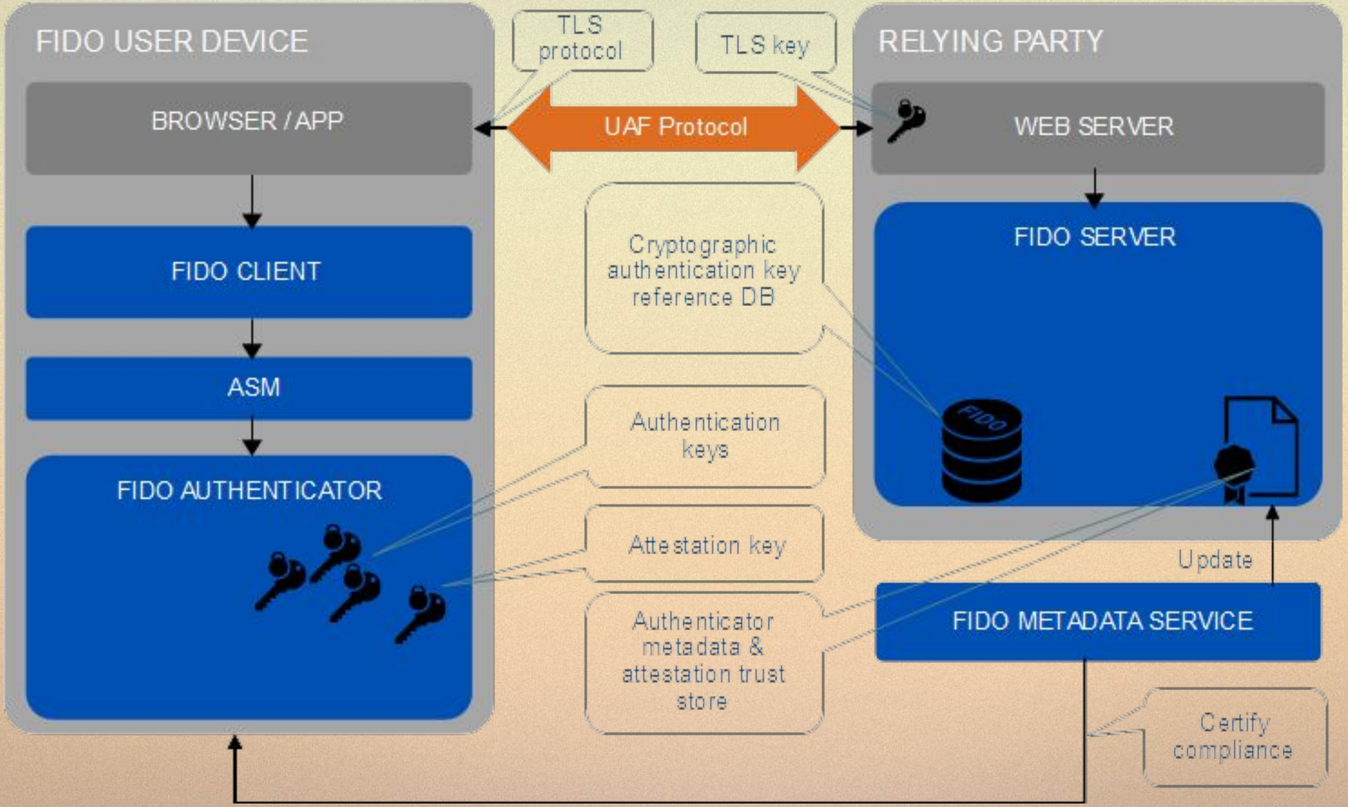
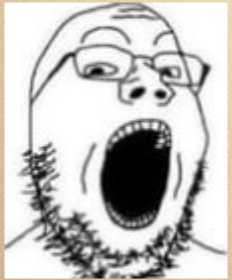
Passkeys : The Good, The Bad, The Ugly

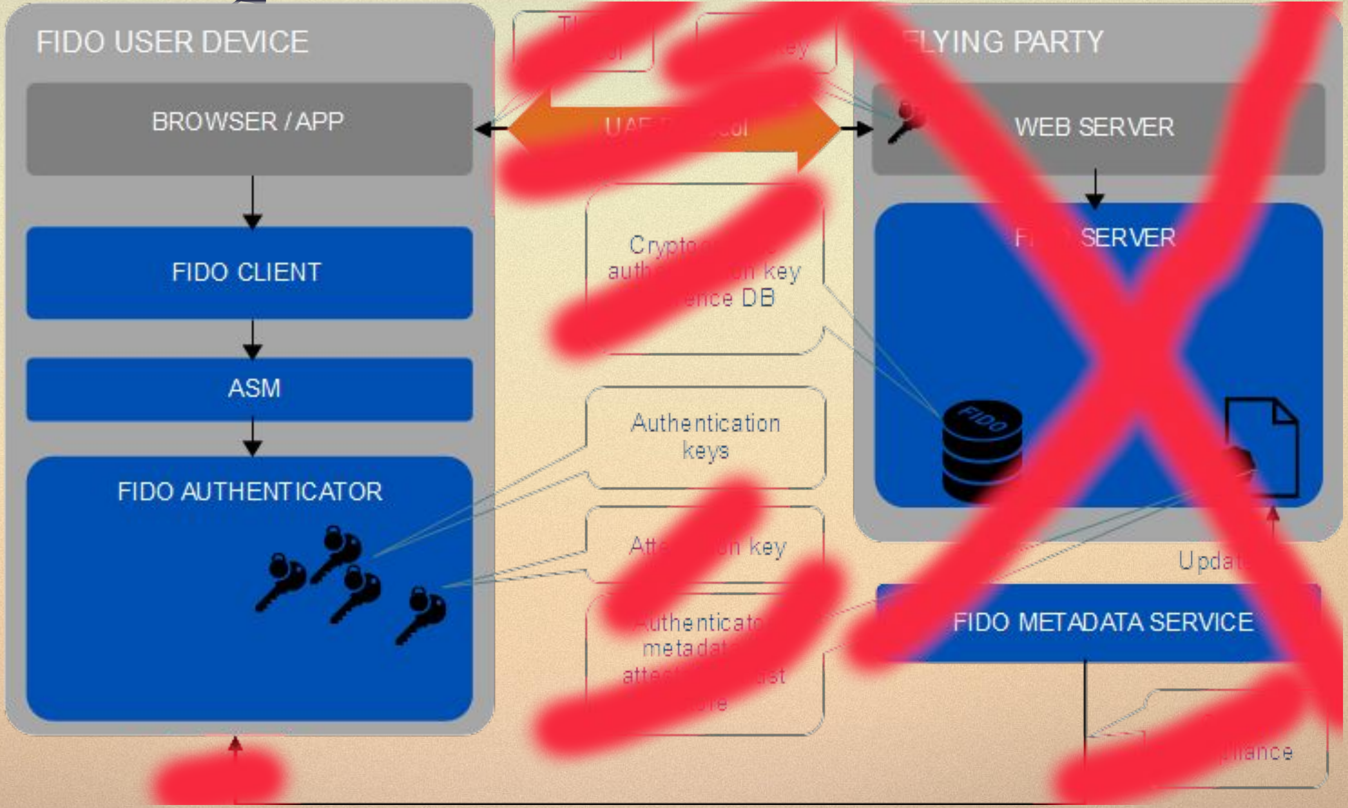
Nicolas Bacca

Why do we care about an IAM protocol ?



FIDO 🐶 for degens





FIDO Protocol in 1 slide

Registration phase

- Verify user presence (biometrics)
- Create a keypair (P256, ed25519 we wish), linked to the requesting web origin
- Return the public key

Authentication phase

- Verify user presence (biometrics)
- Get a challenge
- Return a signature

Recent (~2023) gas optimizations & math wizardry to validate it onchain :
https://github.com/get-smooth/crypto-lib

FIDO is the original abstractoooor : User presence and signer abstraction

God tier UX on mobile with Account Abstraction

Could create a wallet and execute transactions in one click

Very similar experience to web2 (biometrics authentication)

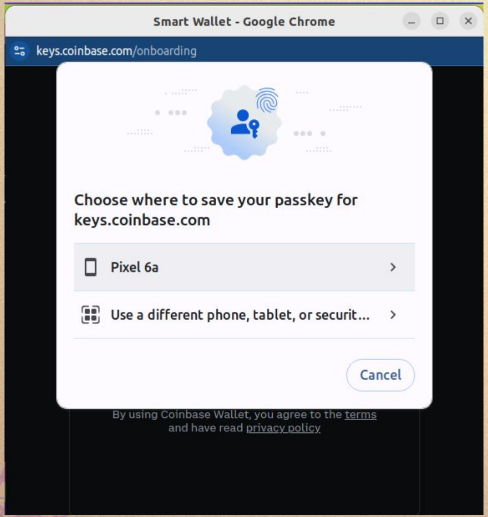
Self custodial

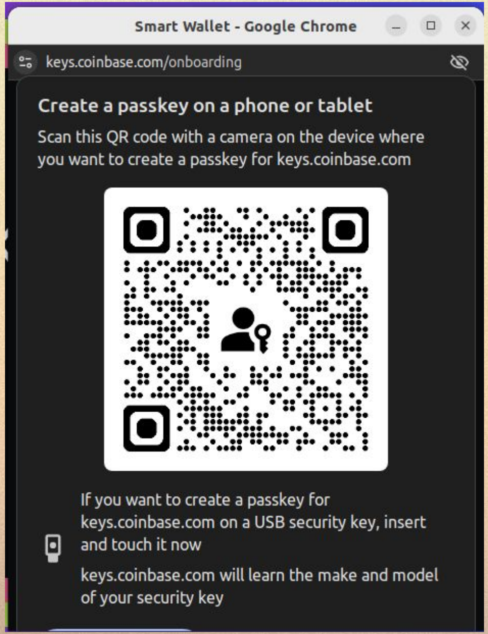
See Coinbase Smart Wallet https://wallet.coinbase.com/smart-wallet

Good enough but standardized experience on desktop

Redirects to mobile with a standard UX (native options with OSX and Windows)

QR pairing, then optional direct connection (caBLE, Google proprietary)





Web developer experience

W3C Standard (WebAuthn)

Registration : navigator.credentials.create

Authentication : navigator.credentials.get

```
const publicKeyCredentialCreationOptions = {
    challenge: Uint8Array.from(
        randomStringFromServer, c => c.charCodeAt(0)),
    rp: {
        name: "Duo Security",
        id: "duosecurity.com",
    },
    user: {
        id: Uint8Array.from(
            "UZSL85T9AFC", c => c.charCodeAt(0)),
        name: "lee@webauthn.guide",
        displayName: "Lee",
    },
    pubKeyCredParams: [{alg: -7, type: "public-key"}],
    authenticatorSelection: {
        authenticatorAttachment: "cross-platform",
    },
    timeout: 60000,
    attestation: "direct"
};

const credential = await navigator.credentials.create({
    publicKey: publicKeyCredentialCreationOptions
});
```

```
const publicKeyCredentialRequestOptions = {
    challenge: Uint8Array.from(
        randomStringFromServer, c => c.charCodeAt(0)),
    allowCredentials: [{
        id: Uint8Array.from(
            credentialId, c => c.charCodeAt(0)),
        type: 'public-key',
        transports: ['usb', 'ble', 'nfc'],
    }],
    timeout: 60000,
}

const assertion = await navigator.credentials.get({
    publicKey: publicKeyCredentialRequestOptions
});
```

Wait a minute ...

A short FIDO history

2013

Creation
U2F,UAF
(mostly
stateless)

Google +
Yubico

2014

Launch

2018

FIDO2

Resident /
Discoverable model

2019

WebAuthn

Android
Strongbox

Android
FIDO2
certified

2020

iOS support

2022

Shared support
announcement
Google/Apple/Microsoft

Passkeys

A short Passkeys history

Proprietary syncable credentials deployment

External password manager support (iOS 17, Android 14)

General Passkey rebranding

Specification for Passkeys synchronization

2023

2024

Standard nerd moment : what is a syncable credential ?

Not specified in the standard authenticator discovery (spoiler, correct name is "multi-device credential")

A discoverable/resident credential in the context of a platform (smartphone) authenticator

Only truly defined in the response

flags	1	• Bit 3: Backup Eligibility (***BE***). ∘ `1` means the public key credential source is backup eligible. ∘ `0` means the public key credential source is not backup eligible. • Bit 4: Backup State (***BS***). ∘ `1` means the public key credential source is currently backed up. ∘ `0` means the public key credential source is not currently backed up.

A tale of multiple security misconceptions

FIDO protects against phishing, not malware

- Origin check performed in software

On the other hand, FIDO was started as a strongly secure hardware backed standard

- Initially on dedicated devices
- Moved to smartphones only when the right hardware was available (Android Strongbox)

What are the impacts of introducing syncable credentials ?

Crypto people proudly make things even more confusing

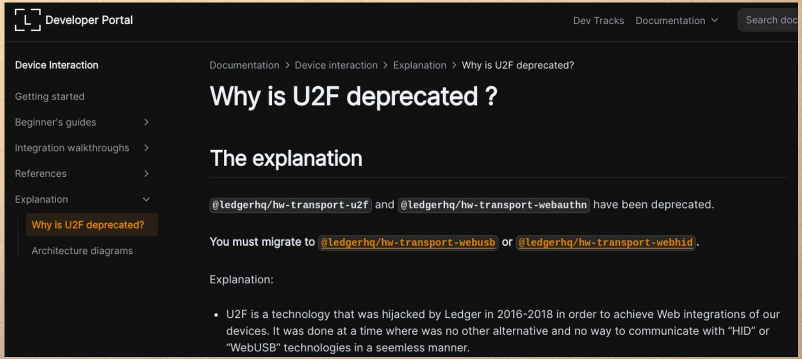
FIDO is designed for authentication only

- Not possible to “clear sign” transactions, by design
- Impact of having a key less secure than expected is way more critical for us

FIDO is designed to isolate web origins

- A multi dapps web wallet needs to break this creatively (pop-up, iframe ...)

Still not our worst though, if you like tunneling



Interlude : why secure hardware is important

Protects against malware

Not possible to access the keys from software

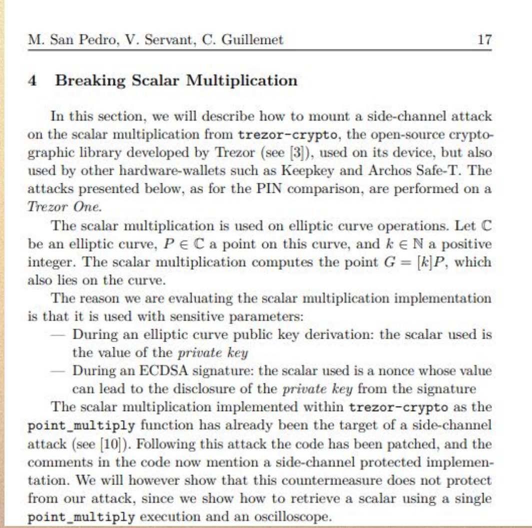
Could still be used as a signing oracle, depending on the implementation

Protects against active physical attacks

Either performed by a physical attacker or remotely, enabled by software (voltage/clock glitching)

Protects against passive physical attacks (probe / speculative)

Not possible to achieve without dedicated hardware, best OSS software gives you constant time, not “constant everything” (SSTIC 2019 on libsecp256k1 on STM32 by Ledger Donjon)



M. San Pedro, V. Servant, C. Guillemet 17

4 Breaking Scalar Multiplication

In this section, we will describe how to mount a side-channel attack on the scalar multiplication from `trezor-crypto`, the open-source cryptographic library developed by Trezor (see [3]), used on its device, but also used by other hardware-wallets such as Keepkey and Archos Safe-T. The attacks presented below, as for the PIN comparison, are performed on a *Trezor One*.

The scalar multiplication is used on elliptic curve operations. Let $\mathbb{C}$ be an elliptic curve, $P \in \mathbb{C}$ a point on this curve, and $k \in \mathbb{N}$ a positive integer. The scalar multiplication computes the point $G = [k]P$, which also lies on the curve.

The reason we are evaluating the scalar multiplication implementation is that it is used with sensitive parameters:

— During an elliptic curve public key derivation: the scalar used is the value of the *private key*

— During an ECDSA signature: the scalar used is a nonce whose value can lead to the disclosure of the *private key* from the signature

The scalar multiplication implemented within `trezor-crypto` as the `point_multiply` function has already been the target of a side-channel attack (see [10]). Following this attack the code has been patched, and the comments in the code now mention a side-channel protected implementation. We will however show that this countermeasure does not protect from our attack, since we show how to retrieve a scalar using a single `point_multiply` execution and an oscilloscope.

Non syncable credentials security model

Keys cannot be extracted by malware

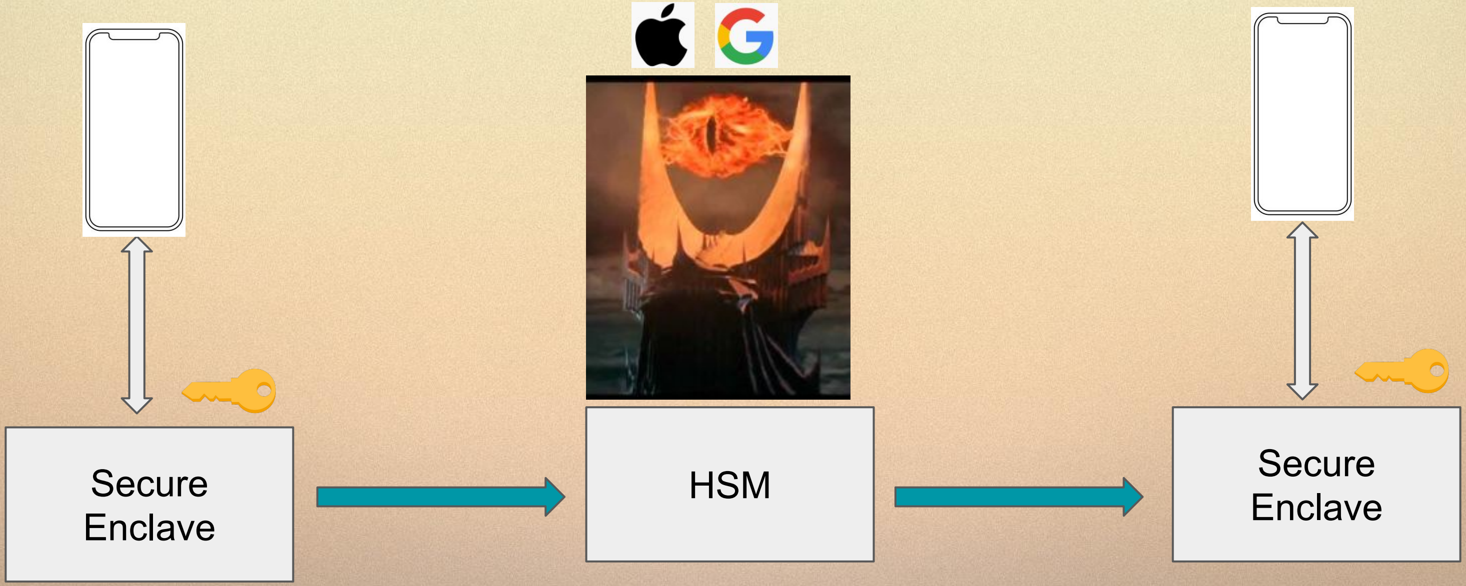
Authentication is enforced by the enclave and necessary for each key operation

A kernel level malware could change the authentication prompt and fool the user into signing arbitrary data

Good synchronization hypothesis

Enclave to enclave synchronization

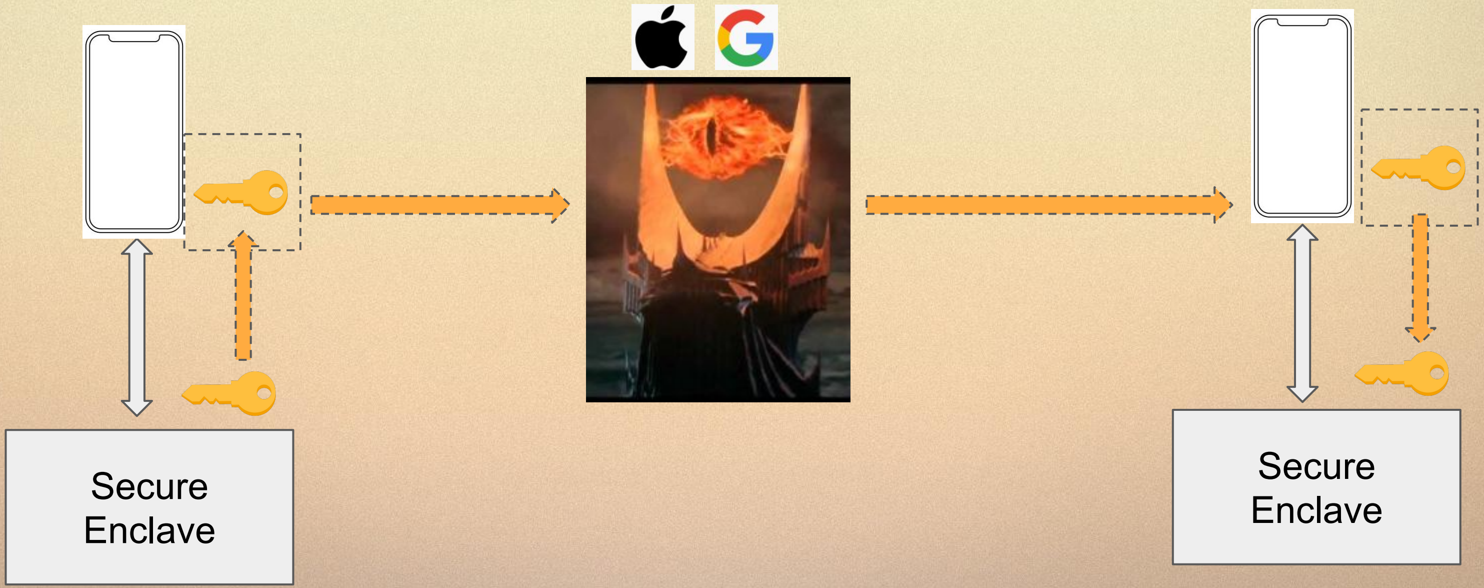
Same security model as non syncable credentials on a similar platform (supposing user synchronization credentials aren't compromised)



Bad synchronization hypothesis

Key is in the enclave but can move to the application platform to synchronize

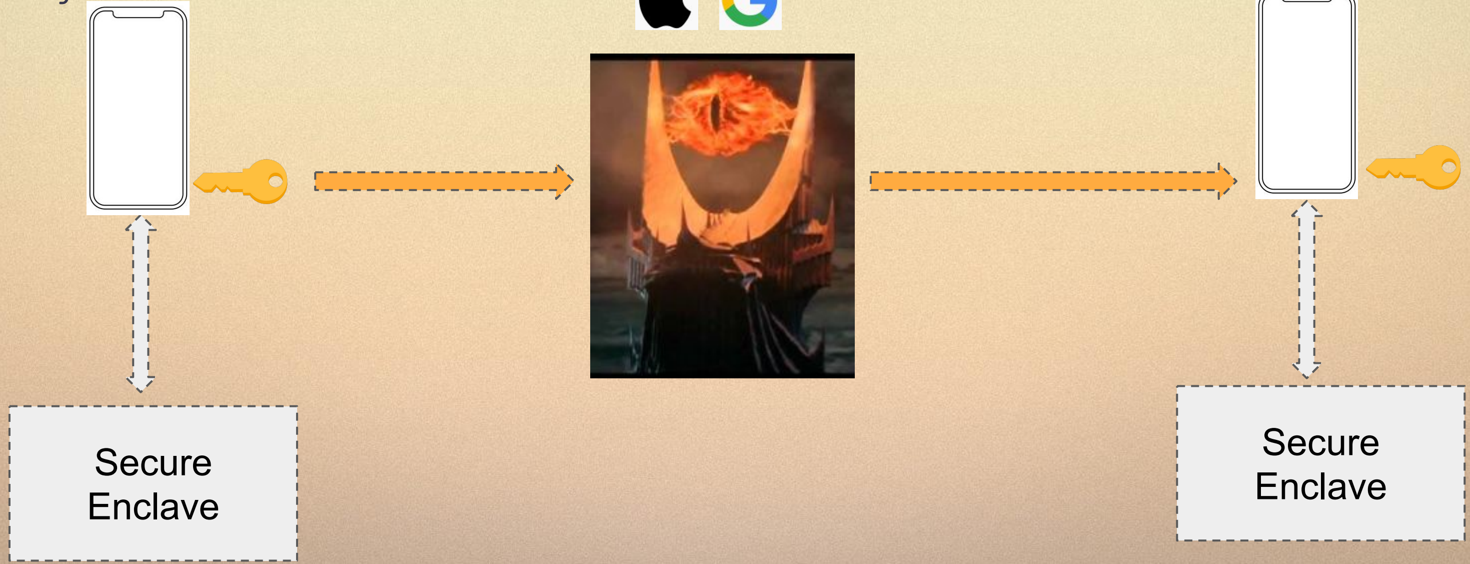
A kernel level malware can steal the key after forcing a synchronization



Ugly synchronization hypothesis

Key is in the application platform and handled like a password, yolo

A (kernel level) malware can steal the key - Secure Enclave might or not require an authentication for each key access



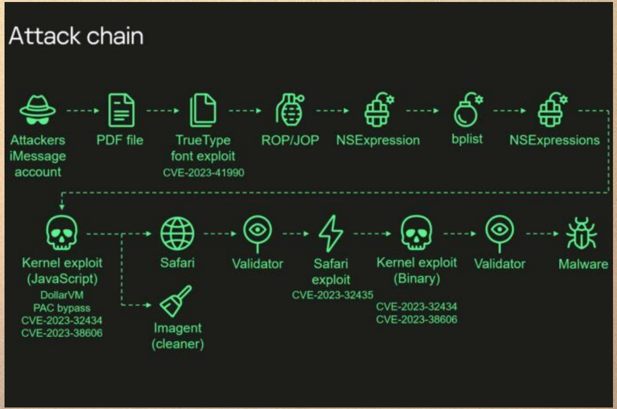
Assessing iOS implementation

Want to act as a malware

Latest iOS versions are quite complex to jailbreak

fun watch “Operation Triangulation” https://www.youtube.com/watch?v=1f6YyH62jFE

We can cheat and synchronize to a more vulnerable one as an acceptable proxy



checkm8

Non fixable bootrom vulnerability from A5 to A11 (iPhone 8, 8+, X) in DFU mode

Latest implementations ~ iOS 16, iPadOS 18.1 on gen 7 (2019)

Multiple implementations using this exploit (used palera1n https://palera.in/)

Tricky to get working, check your cable (USB A) and USB controller (only USB 2 worked for me)

Keychain implementation

Availability	File data protection	Keychain data protection
When unlocked	NSFileProtectionComplete	kSecAttrAccessibleWhenUnlocked
While locked	NSFileProtectionComplete UnlessOpen	
After first unlock	NSFileProtectionComplete UntilFirstUserAuthentication	kSecAttrAccessibleAfterFirstUnlock
Always	NSFileProtectionNone	kSecAttrAccessibleAlways
Passcode enabled		kSecAttrAccessibleWhen PasscodeSetThisDeviceOnly

Local keychain and iCloud keychain merged on iOS

High level format documented by Apple

- https://support.apple.com/fr-fr/guide/security/secb0694df1a/web
- First oops, everything stays accessible after an unlock

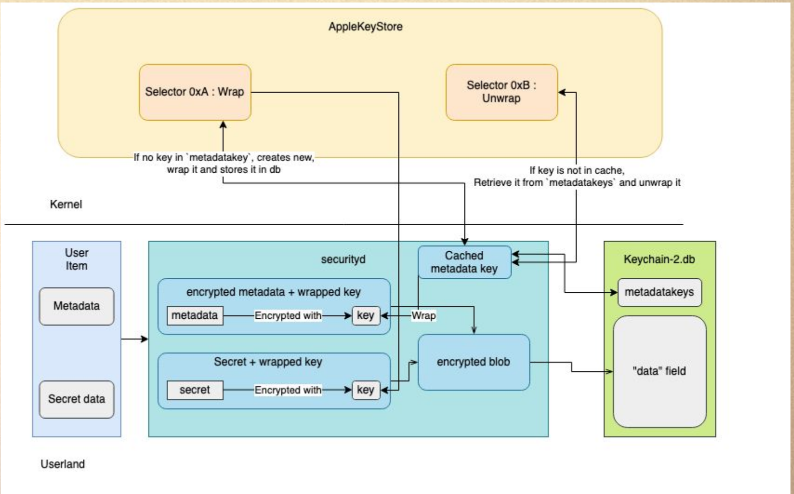
Dumping the keychain

Practical (outdated) implementation, version 8 vs version 7

https://github.com/xperylabhub/ios_keychain_decrypter/tree/master

Wrapping scheme also documented here

https://shindan.io/posts/keychain_module_analysis



ItemV7 vs ItemV8

```
syntax = "proto2";

message SecDbKeychainSerializedItemV7 {
    required bytes encryptedSecretData = 1;
    required bytes encryptedMetadata = 2;

    enum Keyclass {
        KEYCLASS_AK = 6;
        KEYCLASS_CK = 7;
        KEYCLASS_DK = 8;
        KEYCLASS_AKU = 9;
        KEYCLASS_CKU = 10;
        KEYCLASS_DKU = 11;
        KEYCLASS_AKPU = 12;
    }
    required Keyclass keyclass = 3 [default = KEYCLASS_AKPU];
}
```

Analysis with protoc –decode_raw and messing around

```
syntax = "proto2";

message KeyReference {
        required bytes wrappedKey = 1;
        optional int32 rfu = 2;
}

message SecretData {
        required bytes encryptedData = 1;
        required KeyReference keyReference = 2;
        required string tamperCheck = 3;
}

message EncryptedMetadata {
        required bytes encryptedMetadata = 1;
        required bytes encryptedMetadataKey = 2;
        required string tamperCheck = 3;
}

message ItemV8 {
        required SecretData secretData = 1;
        required EncryptedMetadata encryptedMetadata = 2;

    enum Keyclass {
        KEYCLASS_AK = 6;
        KEYCLASS_CK = 7;
        KEYCLASS_DK = 8;
        KEYCLASS_AKU = 9;
        KEYCLASS_CKU = 10;
        KEYCLASS_DKU = 11;
        KEYCLASS_AKPU = 12;
    }
    required Keyclass keyclass = 3 [default = KEYCLASS_AKPU];
}
```

Decrypted item


```
{'musr': b'', 'asen': '0', 'crtr': '0', 'decr': '1', 'drve': '1', 'encr': '0', 'extr': '1', 'kcls': '1', 'modi': '1', 'next': '0', 'perm': '1', 'priv': '1', 'sens': '0', 'sign': '1', 'snrc':
'0', 'sync': '1', 'tomb': '0', 'type': '73', 'unwp': '1', 'vrfy': '0', 'vyrc': '0', 'wrap': '0', 'bsiz': '256', 'esiz': '256', 'pdmn': 'ak', 'edat': '20010101000000Z', 'sdat': '20010101000000Z',
'labl': 'learnpasskeys.io', 'klbl': b'\x01\\\xa4\xa5\xf0?\xba\xb9\x89\xfd\xd7N\xa4f\xb2\x1e\x89\x82P\x05', 'sha1': b'\x82\x13\x9b\xf0\xd6\x06\x91\x99\xe9\xd5\x95\xb1\xa1\xc3|U\x91\x14\x8c<',
'cdat': '20241103171115.258359Z', 'mdat': '20241103171115.258359Z', 'persistref': b'\xee\xcb\x83\xde\x95;Hv\xa6?\xf3\x1d\xca\xe1f\xb5', 'agrp': 'com.apple.webkit.webauthn', 'SecAccessControl':
b'1\x0c0\n\x0c\x04prot\x0c\x02ak', 'UUID': '03A2A3D5-1F15-487F-96B7-B44D54A23E61', 'TamperCheck': 'B97AA569-876B-4A15-808D-264314A004CA', 'atag': b'\xa3bidX z0yExDWe6ME6Hsw-8ndoZwWZKt-
Gw6xIdnamesoptimal.hester.9580kdisplayNamelHester Bosco'}
{'TamperCheck': 'B97AA569-876B-4A15-808D-264314A004CA', 'v_Data':
b'\x04@\tB\x90\xc2\xb7x&\x8f\xb4#:\x97\xe3\x9dQl\x08@8i\x0fl\xab\x129\x1e\xdc\x91\xc9\x8a\xafb\xafXk\xf38P\xb5\x0b\x0f1\x8e\x9fJcW\xea\x05\x96\xe1\xca\nO\xd3\xcc\xa6XE\xeb\x01\x7fI\xf6\xca\xf0\x
df\x81\x90\xd0g\x8c\x97ROU\x1c\xf9\xef\xe41nh\xbe\x8f\xbd\x13q\xa8M\xe9U`u\x84'}
```


v_Data length is 65 + 32, starts with 04, what could it be ...

```
Public key 0440094290c2b778268fb4233a97e39d516c084038690f6cab12391edc91c98aaf62af586bf33850b50b0f318e9f4a6357ea0596e1ca0a4fd3cca65845eb017f49
Private key f6caf0df8190d0678c97524f551cf9efe4316e68be8fbd1371a84de955607584
```

CREDENTIAL PUBLIC KEY


```
-----BEGIN PUBLIC KEY-----
MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEQAlCkMK3eCaPtCM6l+OdUWwIQDhp
D2yrEjke3JHJiq9ir1hr8zhQtQsPMY6fSmNX6gWW4coKT9PMplhF6wF/SQ==
-----END PUBLIC KEY-----
```


```
nba@Carbonouvo:~/Downloads$ openssl asn1parse -inform pem -in /tmp/pubkey.asc -d
ump
    0:d=0  hl=2 l=  89 cons: SEQUENCE
    2:d=1  hl=2 l=  19 cons: SEQUENCE
    4:d=2  hl=2 l=   7 prim: OBJECT            :id-ecPublicKey
   13:d=2  hl=2 l=   8 prim: OBJECT            :prime256v1
   23:d=1  hl=2 l=  66 prim: BIT STRING
      0000 - 00 04 40 09 42 90 c2 b7-78 26 8f b4 23 3a 97 e3   ..@.B...x&..#:..
      0010 - 9d 51 6c 08 40 38 69 0f-6c ab 12 39 1e dc 91 c9   .Ql.@8i.l..9....
      0020 - 8a af 62 af 58 6b f3 38-50 b5 0b 0f 31 8e 9f 4a   ..b.Xk.8P...1..J
      0030 - 63 57 ea 05 96 e1 ca 0a-4f d3 cc a6 58 45 eb 01   cW......O...XE..
      0040 - 7f 49                                             .I
```

Now let’s do Android

Jailbreaking is a given freedom with the right hardware (use Pixel phones please)
Unlock bootloader + Magisk (https://github.com/topjohnwu/Magisk), done

Can then trace applications with Frida https://frida.re/

Frida current release 16.5.6 is not stable on Android 15, hopefully fixed soon
Build from source with https://github.com/frida/frida-java-bridge/pull/337 (how to build with an out of release bridge : https://github.com/frida/frida/issues/2958#issuecomment-2381703028)

Where to start from ?

Apple Passkey architecture is well defined, Google less so

Open a browser, start authenticating, look at the logs (adb logcat)

```
11-04 12:11:38.576  2896  2896 D BoundBrokerSvc: onBind: Intent { act=com.google
.android.gms.fido.fido2.zeroparty.START dat=chimera-action:/... cmp=com.google.a
ndroid.gms/.chimera.GmsBoundBrokerService }
11-04 12:11:38.576  2896  2896 D BoundBrokerSvc: Loading bound service for inten
t: Intent { act=com.google.android.gms.fido.fido2.zeroparty.START dat=chimera-ac
tion:/... cmp=com.google.android.gms/.chimera.GmsBoundBrokerService }
```

Google Mobile Services seems to be a good start, let’s instrument it

Instrumentation code

```
console.log("frida-dump");

Java.perform(() =>{
        var signature = Java.use("java.security.Signature");
        signature.getInstance.overload('java.lang.String').implementation = function (var0) {
            console.log("[*] Signature.getInstance called with algorithm: " + var0 + "\n");
            return this.getInstance(var0);
        };

        signature.initSign.overload("java.security.PrivateKey").implementation = function(privateKey) {
            console.log("Signature key " + privateKey.$className);
            return signature.initSign.overload("java.security.PrivateKey").call(this, privateKey);
        };
});
```

Investigating GMS

Hook the right process (some guessing involved) and let's have a look

```
(snake3-frida) nba@Carbonouvo:~/android$ frida-ps -U |grep gms
2896  com.google.android.gms

2377  com.google.android.gms.persistent

7601  com.google.android.gms.ui

4382  com.google.android.gms.unstable
```

```
(snake3-frida) nba@Carbonouvo:~/android$ frida -U 7601 -l frida-dump-mini.js
     ____
    / _  |   Frida 16.5.6 - A world-class dynamic instrumentation toolkit
   | (_| |
    > _  |   Commands:
   /_/ |_|       help      -> Displays the help system
   . . . .       object?   -> Display information about 'object'
   . . . .       exit/quit -> Exit
   . . . .
   . . . .   More info at https://frida.re/docs/home/
   . . . .
   . . . .   Connected to Pixel 6a (id=36221JEGR19406)
Attaching...
frida-dump
[Pixel 6a::PID::7601 ]-> %resume
[Pixel 6a::PID::7601 ]-> [*] Signature.getInstance called with algorithm: SHA256
withECDSA

Signature key com.google.android.gms.org.conscrypt.OpenSSLECPrivateKey
[Pixel 6a::PID::7601 ]->
```

Trying to get the private key

Class described here
https://android.googlesource.com/platform/libcore/+/96b54bb/crypto/src/main/java/org/conscrypt/OpenSSLECPrivateKey.java

Generic wrapper so it could still be hardware based ...

More instrumentation code

```
console.log("frida-dump");

Java.perform(() =>{

    const OpenSSLECPrivateKey = Java.use("com.google.android.gms.org.conscrypt.OpenSSLECPrivateKey");

    // Signature
        var signature = Java.use("java.security.Signature");
        signature.getInstance.overload('java.lang.String').implementation = function (var0) {
            console.log("[*] Signature.getInstance called with algorithm: " + var0 + "\n");
            return this.getInstance(var0);
        };

        signature.initSign.overload("java.security.PrivateKey").implementation = function(privateKey) {
            console.log("Signature key " + privateKey.$className);
            if (privateKey.$className == "com.google.android.gms.org.conscrypt.OpenSSLECPrivateKey") {
                var privateKey2 = Java.cast(privateKey, OpenSSLECPrivateKey);
                console.log(privateKey2.getS());
            }
            return signature.initSign.overload("java.security.PrivateKey").call(this, privateKey);
        };

});
```

And we get the key

```
[*] Signature.getInstance called with algorithm: SHA256withECDSA

Signature key com.google.android.gms.org.conscrypt.OpenSSLECPrivateKey
14264183326736513129348188007383962361540824746794863684841211464835620388619
```

CREDENTIAL PUBLIC KEY


```
-----BEGIN PUBLIC KEY-----
MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEk+PyjirDu42PuZ27VUSL7MasQuSX
aiAPP49ybCXYTzsYvCDBmhbMcUo6odbnUOxvSzofvqESaX9bjNOIjdFNDQ==
-----END PUBLIC KEY-----
```

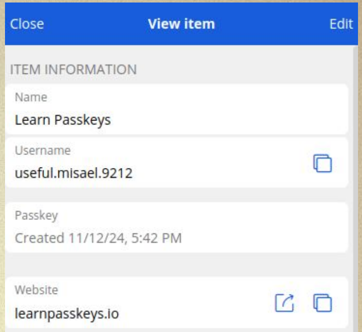

```
(snake3-frida) nba@Carbonouvo:~/android$ openssl asn1parse -in /tmp/pubkey.asc -
inform pem -dump
    0:d=0  hl=2 l=  89 cons: SEQUENCE
    2:d=1  hl=2 l=  19 cons: SEQUENCE
    4:d=2  hl=2 l=   7 prim: OBJECT            :id-ecPublicKey
   13:d=2  hl=2 l=   8 prim: OBJECT            :prime256v1
   23:d=1  hl=2 l=  66 prim: BIT STRING
      0000 - 00 04 93 e3 f2 8e 2a c3-bb 8d 8f b9 9d bb 55 44   ......*.......UD
      0010 - 8b ec c6 ac 42 e4 97 6a-20 0f 3f 8f 72 6c 25 d8   ....B..j .?.rl%.
      0020 - 4f 3b 18 bc 20 c1 9a 16-cc 71 4a 3a a1 d6 e7 50   O;.. ....qJ:...P
      0030 - ec 6f 4b 3a 1f be a1 12-69 7f 5b 8c d3 88 8d d1   .oK:....i.[.....
      0040 - 4d 0d                                             M.
```

```
>>> from Crypto.PublicKey import ECC
>>> import binascii
>>> key = ECC.construct(d=142641833267365131293481880073839623615408247467948636
84841211464835620388619, curve="P-256")
>>> binascii.hexlify(key.public_key().export_key(format="raw"))
b'0493e3f28e2ac3bb8d8fb99dbb55448becc6ac42e4976a200f3f8f726c25d84f3b18bc20c19a16
cc714a3aa1d6e750ec6f4b3a1fbea112697f5b8cd3888dd14d0d'
```

Bonus

We get the key before the additional one time device screen lock validation

External password manager



No expectations, no disappointment - export with Bitwarden CLI


```
ogin":{"fido2Credentials":[{"credentialId":"8385b83a-cb9a-49a0-aab0-a5bc0c259198
","keyType":"public-key","keyAlgorithm":"ECDSA","keyCurve":"P-256","keyValue":"M
IGHAgEAMBMGByqGSM49AgEGCCqGSM49AwEHBG0wawIBAQQgIlF3-OWQZRAnGykiyL4XzBvLN80xK8JQa
Ys5fWAaGfyhRANCAARZeRGfDjwzkg7VsmjneBCgWextdly--OPkUNRsz1D6-X3xw4mCwZl2gsG7SaEVE
EHFPw_jNgoQb-x-n8qtFbti","rpId":"learnpasskeys.io","userHandle":"ajFWd2NLWlgzM0h
```


CREDENTIAL PUBLIC KEY

```
-----BEGIN PUBLIC KEY-----
MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEWXkRnw48M5IO1bJo53gQoFnsbXZc
vvjj5FDUbM9Q+vl98cOJgsGZdoLBu0mhFRBBxT8P4zYKEG/sfp/KrRW7Yg==
-----END PUBLIC KEY-----
```

Base64 URL decoding of keyValue

```
PrivateKeyInfo SEQUENCE (3 elem)
  version Version INTEGER 0
  privateKeyAlgorithm AlgorithmIdentifier SEQUENCE (2 elem)
    algorithm OBJECT IDENTIFIER 1.2.840.10045.2.1 ecPublicKey (ANSI X9.62 public key type)
    parameters ANY OBJECT IDENTIFIER 1.2.840.10045.3.1.7 prime256v1 (ANSI X9.62 named ellipt
  privateKey PrivateKey OCTET STRING (109 byte) 306B020101042022517 7F8E5906510271B2922C8BE17CC
    SEQUENCE (3 elem)
      INTEGER 1
      OCTET STRING (32 byte) 225177F8E5906510271B2922C8BE17CC1BCB37CD312BC250698B397D601A19FC
      [1] (1 elem)
        BIT STRING (520 bit) 0000010001011001011110010001000110011111000011100011110000110011l
```

```
nba@Carbonouvo:/tmp/blah$ openssl asn1parse -in key.bin -inform der -dump
    0:d=0  hl=3 l= 135 cons: SEQUENCE
    3:d=1  hl=2 l=   1 prim: INTEGER           :00
    6:d=1  hl=2 l=  19 cons: SEQUENCE
    8:d=2  hl=2 l=   7 prim: OBJECT            :id-ecPublicKey
   17:d=2  hl=2 l=   8 prim: OBJECT            :prime256v1
   27:d=1  hl=2 l= 109 prim: OCTET STRING
      0000 - 30 6b 02 01 01 04 20 22-51 77 f8 e5 90 65 10 27   0k.... "Qw...e.'
      0010 - 1b 29 22 c8 be 17 cc 1b-cb 37 cd 31 2b c2 50 69   .)"......7.1+.Pi
      0020 - 8b 39 7d 60 1a 19 fc a1-44 03 42 00 04 59 79 11   .9}`....D.B..Yy.
      0030 - 9f 0e 3c 33 92 0e d5 b2-68 e7 78 10 a0 59 ec 6d   ..<3....h.x..Y.m
      0040 - 76 5c be f8 e3 e4 50 d4-6c cf 50 fa f9 7d f1 c3   v\....P.l.P..}..
      0050 - 89 82 c1 99 76 82 c1 bb-49 a1 15 10 41 c5 3f 0f   ....v...I...A.?.
      0060 - e3 36 0a 10 6f ec 7e 9f-ca ad 15 bb 62            .6..o.~.....b
```

```
>>> from Crypto.PublicKey import ECC
>>> import binascii
>>> key = ECC.construct(d=0x225177f8e5906510271b2922c8be17cc1bcb37cd312bc250698b
397d601a19fc, curve="P-256")
>>> binascii.hexlify(key.public_key().export_key(format="raw"))
b'045979119f0e3c33920ed5b268e77810a059ec6d765cbef8e3e450d46ccf50faf97df1c38982c1
997682c1bb49a1151041c53f0fe3360a106fec7e9fcaad15bb62'
```

This is where we are on smartphones

Syncable passkeys are handled by the Application Processor
thus vulnerable to malware extraction when the device is unlocked
as well as (passive) physical attacks

User presence enforcement is not tightly linked to usage
of the key for syncable passkeys

Non syncable passkeys (at the moment, non resident FIDO creds)
are fully protected by a Secure Enclave

How did we get there ? (just speculating)

Conflicting rules between vendors for key import into an enclave

- Can't encode preexisting keys. You must use the Secure Enclave to create the keys. Not having a mechanism to transfer plain-text key data into or out of the Secure Enclave is fundamental to its security.

All agree for no export

Import encrypted keys into secure hardware

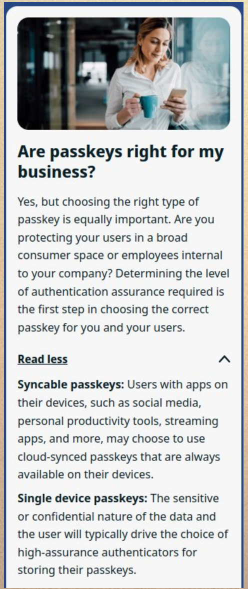
Android 9 (API level 28) and higher lets you import encrypted keys securely into the keystore using an ASN.1-encoded key format. The Keymaster then decrypts the keys in the keystore, so the content of the keys never appears as plaintext in the device's host memory. This process provides additional key decryption security.

★ **Note:** This feature is supported only on devices that ship with Keymaster 4 or higher.

There have been signs...

No attestation provided for syncable Passkeys (higher MITM risk during the registration phase)

Enterprise narrative pushed by FIDO Alliance members



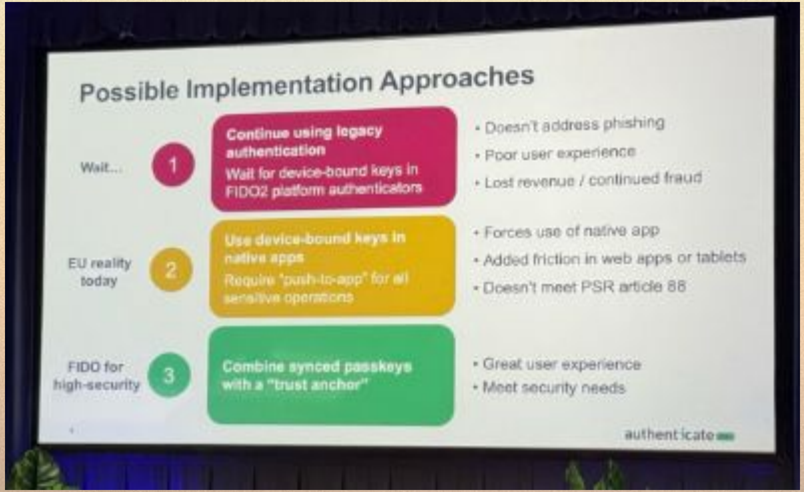
Are passkeys right for my business?

Yes, but choosing the right type of passkey is equally important. Are you protecting your users in a broad consumer space or employees internal to your company? Determining the level of authentication assurance required is the first step in choosing the correct passkey for you and your users.

Read less

Syncable passkeys: Users with apps on their devices, such as social media, personal productivity tools, streaming apps, and more, may choose to use cloud-synced passkeys that are always available on their devices.

Single device passkeys: The sensitive or confidential nature of the data and the user will typically drive the choice of high-assurance authenticators for storing their passkeys.



A brighter future ?

Draft “Credential Exchange” protocol pushed by the FIDO Alliance

  https://fidoalliance.org/specifications-credential-exchange-specifications/

  Doesn’t cover how credentials are used once synchronized

Trust anchor : linking a device bound passkey to a syncable passkey

  https://github.com/w3c/webauthn/issues/2075

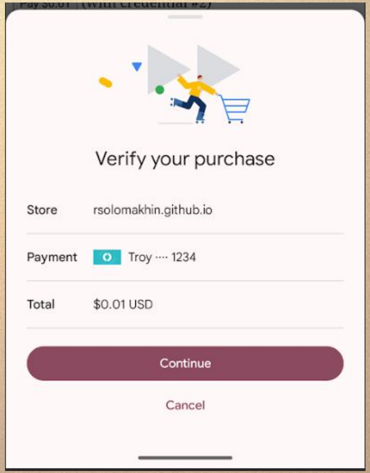
  devicePubKey extension superseded by supplementalPubKey then dropped

Additional confirmation information in passkey UI

  txAuthSimple / txAuthGeneric extensions dropped

  Secure Payment Confirmation is stalling on Safari

  https://www.w3.org/TR/secure-payment-confirmation/



Disappointed by platform authenticators ?

Popular hardware wallets can be used as backupable (BIP 39) cross platform authenticators

Ledger (also NFC with Flex/Stax yay) https://github.com/LedgerHQ/app-security-key

Trezor https://github.com/trezor/trezor-firmware/tree/main/core/src/apps/webauthn

SoloKey / Nitrokey (USB, NFC)

https://github.com/trussed-dev/fido-authenticator

Java Card (PC/SC, NFC)

https://github.com/BryanJacobs/FIDO2Applet

State of onchain Passkey support

Movement pioneered by Cartridge on Starknet

Optimizations allowing to verify P-256 Passkeys signature efficiently, RIP 7212, RIP 7696

Also precompile independent library https://github.com/get-smooth/crypto-lib

Deployed in production by several teams : Safe{Core}, Metamask Delegation Toolkit, Coinbase Smart Wallet, Soul Wallet, ZeroDev ...

Some more opinionated than others regarding Passkey handling

End of 20 mins talk AA kernels comparison

ZeroDev

AA native
Validator permissions oriented
Passkey as standard validator

Safe{Core}

4337 module
Multisignature
Passkey only supported but discouraged

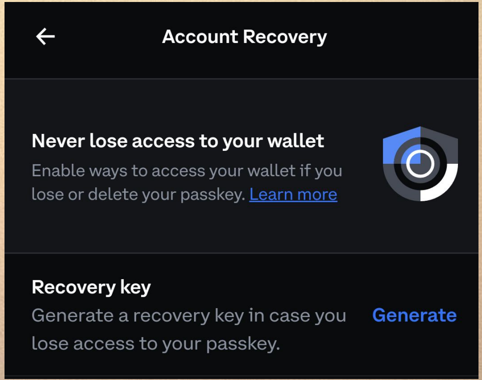
Passkey 4337 Support

This directory contains additional support contracts for using passkeys with Safes over ERC-4337.

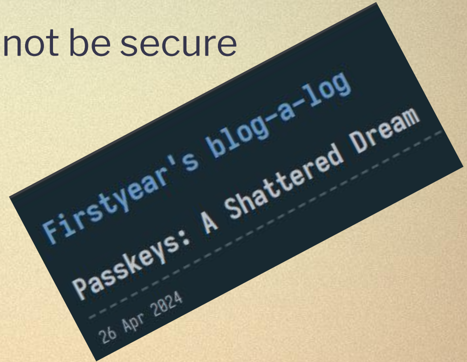
> These contracts are only needed when deploying Safes with initial passkey owners that are required for verifying the very first ERC-4337 user operation with `initCode`. We do not, however, recommend this as it would tie your Safe account's address to a passkey which may not be always available. In particular: the WebAuthn authenticator that stores a device-bound credential that does not allow for backups may be lost, the domain the credential is tied to may no longer be available, you lose access to the passkey provider where your WebAuthn credentials are stored (for example, you no longer have an iPhone or MacBook with access to your iCloud keychain passkeys), etc.
>
> **As such, for the moment, we recommend that Safes be created with an ownership structure or recovery mechanism that allows passkey owners to be rotated in case access to the WebAuthn credential is lost.**

Coinbase Smart Wallet

AA native
All signers equivalent
Multiple passkeys and ECDSA signers



Should we drop passkeys ?



Storing and using keys like passwords is convenient to synchronize but might not be secure enough for all crypto use cases

Rift between consumer and enterprise passkeys is confusing

Still hard to pass on the magic onboarding feeling

“Training wheels” for syncable passkeys, enforced by the smart account (check the Backup Eligibility bit during registration, positive+negative test, or the lack of attestation)

Know your threat model, code accordingly

One last for the road

Thank you!

Nicolas Bacca

@btchip