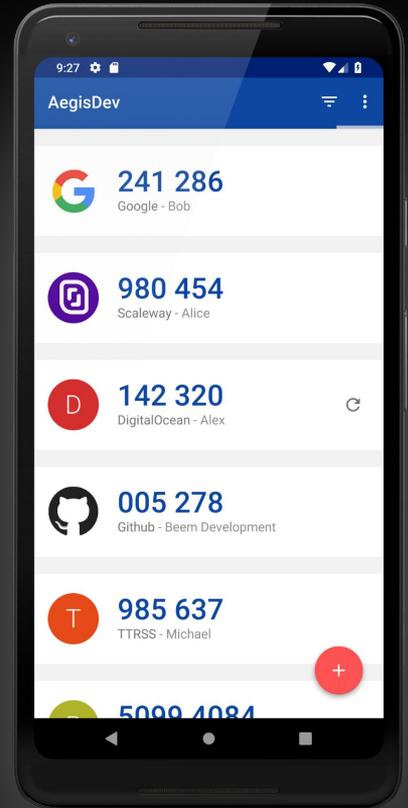
TOTP: Time-Based One-Time Password Algorithm

- Defined in **RFC 6238**
- Both the client and the server share a secret key or seed.
- Server use as a challenge value **T** the current time (Unix epoch) in steps of 30 seconds.
- Client sends:

$$\text{TOTP}(\text{Key}, \text{T}) = \text{Truncate}(\text{HMAC-SHA-1}(\text{Key}, \text{T}))$$

- The result of truncate is a 6-digit number.
- Check online:

<https://www.token2.com/site/page/totp-toolset>



TOTP with security key

- The advantage of using a security key for TOTP is that the secret key never leaves the security key
- The drawback is that if you lose your security key you may be locked out of your account
- On Yubikeys you can set-up up to 32 accounts.
- Can be used together with a mobile application (Then secure as the weakest link)
- Some software TOTP also exist like : [totp-cli](#)

TOTP with security key

Prerequisites for Yubikeys:

PCSC tools and screenshot:

```
$ sudo apt install pcscd gnome-screenshot
```

Install Yubico Authenticator

- Get latest version from

<https://www.yubico.com/products/yubico-authenticator/>

```
$ tar -xvf yubico-authenticator-latest-linux.tar.gz  
$ cd yubico-authenticator-6.3.0-linux/  
$ ./authenticator
```

TOTP with security key (Arch)

Prerequisites for Yubikeys:

PCSC tools and screenshot:

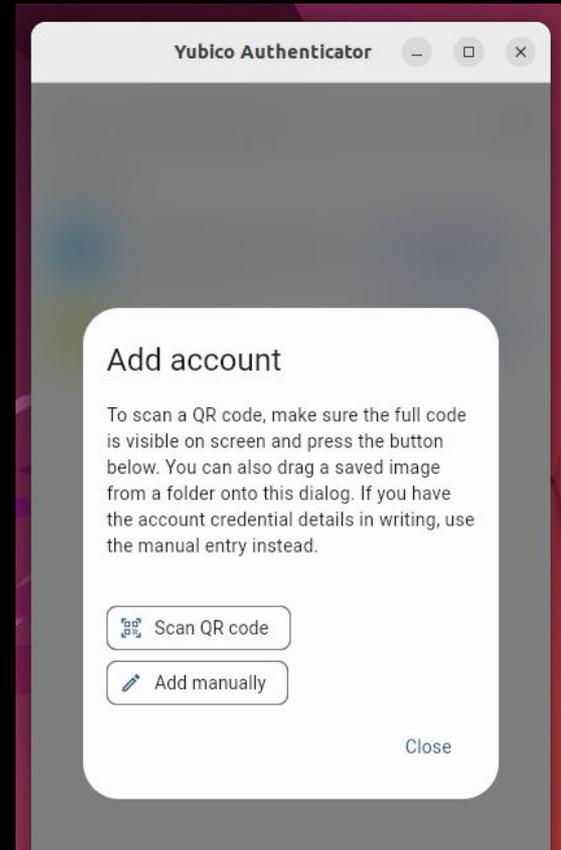
```
$ sudo pacman -S pcsc-tools gnome-screenshot  
$ sudo systemctl enable --now pcscd
```

Install Yubico Authenticator:

```
$ cd workshop  
$ tar -xvf yubico-authenticator-latest-linux.tar.gz  
$ cd yubico-authenticator-6.3.0-linux/  
$ ./authenticator
```

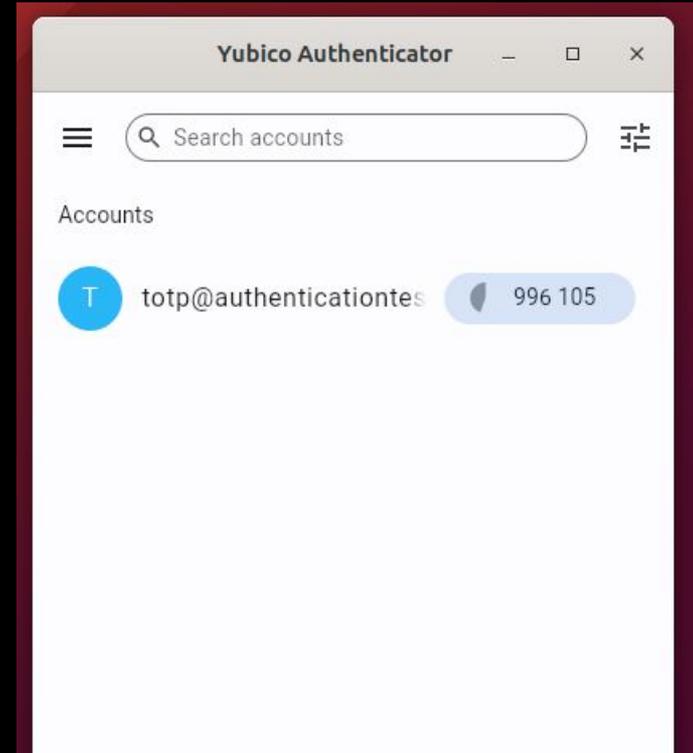
TOTP with security key

- Go to <https://authenticationtest.com/totpChallenge/>
- Add an account in authenticator:
 - Scan QR code
 - Or enter code manually



TOTP with security key

- Go to <https://authenticationtest.com/totpChallenge/>
- Add an account in authenticator:
 - Scan QR code
 - Or enter code manually
- Use the TOTP to login.
- Check that the code changes every 30 seconds



Yubikey Manager

Install Yubikey Manager

```
$ sudo apt install yubikey-manager
```

```
$ ykman info
```

```
Device type: YubiKey 4
```

```
Serial number: 5409811
```

```
Firmware version: 4.3.4
```

```
Enabled USB interfaces: OTP, FIDO, CCID
```

```
...
```

Yubikey Manager

Install Yubikey Manager (Arch)

```
$ sudo pacman -S yubikey-manager
```

```
$ ykman info
```

```
Device type: YubiKey 4
```

```
Serial number: 5409811
```

```
Firmware version: 4.3.4
```

```
Enabled USB interfaces: OTP, FIDO, CCI
```

TOTP on the command line

Create TOTP account:

```
$ ykman oath accounts add test  
Enter a secret key (base32): I65VU7K5ZQL7WB4E
```

Get TOTP codes:

```
$ ykman oath accounts code  
test 389685  
totp@authenticationtest.com 389685
```

Your turn now...

- Github
- Gitlab
- Gmail
- YesWeHack
- Wordpress
- X (Twitter)

A complete list of supported services is here: <https://www.dongleauth.com/>

Backup and disaster recovery

Backup your keys ! A mobile app can be a backup of security key.

Be careful of **fall-back settings...**

How Ethereum's Vitalik Buterin Got Hacked and the Damage Done



Published September 12, 2023 11:50 AM

By [Teuta Franjkovic](#)

KEY TAKEAWAYS

- Vitalik Buterin said that a SIM swap assault was the cause of the hack of his X account.
- The incident occurred on September 9 and resulted from con artists advertising a false NFT giveaway, which cost victims more than \$690,000.
- Buterin stressed the value of Twitter account security and the advantages of decentralized social platforms for security.

Week 32: WhatsApp hacking via voicemail

15.08.2023 - Cybercriminals are still targeting WhatsApp accounts. Attackers are pulling out all the stops to obtain the PIN code for resetting an account and they particularly appreciate having the code read out over the phone. If this is done at night, the code usually ends up being sent to voicemail, which is then hacked to obtain the information. The NCSC is currently receiving a lot of reports of hacked WhatsApp accounts.



Use case: Passkeys/FIDO2 Security key as a 1st factor

Passkeys

- A new way to sign in to apps and websites
- Sign in the same way you unlock your device
 - Face scan, fingerprint, screen lock PIN
- The first factor is now an “authenticator”
 - Not a password anymore
 - The authenticator can be a **smartphone**, a **laptop**, etc.
 - But it can also be a **security key**

WebAuthn.io

A demo of the WebAuthn specification

foobar

Register

Choose a device for your passkey

Select where you would like to save your passkey for **webauthn.io**

 NFC security key

 USB security key

 This device

Cancel

Passkeys (continued)



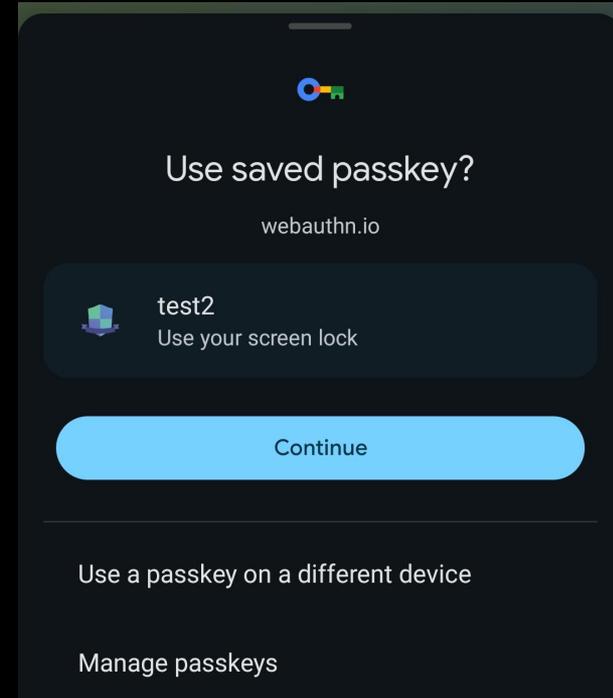
- Passkeys are built with FIDO2
 - FIDO2 = WebAuthn + CTAP2
 - WebAuthn: specification for web browsers
 - CTAP2: specification for Client <-> Authenticator communication (USB, NFC)
- Web browser must support WebAuthn Javascript API
 - Supported by all major web browsers
- Authenticator must support CTAP2
 - Supported by all “FIDO2-compatible” security keys

Remember FIDO2?

- Previously: **U2F** (2014)
 - Designed for 2nd factor only
- Previously: **FIDO2** (2019)
 - Passkey was locked to the device it was generated on
- **Passkeys** (October 2022)
 - Passkey = FIDO2 + More intuitive user experience
- But what's different/new?
 - **Cloud synchronized keys**
 - Android: Google Password Manager + 3rd party (14+)
 - iOS: iCloud Keychain + 3rd party (17+)
 - **Cross-device authentication** (CDA)
 - Third-party passkey providers
 - Strongest attempt so far to achieve large-scale adoption of a replacement for passwords

How it works (simplified)

- Registration
 - Generate key pair
 - A) Store private key on device (**resident credential**)
 - Uses space on the device
 - B) Offload storage to website/app
 - Simplified: key is encrypted using master key
 - [-] Cannot do “**username-less**” scenario
 - [+] Unlimited storage
 - Website stores public key
- Login
 - Generate signature with private key
 - Website verifies signature with public key



Passkeys: phishing resistance

- Credentials can only be used with the website they were generated for
 - Enforced at the protocol level
- Registration
 - **Website URL** is computed via browser internal API
 - URL is stored alongside credentials
- Login
 - **Website URL** is computed via browser internal API
 - Browser asks authenticator if credentials for this URL exist
- This **completely eliminates phishing** attacks
 - Unless browser exploit exists!
- Also: **breach resistance**... or is it really? :)
 - What about cloud-synchronized passkeys?



Are passkeys replacing security keys?

- No
- You may not want to sync your passkeys in the cloud
 - Device-bound passkey
 - Keep the passkey on a security key, **offline**
- You may need Authenticator Assurance Level 3 security (AAL3)
 - <https://pages.nist.gov/800-63-3-Implementation-Resources/63B/AAL/>
- It can be used as a **backup**, in case you lose access to all your other passkey devices

Disaster recovery

- Example: Passkey stored on smartphone
- Smartphone becomes unavailable (stolen, lost, broken)
- Are you locked out of your account forever?
- No, fallback method can be used
 - Password + 2nd factor (**security key**, TOTP, SMS, recovery code, etc.)
 - Backup passkey on a security key
 - Cloud-synced passkey with biometrics (test it first!)
- Make sure to setup a fallback method
 - Don't get locked out

Device support

- It's still very new
- Being rolled out to major OSes and web browsers
- <https://passkeys.dev/device-support/>

Managing your FIDO2 security key

- Chrome settings
 - Settings > Privacy and security > Security > **Manage security keys**
 - Direct link: **chrome://settings/securityKeys**
 - List/delete **credentials**
 - List/enroll/delete **fingerprints**
 - Set/change **PIN**
 - **Reset** security key
- To set fingerprints, use **ctapcli** if Chrome doesn't work
 - (installs to `~/.cargo/bin`)

```
$ cargo install ctap-hid-fido2 --example ctapcli
$ ctapcli bio -h
```

- Alternative: `fido2-token` from the “libfido2” package (Ubuntu: `fido2-tools`)
 - Not as intuitive to use
- Yubico tools, such as `ykman-gui`, are locked to products from that vendor only

Managing your FIDO2 security key (continued)

```
$ ctapcli info
- versions                = ["U2F_V2", "FIDO_2_0", "FIDO_2_1_PRE"]
- extensions              = ["credProtect", "hmac-secret"]
- aaguid(16)              = 8876631BD4A0427F57730EC71C9E0279
- options                 = [{"rk", true}, {"up", true}, {"plat", false}, {"credMgmt", true}, {"clientPin", false}]
```

```
$ ctapcli cred
PIN:
Enumerate discoverable credentials.
- existing discoverable credentials: 1/49
- rp: (id: webauthn.io, name: test)
  - credential: (id: 6447567A6441, name: test, display_name: test)
```

References

- <https://passkeys.dev/>
- <https://developer.apple.com/passkeys/>
- <https://fidoalliance.org/white-paper-multi-device-fido-credentials/>
- <https://security.googleblog.com/2022/10/SecurityofPasskeysintheGooglePasswordManager.html>
- <https://web.dev/articles/passkey-registration>
- <https://www.passkeys.io/>
- <https://webauthn.me/debugger>
- <https://www.eff.org/deeplinks/2023/10/what-passkey>
- <https://www.imperialviolet.org/2023/07/23/u2f-to-passkeys.html>
- <https://www.imperialviolet.org/2022/09/22/passkeys.html>
- <https://developers.yubico.com/Passkeys/>

Exercise

- Go to <https://WebAuthN.io>
- Register
- Authenticate
- Try with
 - 1) Laptop: security key
 - 2) Smartphone: internal passkey
 - 3) Laptop: Cross-device authentication
 - Login to Chrome
 - Turn on bluetooth
 - Login with passkey stored on smartphone, but from laptop

Your turn now...

- <https://passkeys.directory/>
- Github
- Google
- Microsoft
- Apple
- Nintendo
- NextCloud (Up to your provider)

Use case: Static password

Yubikey slots

- There are 2 slots for shared applications (on Yubikey)
 - Yubico OTP
 - HMAC-based OTP (OATH HOTP)
 - Static password
 - Challenge-response (HMAC-SHA1 or Yubico OTP algorithms)

```
$ ykman otp info  
Slot 1: programmed  
Slot 2: programmed
```

Yubikey slots

- Short touch the Yubikey's round part, it will type the value stored in slot 1 as a keyboard
- For Raspberry Pi Pico it is the "Boot" button
- Long touch (about 3 seconds) will type the value in slot 2



Static password

- Use cases
 - Arbitrary string you type often
 - Passwords on devices where it's slow to type
 - Netflix on vacation TV
 - Apple TV
 - Game console
 - Make sure to set higher intra-character pacing
- If storing passwords
 - Only store a part of the password in the OTP slot
 - Type the rest manually
 - Someone who steals your security key doesn't get your full password
 - **Example:** myPa\$\$word76
 - Type manually: myP
 - Store: a\$\$word76

Static password setup (Yubikey)

- By default, can only store modhex data, to be compatible with all keyboard layouts
 - These 16 characters are allowed only: **cbdefghijklnrtuv**
 - Make sure to pass “-k fr” keyboard layout to enable use of additional characters
- Yubikey 4: max length = 38 modhex characters per OTP slot
- Store secret

```
$ ykman otp static -k fr <slot number>
```

- Generate secret

```
$ ykman otp static --length 24 --generate <slot number>
```

- Print secret
 - Short touch the Yubikey’s round part, it will type the value stored in slot 1 as a keyboard
 - Long touch (about 3 seconds) will type the value in slot 2

Intra-character pacing, final Enter keystroke

- On game consoles, the pacing at which keystrokes are typed may be too fast
 - Some keys may not be registered correctly
- You may not want to send a final **Enter** either
- Change the OTP slot settings:

```
$ ykman otp settings --pacing <pacing> --no-enter <slot number>
```

- **Pacing**: number of milliseconds between each keystroke (default=0)
 - Possible values: 0, 20, 40, 60
- **--no-enter**: Do not send an Enter keystroke after slot output (default=type enter)

Use case:
Sudo with a security key

Prerequisites

Install libpam-u2f (Ubuntu):

```
$ sudo apt-get install libpam-u2f
```

Install libpam-u2f (Arch):

```
$ sudo pacman -S pam-u2f
```

Sudo configuration

Associate the key with your account:

```
$ mkdir -p ~/.config/Yubico  
$ pamu2fcfg > ~/.config/Yubico/u2f_keys  
Enter PIN for /dev/hidraw2:  
$ sudo nano /etc/pam.d/sudo
```

Sudo configuration

Edit permissions in `/etc/pam.d/sudo` at the beginning of the file:

```
#!/PAM-1.0

auth required pam_u2f.so cue [cue_prompt=Tap your security key]

# Set up user limits from /etc/security/limits.conf.

...
```

required means password AND security key.

sufficient means password OR key.

Sudo configuration

Test it works:

```
$ sudo echo test  
Tap your Yubikey  
[sudo] password for user:  
test
```

Warning when using “required”: if you lose your security key, you can no longer use sudo !

Recommended to register multiple security keys in case one is lost with:

```
$ pamu2fcfg -n >> ~/.config/Yubico/u2f_keys
```

Sudo configuration

Test:

```
$ sudo echo test
```

Tap your Yubikey

```
test
```

When using “sufficient” it falls back to password if no security key is detected.

Use case:

Login to Linux with a security key

Login configuration

Edit permission in `/etc/pam.d/gdm-password` (Gnome) or `sddm` (KDE):

```
#%PAM-1.0

auth required pam_u2f.so cue [cue_prompt=Tap your security key]

# Set up user limits from /etc/security/limits.conf.

...
```

required means passwords AND security key.

sufficient means password OR key.

Login with security key

Test 1FA



Login with security key and password

Test 2FA



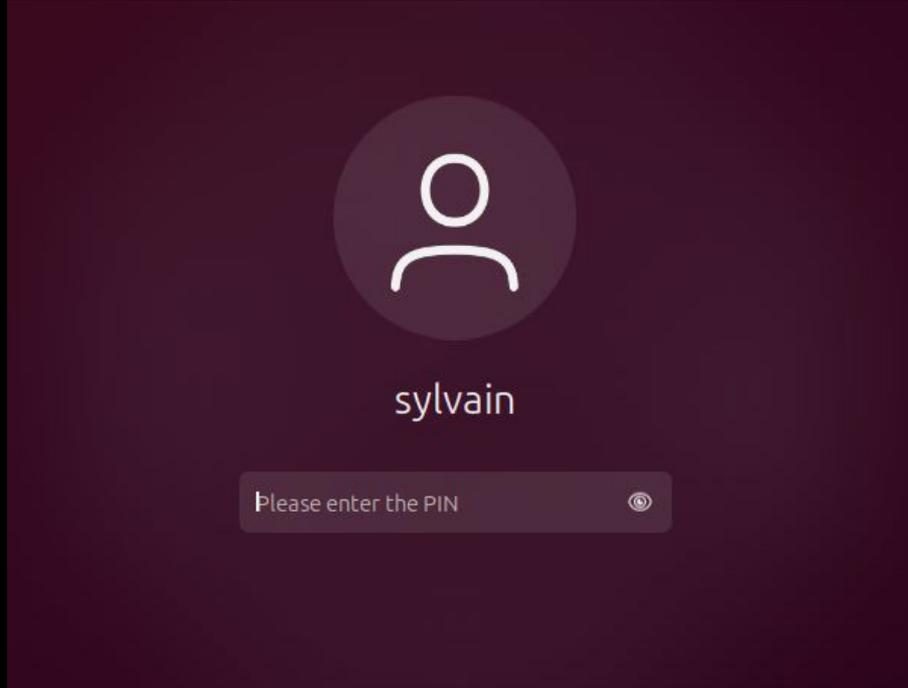
Login with security key and PIN

Configuration to request the security key PIN

```
$ pamu2fcfg -N > ~/.config/Yubico/u2f_keys  
Enter PIN for /dev/hidraw2:
```

Login with security key and PIN

Test 2FA



Risk analysis

Sufficient: Add no security, maybe decrease security. Think if someone steal you laptop and your key.

Required: Add a second factor for example the PIN. Think of what happen if you lose your key.

Warning: It does not configure the TTY terminals ! Once you are ready:

```
$ sudo nano /etc/pam.d/common-auth
```

```
##%PAM-1.0
```

```
auth required pam_u2f.so cue [cue_prompt=Tap your security key]
```

Use case: SSH over FIDO2

Your SSH key on a security key

SSH use case

- Requires OpenSSH 8.2 or later (released Feb 2020)
- Generate a portable SSH key to be stored on a FIDO2 security key:

```
$ ssh-keygen -t ecdsa-sk -O resident
```

- This will create
 - `~/.ssh/id_ecdsa_sk` (private key **handle***)
 - `~/.ssh/id_ecdsa_sk.pub` (public key)
- The private key **handle*** cannot be used by itself, it still requires the security key to be present
 - It's a pointer to the actual private key, stored on the security key

SSH use case (continued)

- Later, on another computer, load the resident key:
- 2 options

A) Load key into ssh-agent →

```
$ ssh-add -K
```

B) Save key to file on disk →

```
$ ssh-keygen -K
```

- Note: option A requires ssh-agent running and setup
- Then SSH into your machine as usual

For A)

```
$ ssh user@host
```

For B)

```
$ ssh user@host -i ~/.ssh/id_ecdsa_sk
```

Use cases

- SSH authentication
- Sign git commits with your SSH key (yes, SSH key, not PGP key)
 - <https://docs.github.com/en/authentication/managing-commit-signature-verification/telling-git-about-your-signing-key#telling-git-about-your-ssh-key>

Exercise

Reminder:

```
$ ssh-keygen -t ecdsa-sk -O resident
```

```
$ ssh-add -K
```

```
or $ ssh-keygen -K
```

- The following SSH server can be used for this exercise:
 - Host: hostname
 - Username: ph0wn
 - Password: ph0wn-security-key-workshop
 - Port: 22
- Generate an SSH key on your FIDO2 security key
- Add the public key to the SSH server's **authorized_keys** file

```
$ ssh-copy-id -i ~/.ssh/id_ecdsa_sk.pub username@hostname
```

- From your own machine, SSH into the server with pubkey authentication

```
$ ssh username@hostname
```

- Check that the SSH key on the security key is really portable
 - From another machine, SSH into the server with your security key

Disk encryption with FIDO2

Disk encryption

Requirements:

- FIDO2 security key with **'hmac-secret'** extension
- Recent OS

Prerequisites:

```
$ sudo apt install cryptsetup
```

Format disk:

```
$ sudo cryptsetup luksFormat /dev/sdb  
$ sudo cryptsetup open /dev/sdb encrypted  
$ sudo mkfs.ext4 /dev/mapper/encrypted  
$ sudo umount /media/user/<disk-id>  
$ sudo cryptsetup close encrypted
```

Partition encryption

Enroll security key:

```
$ sudo systemd-cryptenroll --fido2-device=auto --wipe-slot=all /dev/sdb
```

Open partition with the security key:

```
$ sudo cryptsetup open --token-only /dev/sdb encrypted
```

Add to the boot: edit **/etc/crypttab**

```
# <target name> <source device> <key file> <options>  
encrypted /dev/sdb - fido2-device=auto
```

Boot



Please enter LUKS2 token PIN:



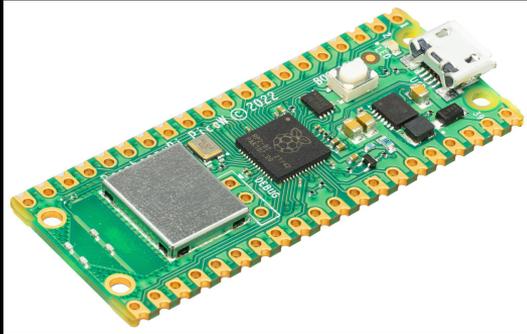
ch

Comparison of security keys

Hardware

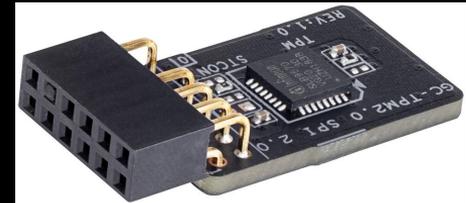
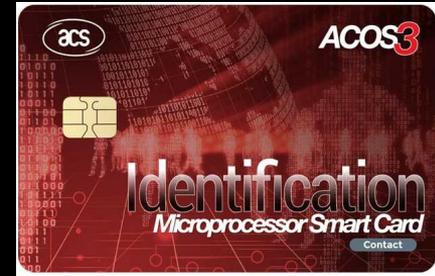
Microcontroller

- General purpose CPU
- Not design for security
- No HW attacks protection



Secure element

- Limited Functionality
- Isolation and designed for security
- May have an **Evaluation Assurance Level (EAL)**



Is the secure element used ?

The New Nitrokey 3 With NFC, USB-C, Rust, Common Criteria EAL 6+

The new Nitrokey 3 is the best Nitrokey we have ever developed. It offers NFC, USB-C for the first time. The Nitrokey 3 combines the features of previous Nitrokey models: FIDO2, one-time passwords, OpenPGP smart card, Curve25519, password manager, Common Criteria EAL 6+ certified secure element, firmware updates. This reliably protects your accounts against phishing and password theft, and encrypts your communications and data. With strong hardware encryption, trustworthy thanks to open source, quality made in Germany.

IMPORTANT NOTE: The included Secure Element is not used at this time. We are currently working on its integration and will enable its use later via firmware update.

Hardware Comparison

- Main possibilities:
 - Yubikey 5 : Infineon SLE78CLUFX5000 secure element (CC EAL6+)
 - Google Titan/Feitian Key: NXP A7005 secure element (CC EAL5+)
 - Ledger Nano X: ST31H320 or ST33J2M0 secure element (CC EAL5+)
 - Ledger Nano S: ST31 (CC EAL5+)
 - Nitrokey 3: Warning: keys stored in internal flash at the moment (Secure element not used)
 - Solokeys Solo 1: STM32L432 microcontroller
 - Solokeys Solo 2: NXP LPC55S69 microcontroller
 - Token2: Unknown hardware
 - TrustKey: eWBM MS500 microcontroller

More on FIDO2 hardware

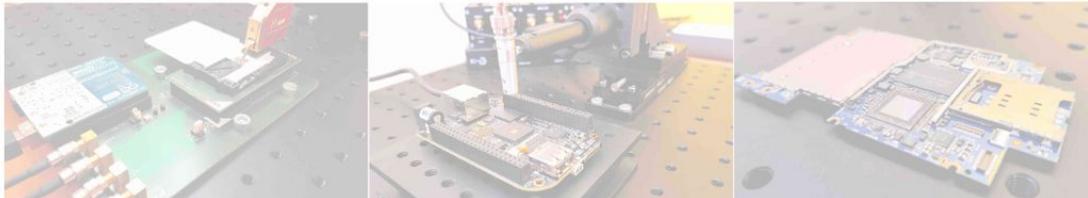
An Overview of the Security of Some Hardware FIDO(2) Tokens

Victor LOMNE

NinjaLab

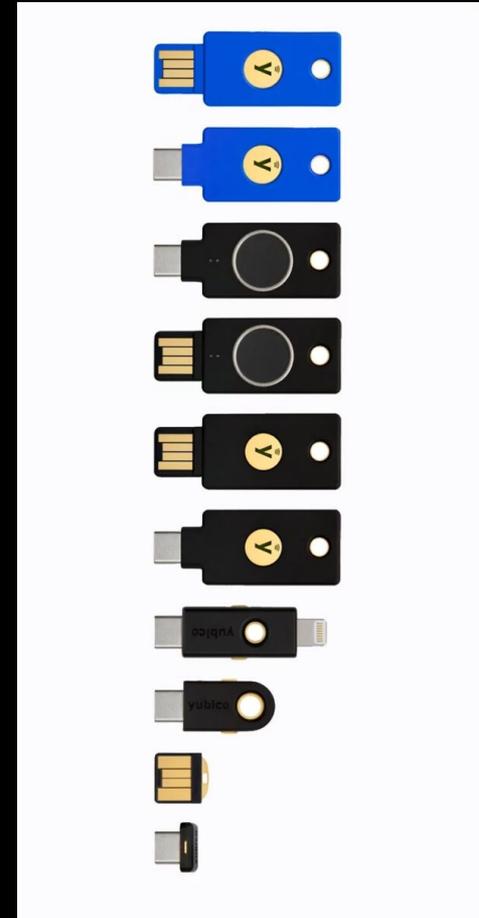
Hardwear.io NL, The Hague, Netherlands

October 28, 2022



Features to look for

- USB-C or USB-A
- NFC support
- TOTP/HOTP support
- FIDO2
 - Max number of resident keys
 - HMAC extension for disk encryption
 - Credential management support (credMgmt)
 - Credential protection (credProtect)
- (U2F)
- Open-source firmware
- Upgradable firmware
- Secure element
- Static password
- PGP
- PIV
- Biometrics (fingerprint reader)
 - Max number of fingerprints
- Certifications (FIPS 140-2, NIST SP800-63B, CC EAL)
- Price
- Form factor/design
- Support / Available software and documentation



	USB Type	NFC	Bio	Open source	FIDO2	OTP	PGP	PIV	SE	Price (USD)	Comment
Yubikey 4	A/C	No	No	No	No	T/H	Yes	Yes	Yes	-	No FIDO2
Yubikey 5	A/C	Yes	No	No	Yes/25	T/H	Yes	Yes	Yes	\$ 50	No biometrics
Security Key NFC	A/C	Yes	No	No	Yes/25	No	No	No	Yes	\$ 25	Limited features
Yubikey Bio	A/C	No	Yes	No	Yes/25	No	No	No	Yes	\$ 90	No NFC, limited features
Nitrokey 3	A/C	Yes	No	Yes	Yes/10	T/H	Yes	Yes	*Yes	\$ 55	*SE not in use (2023)
SoloKeys Solo 2+	A/C	Yes	No	Yes	Yes/100	No*	No	No*	No	\$ 46	Limited features, no SE, not production-ready
Google Titan Key	A/C	Yes	No	No	Yes/250	No	No	No	Yes	\$ 35	Limited features, limited docs

FIDO2 Certified Authenticator Levels

FIDO Authenticator Certification Examples		
	 <p>USB U2F Token built on a CC-certified Secure Element Certification: L3+</p>	
	 <p>USB U2F Token built on a basic simple CPU, OS, is certified. Good physical anti-tampering enclosure</p>	 <p>UAF implemented as a TA running on a certified TEE with POP memory</p>
	 <p>UAF implemented as a TA in an uncertified TEE</p>	
	 <p>UAF in downloadable app using white box crypto and other techniques Certification: L1+</p>	
	 <p>Downloaded app making use of Touch ID on iOS Certification: L1</p>	 <p>FIDO2 making use of the Android keystore. Keystore is not certified Certification: L1</p>
		 <p>FIDO2 built into a downloadable web browser app Certification: L1</p>

Conclusions

Questions to ask yourself

- What use cases (features) do I need?
- What is my threat model ?
 - What happens if my security key is stolen ?
 - Am I able to revoke my security key quickly ?
 - Do I care about hardware attacks ?

Conclusions

- Security keys can
 - Improve your security
 - Make you save time, make your life easier
- Always set up a fallback method - don't get locked out
 - If possible, **register 2 security keys**
- To choose the right security key for you, think of
 - Features you need
 - Threat model
 - Limitations
- There are multiple ways to achieve the same thing
 - See advanced topics in the bonus slides
- We hope we could answer the questions at the beginning
- Feel free to reach out if you have any questions

Thank you



Advanced topics

HOTP

HOTP on Yubikey

```
$ ykman otp hotp <slot number 1|2>
```

```
Enter a secret key (base32):
```

```
OBUDA53NNZPXO33SNNZWQ33Q
```

```
Program a HOTP credential in slot 2? [y/N]: y
```

Test on <https://www.verifyr.com/en/otp/check#hotp>

Challenge response

Challenge response

- Generic feature to allow further application using security keys.
- A 20-byte secret key can be programmed in a slot:

```
$ ykman otp chalresp -g 2
```

```
Using a randomly generated key (hex):
```

```
6d84db776b333d18030e7f03cf892633d106a47a
```

```
Program a challenge-response credential in slot 2? [y/N]: y
```

Challenge response

- The security key will receive a challenge of up to 64 bytes and return a response
- The response is the SHA1-HMAC (20 bytes) of the challenge with respect to the programmed key

```
$ ykman otp calculate 2
Enter a challenge (hex): 506830776e5f776f726b73686f70
8a2554113d0c1ef90177917910476583abb37c8f
```

- This feature was used previously to enable disk encryption in LUKS
 - See for example: [yubikey-luks](#)
- Can be used in an application through API

PGP

Prerequisites

Install smartcard daemon:

```
$ sudo apt-get install sddaemon
```

Test:

```
$ gpg --card-status
Reader .....: Yubico YubiKey OTP FIDO CCID 00 00
Application ID ...: D2760001240102010006054098110000
...
```

Secret key

List existing secret keys:

```
$ gpg -K
```

Generate a new one:

```
$ gpg --default-new-key-algo "rsa4096/cert,sign"  
--quick-generate-key "user@email.com"
```

PGP PIN codes

PGP applications has two PIN codes different from FIDO2 mode.

On Yubikey by default it is 123456 and the Admin PIN is 12345678

Add the key to the security key

Get the key ID

```
$ gpg -K  
/home/user/.gnupg/pubring.kbx  
-----  
sec   rsa4096 2023-11-06 [SC] [expires: 2025-11-05]  
      1704FD9D2803731B3F3E1C52CE2B95A5BB368B87  
uid   [ultimate] user@email.com
```

Add the key to the security key

Store the private key on the security key:

```
$ gpg --edit-key 1234ABC # where 1234ABC is the key ID of your key
gpg> keytocard
Really move the primary key? (y/N) y
Please select where to store the key:
  (1) Signature key
  (3) Authentication key
Your selection? 1
# Enter the passphrase of the key
# Enter the Admin PIN
```

Keep the public key somewhere.

Sign messages

Try signing a message:

```
$ echo "Hello" > msg.txt  
$ gpg --default-key sylvain@email.com --sign msg.txt  
$ gpg --verify secret.txt.gpg
```

Possible applications:

- Sign git commit
- (go)pass encryption key

Sign git commit

If you have set-up a GPG key in your Github account you can configure the url to fetch your public key in the Yubikey:

<https://github.com/<username>.gpg>

Add new GPG key

Title

Sylvain Pelissier

Key

-----BEGIN PGP PUBLIC KEY BLOCK-----

```
mQENBG TkqHgBCADHdmkeqLq7BcieS/+8gz8XF  
ljEjPTvh87MGrj32c2YmgYCUSiUxnAdabz2a4TJA9C  
wsiTJqG1Gz3+dHSZZeHLDmLaEwwWT4BePUTs7f  
epUm0gkOljYO3t4gv1tek50D10vD13u6pZ4bSNtc  
Mg7SWgtSfVIfNo4Tj1rJK/6gXvpFqLwbIr4afNeQY3  
ak+YYOsZoiWi0xg90X3KoGI93IYjj+Cxv/x9ABEBAA  
aW4ucGVsaXNzaWVyQGdtYWlsLmNvbT6JAU4EEv
```

Add GPG key

Configure public key URL

```
$ gpg --card-edit
```

```
gpg/card> admin
```

```
Admin commands are allowed
```

```
gpg/card> url
```

```
URL to retrieve public key: https://github.com/<user>.gpg
```

Fetch public key

```
$ gpg --card-edit
gpg/card> fetch
gpg: requesting key from 'https://github.com/user.gpg'
gpg: key 114312640BB4D65E: public key "user <user@email.com>"
imported
gpg: Total number processed: 1
gpg:             imported: 1
```

Sign git commit

```
$ git config --global user.email "user@email.com"
$ git config --global user.signingkey 37C79B97D4B...
$ git commit -S -m "YOUR_COMMIT_MESSAGE"
$ git cat-file -p HEAD
tree e34999490a93ec82c4c2508d359272ad31d9129a
parent 8c145574d6de3c440a01e79a86e2737b99099788
author user <user@email.com> 1699601993 +0100
committer user <user@email.com> 1699601993 +0100
gpgsig -----BEGIN PGP SIGNATURE-----
...
```

Use case: PIV smart card

Security key as a PIV smart card

PIV smart card use case

- PIV = **Personal Identity Verification**
 - PIV Interface defined in NIST SP 800-73 standard
- PIV card contains slots
- Each slot can store a **certificate** and its associated **private key**
- Each slot has a different usage
 - Slot 9a: **Authentication** (system login, SSH, etc.)
 - Slot 9c: Digital signature (emails, documents)
 - Slot 9d: Encryption (emails, documents) - AKA "Key Management"
 - Slot 9e: Physical access (building doors)
 - etc.
- Yubikey emulates a smart card reader with a card always inserted



PIV smart card use case (continued)

- Also stores values in containers
 - Card holder's name
 - Facial picture
 - Biometrics
 - etc.
- Containers are accessed by their ID
 - Example: Facial picture = 0x5fc108
- Card is protected with a PIN, a PUK and a Management Key
 - PIN is required for sign/decrypt operations
 - Management key is required for importing certificates and private keys, setting some values

PIV usage - Pivy

- Install “pivy”
 - <https://github.com/arekinath/pivy>
- Guided install (quick start)

```
$ pivy-tool setup
```

- Will generate a new key pair and self-signed certificate for the 4 main slots, set PIN, etc.
- Sign something

```
$ echo foobar | pivy-tool sign 9a
```

- Load cert into slot 9a

```
$ cat cert.der | pivy-tool write-cert 9a
```

Certificates

[Home](#) / [PIV](#) / Certificates[Authentication](#)[Digital Signature](#)[Key Management](#)[Card Authentication](#)

Authentication (Slot 9a)

Issuer: piv-auth@88D2858F**Subject name:** piv-auth@88D2858F**Expiration date:** 2033-11-04[Delete](#)[Export](#)[Generate](#)[Import](#)[< Back](#)

PIV usage - ykman

- Ykman can also be used to manage PIV
- Also available with a GUI: `ykman-gui`

- Import a private key at a slot

```
$ ykman piv keys import <slot> <private_key_file>
```

- Import a certificate at a slot

```
$ ykman piv certificates import <slot> <certificate_file>
```

- Store/dump photo on your PIV card

```
$ ykman piv objects import 5fc108 photo.jpg
```

```
$ ykman piv objects export 5fc108 out.jpg
```

SSH with PIV smartcard via PKCS #11

- Install opensc

```
$ sudo apt install opensc-pkcs11
$ sudo pacman -S opensc
```

- Get your SSH public key from slot 9a in OpenSSH format

```
$ pivy-tool pubkey 9a
```

- Add the public key to the target SSH server's **authorized_keys** file
- SSH into the machine, using PKCS#11 library path
 - Path may change, on Ubuntu: `/usr/lib/x86_64-linux-gnu/opensc-pkcs11.so`

```
$ ssh -I /usr/lib/opensc-pkcs11.so user@host
```

- To make the change persistent, modify `~/.ssh/config` and add:

```
Host your_ssh_server_hostname
    PKCS11Provider /usr/lib/opensc-pkcs11.so
```

Other use-cases

- We won't go into the details of these, but here are a few other use cases
 - **Unix account login**
 - PAM module for pkcs#11
 - Login to your Linux box using your PIV smart card
 - **OpenVPN**
 - Connect to your VPN using PIV smart card authentication
 - **Wireguard**
 - Same as above but with Wireguard
 - <https://www.procustodibus.com/blog/2023/02/wireguard-yubikey/#piv-slot>

References

- <https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-73-4.pdf>
- <https://ubuntu.com/server/docs/security-smart-cards>
- https://developers.yubico.com/PIV/Guides/SSH_with_PIV_and_PKCS11.html