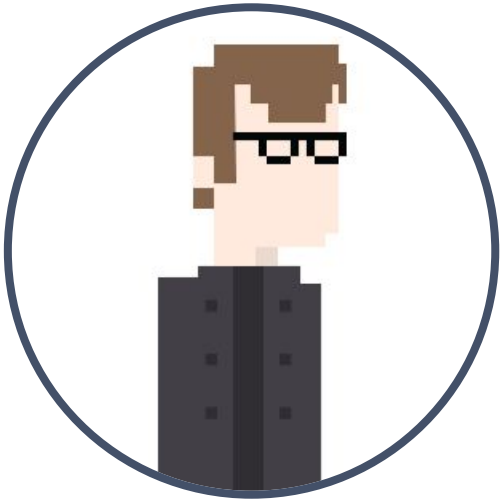
WebAuthn Development

Demystifying the Hard Parts

whoamwe

Nick Steele

Principal Researcher at Gemini
@codekaiju



James Barclay

Senior Security Engineer at Cruise
@futureimperfect

Why are we here?

- Began working on WebAuthn at Duo Security back in 2017
  - Wrote one of the first WebAuthn/FIDO2 servers
- Developed WebAuthn support/features at Duo Security & Cisco
- Started asking a lot of questions about how best to implement the framework
- Got involved with The FIDO Alliance and W3C to answer some of those questions
- Wrote the first Python and Go WebAuthn libraries


BUICK
fido™

What are we talking about?

This is not a talk about WebAuthn/FIDO2 in the enterprise.

This is talk about your customers.

This is not a talk about how we did it.

(at least not entirely)

This is a talk about how
you can do it.

Why are *you* here?

- You're a developer or security person thinking about implementing FIDO2/WebAuthn for your business
  - But you have 500 other things to do…

- You want to have the best security for your customers and employees
  - But your security team is 0 to 10 people

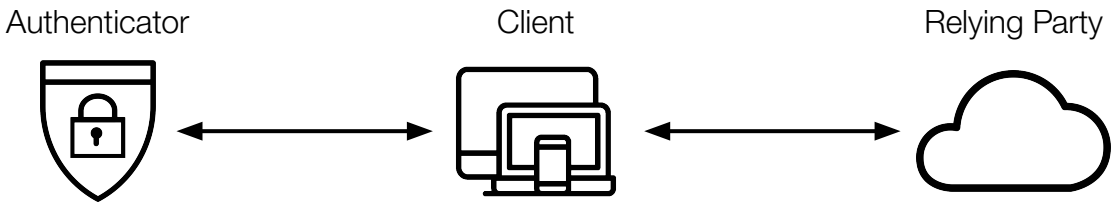
- You want to know where you're likely to have problems
  - But don't know how to define those problems

What we'll cover

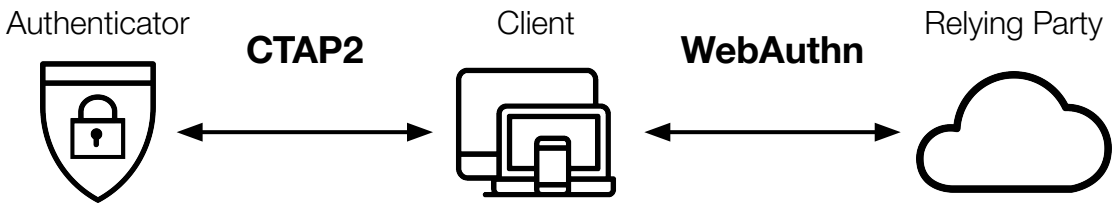
- Terminology
  - To help define the problems
- Things you should know about
  - But not necessarily care about
- Things you might care about
  - And where they may cause problems
- How to get started with WebAuthn and FIDO2
  - Tips, tricks, and planning for **the hard parts**

Terminology

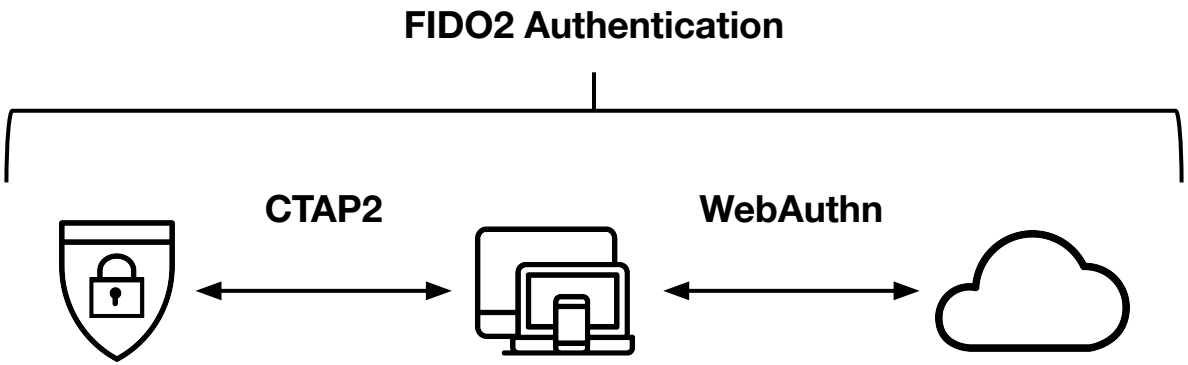
Terminology



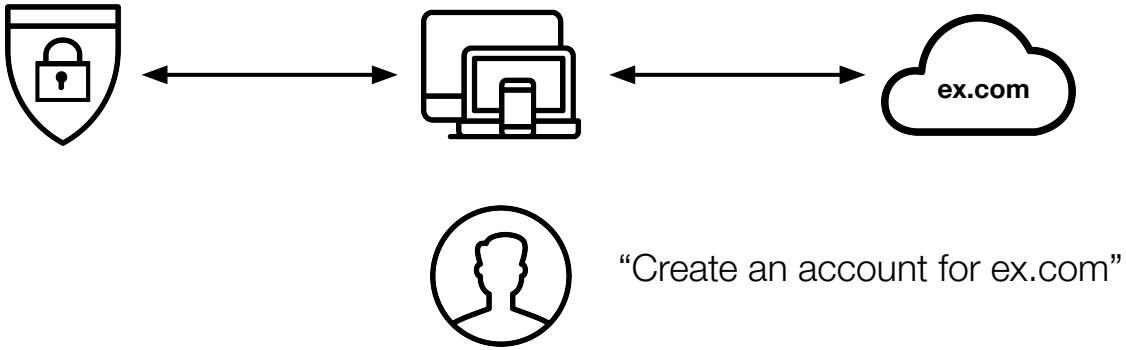
Terminology



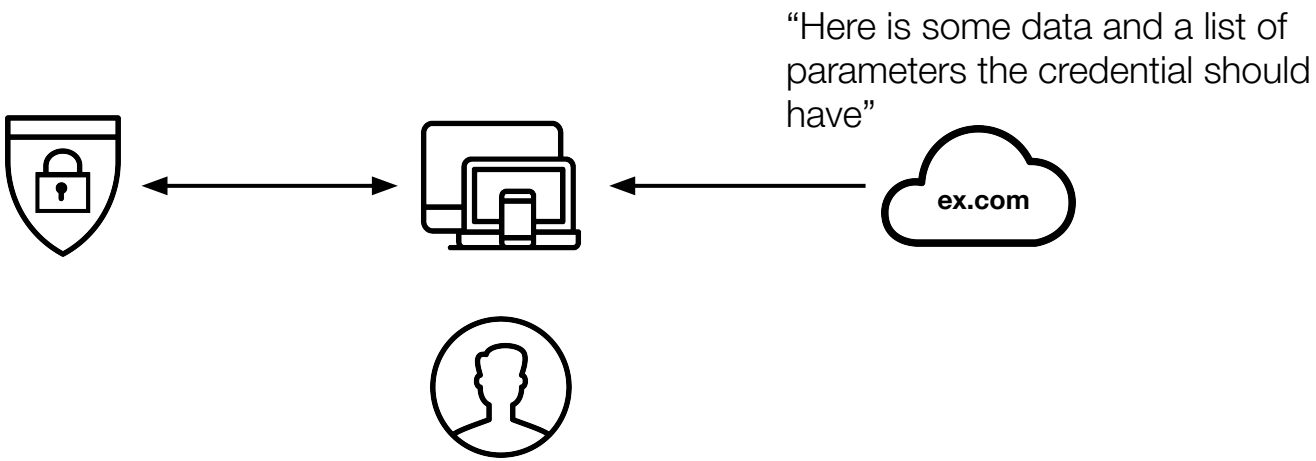
Terminology



Registration

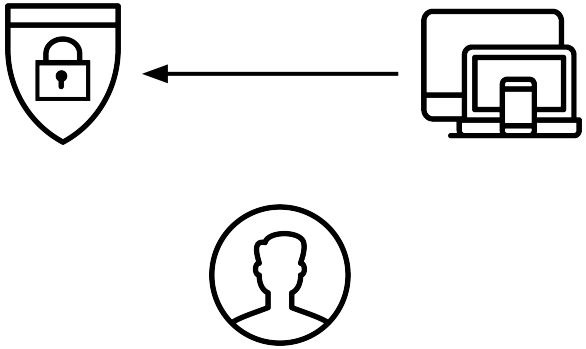


Registration

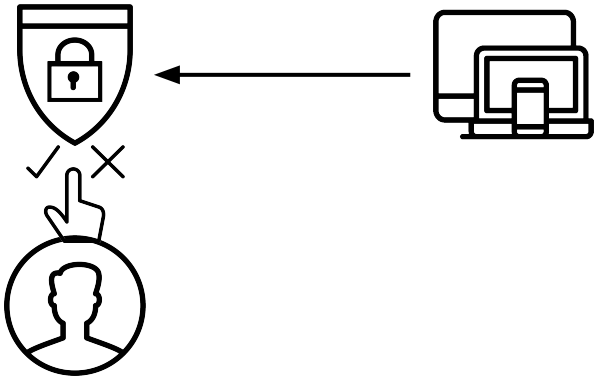


Registration

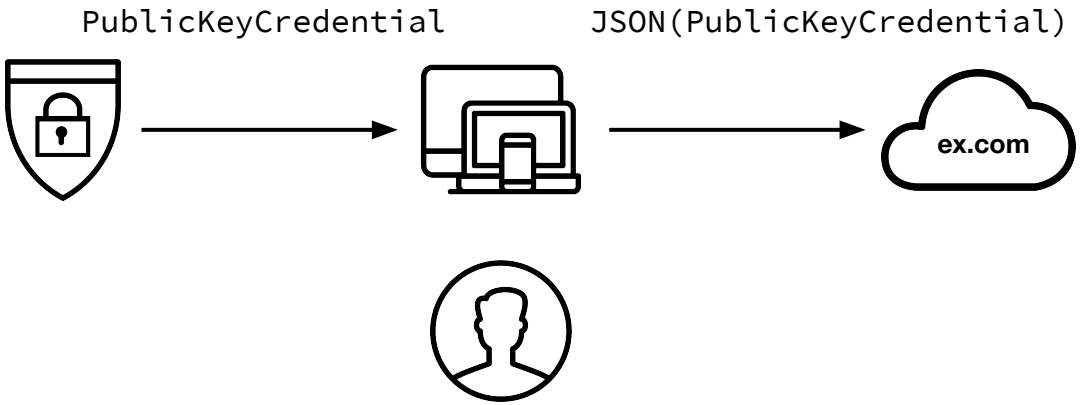
“Given these options, try making a Credential”



Registration



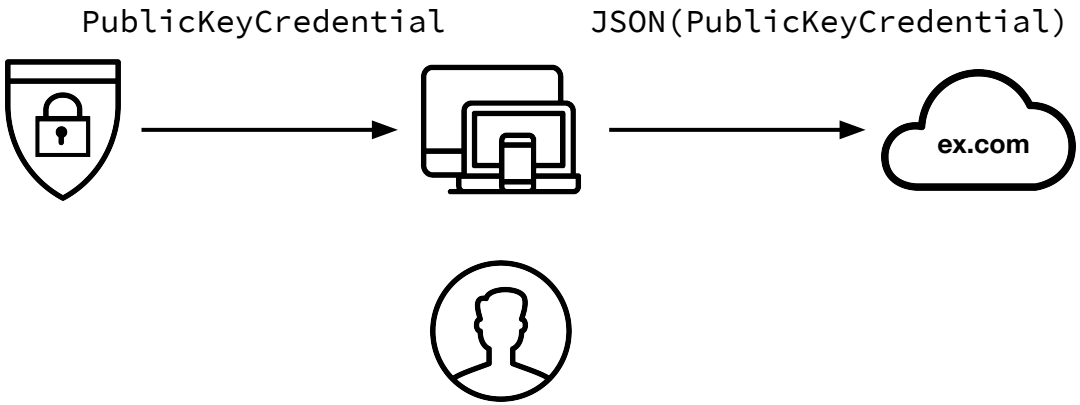
Registration



Hark! This is where Attestation can Occur!

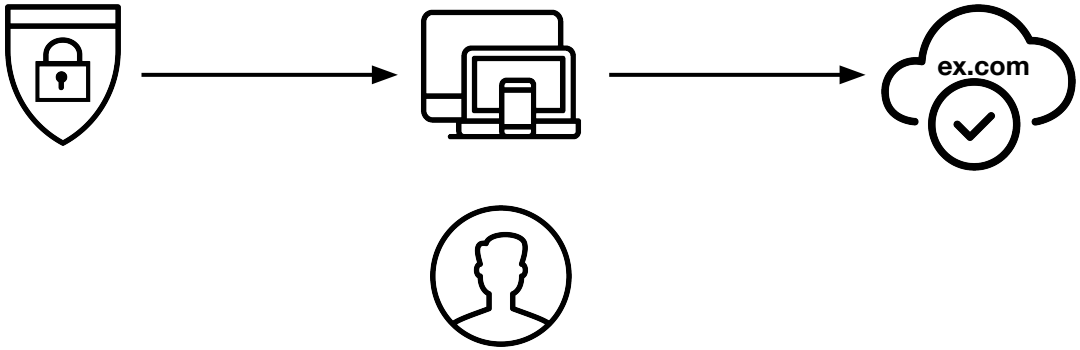
- When an authenticator hands back the new credential it can cryptographically **attest** that it created the credential
- Helpful if you want to control the *type* of authenticators your users log in with
- I don’t recommend dealing with it
  - We’ll talk more about this later though

Registration

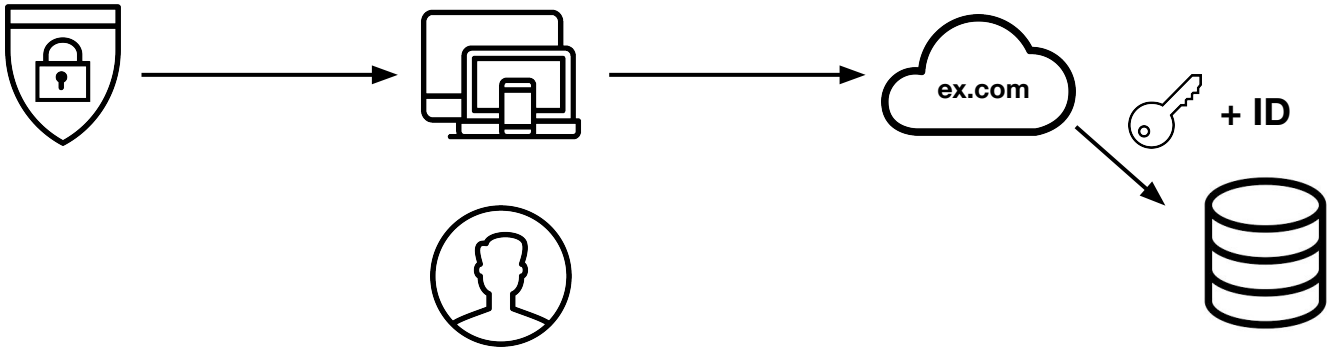


Registration

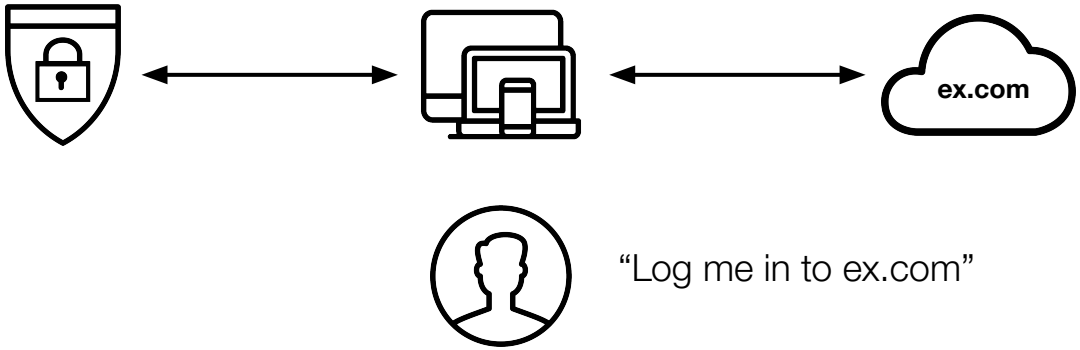
“The data I sent you is signed correctly and you followed my options!”



Registration

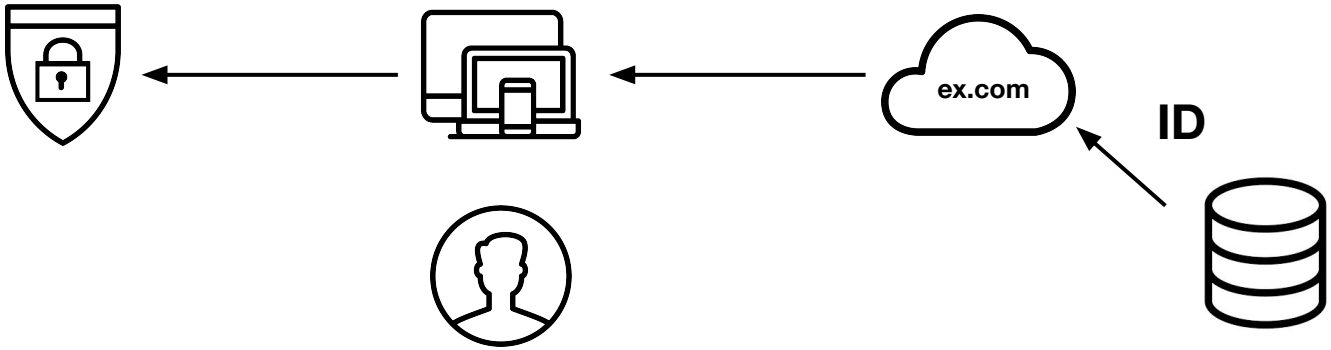


Login (Assertion)

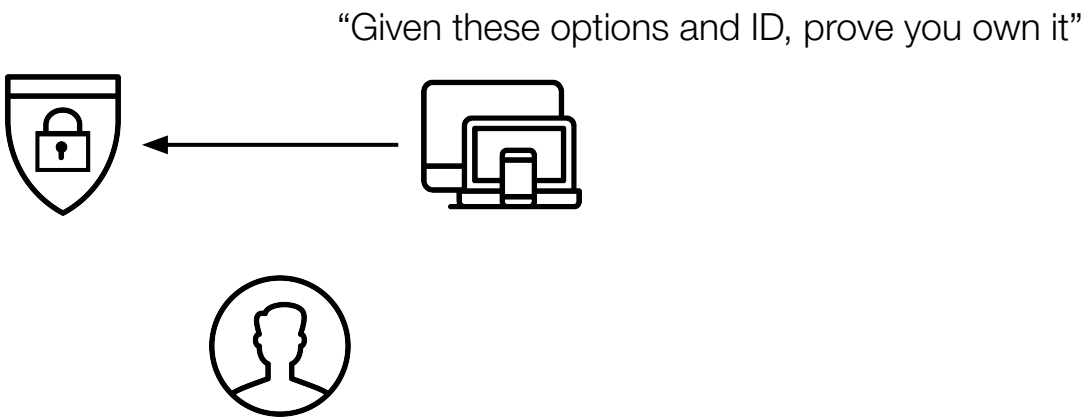


Login (Assertion)

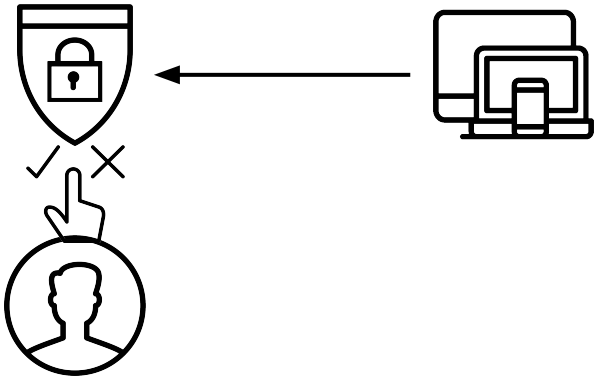
“Assert that you own the private key for this credential with the given ID by signing this data”



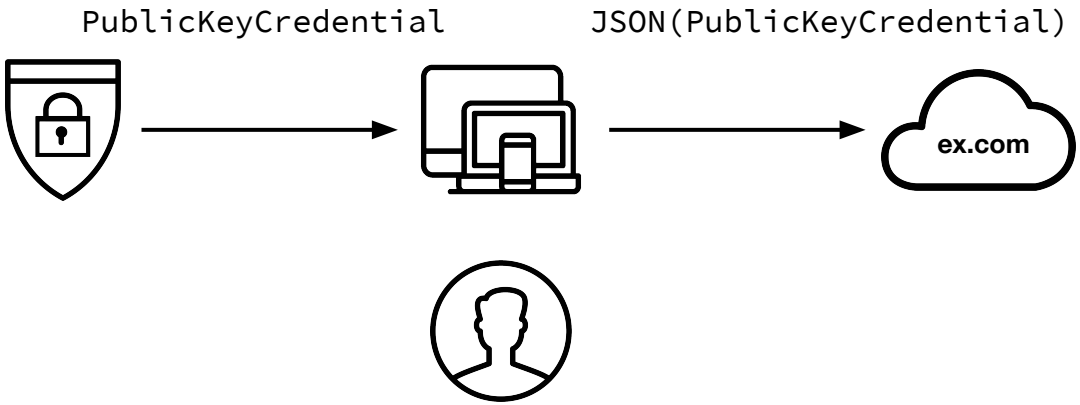
Login (Assertion)



Login (Assertion)

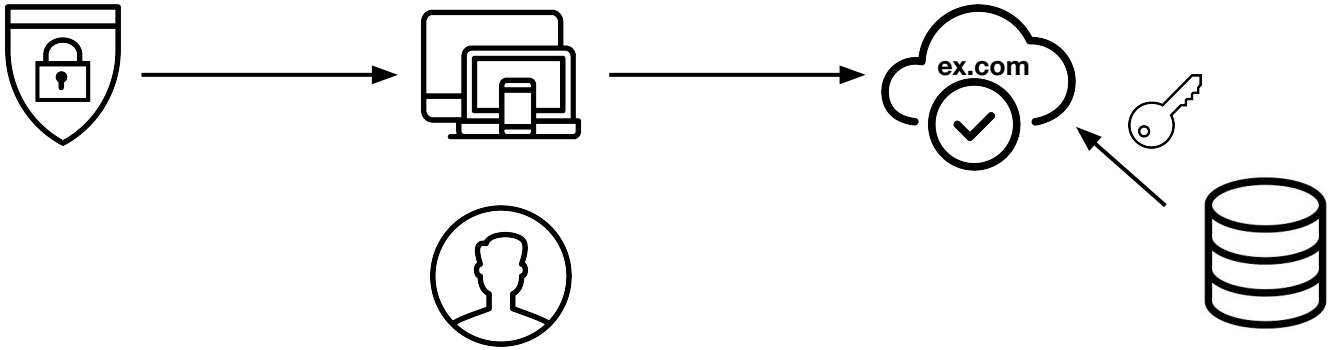


Login (Assertion)



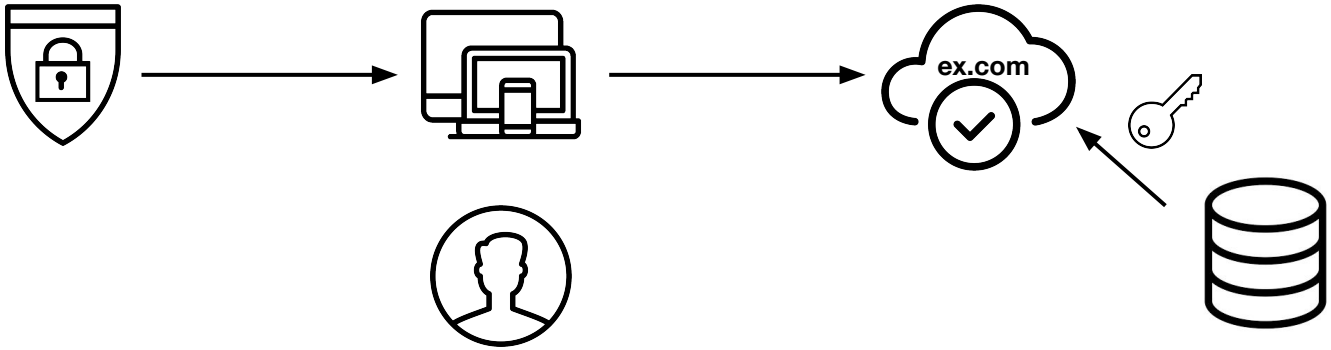
Login (Assertion)

“This data is properly signed and proves you own the private key”



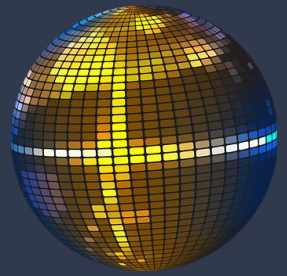
Login (Assertion)

“This data is properly signed and proves you own the private key”

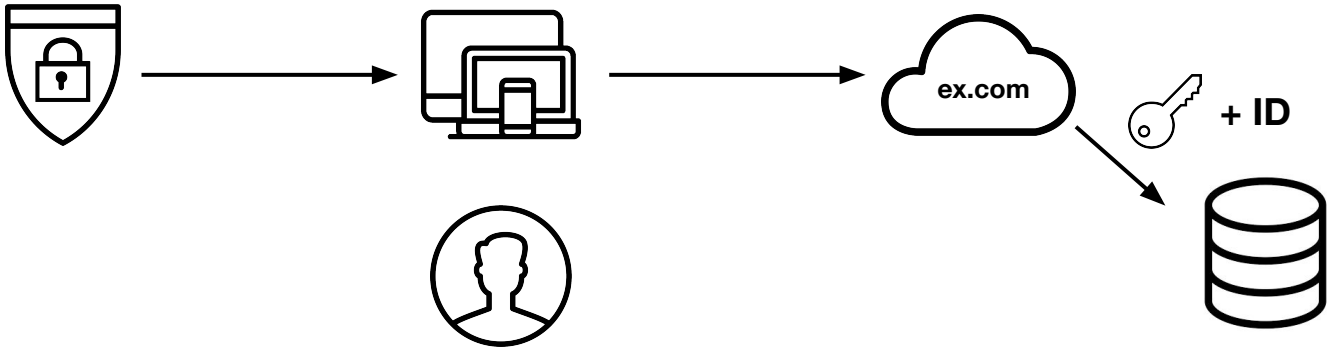


Oh right, one more thing!

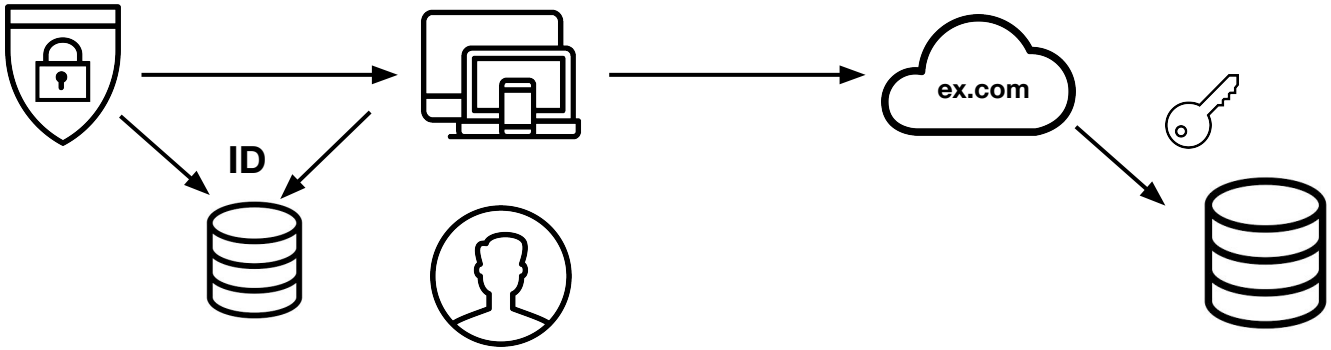
Discoverable Keys



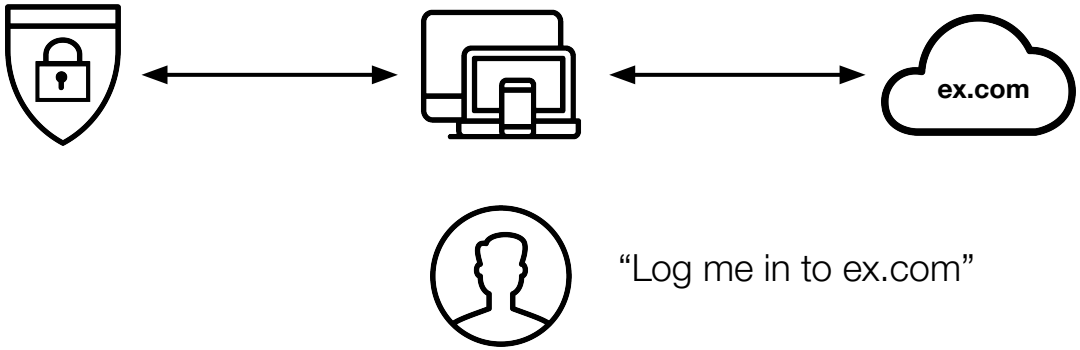
Registration



Discoverable Keys

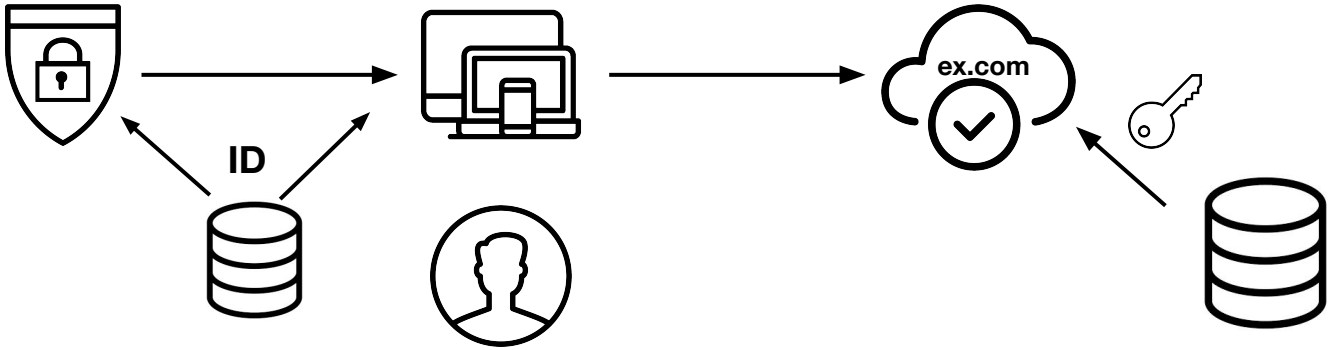


Login (Assertion)



Login (Assertion) with Discoverable Keys

“This data is properly signed and proves you own the private key”



What are Discoverable Keys are meant for?

- I believe username-less login is the most compelling reason to use them
- Could potentially be used to “federate” credentials.
  - You could have credentials specific to an authenticator and a browser profile!

Should You Care About Them?

Should You Care About Them?

- No

Should You Care About Them?

- No
- Won’t affect your implementation very much unless you require them.

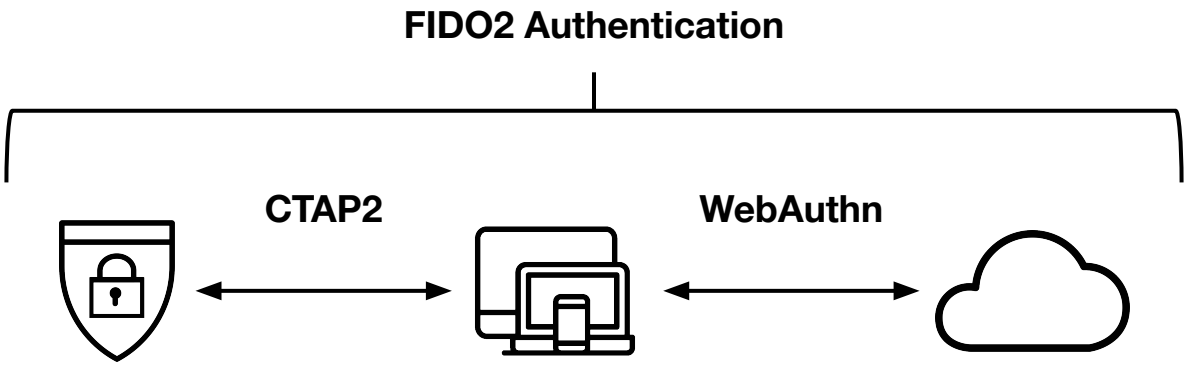
Other things you probably shouldn’t care about

But should know about

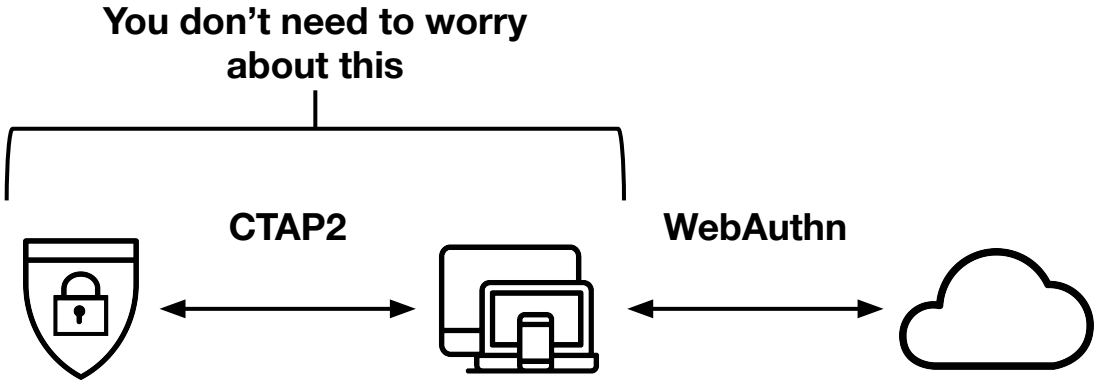
Things to know about

- The Non-Webauthn bits of FIDO2
- Authenticators
- Extensions
- Transitioning from U2F/CTAP1 to FIDO2

The Non-WebAuthn Bits



The Non-WebAuthn Bits



The Non-WebAuthn Bits

- CTAP2 is the Client To Authenticator Protocol
  - Version 2!
- **CTAP1 is now U2F**
- Unless you are a hardware vendor, you need to know nothing about this protocol.
- If you are currently using CTAP1/U2F you can still support those keys!

Authenticators

- Doesn't need to be a physical security key
  - Security keys are often referred to as "roaming" or "cross-platform"
- Can be embedded inside users' devices
  - Windows Hello, Android, and iOS devices are capable of biometric-backed authentication
- Google has been working on CaBLE
  - Allows phones to be roaming authenticators

Extensions

- WebAuthn has extensions!
  - Extensions can be requested during registration and login and provide a wide array of data.
- Extension support is not necessary for a base implementation of FIDO2
- Unless...

Taking your U2F users into WebAuthn Land

- CTAP1/U2F Authenticators are retroactively supported via an extension.
- If you already have users that rely on U2F credentials, you can use this extension to support them.
- Ideally however, you should start fresh and focus on native FIDO2 support.

The biggest thing(s) you should care about...

Practical Deployment

Use great resources

- Use WebAuthn/FIDO2 deployment resources
  - webauthn.io
  - webauthn.guide
  - herrjemand/awesome-webauthn
  - fido-alliance/how-to-fido
  - webauthn.how (Coming Soon!)

Use a library

- Seriously, unless you absolutely need to implement your own logic to handle WebAuthn RP operations, use a library
  - Even if you think you can do better, consider just contributing to an existing library
  - Or, potentially open-source your own!

webauthn.io Featured Libraries

Go
duo-labs/webauthn
Duo Labs
Server

Go
duo-labs/webauthn.io
Duo Labs
Demo

Go
koesie10/webauthn
Koen Vlaswinkel
Server

Java
duo-labs/android-webauthn-authenticator
Duo Labs
Authenticator

Java
google/webauthndemo
Google
Demo

Java
webauthn4j/webauthn4j
Yoshikazu Nojima
Server

Java
Yubico/java-webauthn-server
Yubico
Server

Javascript
fido-alliance/webauthn-demo
Fido Alliance
Demo

.NET
abergs/fido2-net-lib
Anders Åberg
Server/Demo

Python
duo-labs/py_webauthn
Duo Labs
Server/Demo

Ruby
cedarcode/webauthn-ruby
Cedarcode
Server

Chrome Extension
google/virtual-authenticators-tab
Nina Satragno
Authenticator

Consider using a client-side library, too

```
// Encode an ArrayBuffer into a base64 string.
function bufferEncode(value) {
    return base64js.fromByteArray(value)
        .replace(/\+/g, "-")
        .replace(/\//g, "_")
        .replace(/=/g, "");
}
```

@github/webauthn-json

```
function create(requestJSON: JSON): Promise<JSON>;
function get(requestJSON: JSON): Promise<JSON>;
function supported(): boolean;
```

Signature Verification

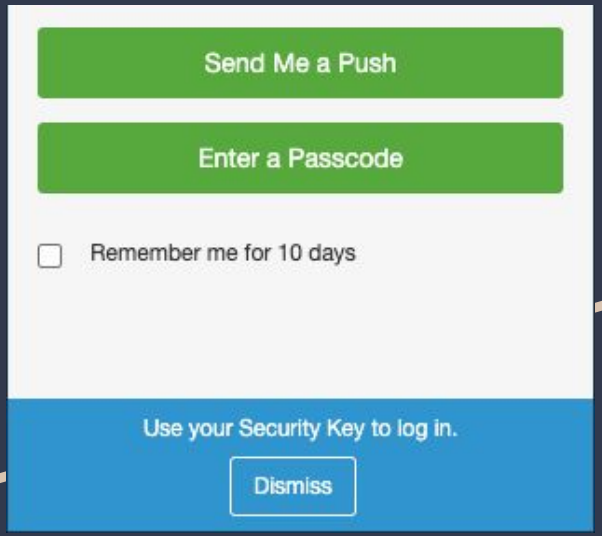
- Make sure to check the return values of signature verification functions
  - Better yet, the don’t rely on return codes at all and raise a specific error or exception type upon failure

Signature Verification

```
from cryptography.exceptions import InvalidSignature


#   * Verify that sig is a valid signature over the
#     concatenation of authenticatorData and clientDataHash
#     using the credential public key with alg.

try:
    _verify_signature(credential_public_key, alg,
                      verification_data, signature)
except InvalidSignature:
    raise RegistrationRejectedException('Invalid signature received.')
```

WebAuthn: The Hard Parts

Usability: Cross-origin IFrame Support



- When we initially developed support for WebAuthn at Duo, the spec still said nothing of cross-origin IFrame limitations
  - But, this changed, and for a time both `.create` and `.get` were restricted to top-level browsing context
- Therefore, in Duo, the UX for U2F became superior to WebAuthn, because one required a popup and the other didn't.
  - This gave WebAuthn a bad reputation internally, because when we switched to it from U2F *the experience worsened*
- Therefore, in Duo, they currently use the U2F API opportunistically, and when you enroll a *security key* they actually dual-enroll it
  - Now, there's Feature Policy which could permit assertions on cross-origin IFrames, but change can be hard once you've deployed a solution that works

CBOR and COSE Registries

ECDH-SS + A192KW	-33	ECDH SS w/ Concat KDF and AES Key Wrap w/ 192-bit key	[RFC8152]	Yes
ECDH-SS + A128KW	-32	ECDH SS w/ Concat KDF and AES Key Wrap w/ 128-bit key	[RFC8152]	Yes
ECDH-ES + A256KW	-31	ECDH ES w/ Concat KDF and AES Key Wrap w/ 256-bit key	[RFC8152]	Yes
ECDH-ES + A192KW	-30	ECDH ES w/ Concat KDF and AES Key Wrap w/ 192-bit key	[RFC8152]	Yes
ECDH-ES + A128KW	-29	ECDH ES w/ Concat KDF and AES Key Wrap w/ 128-bit key	[RFC8152]	Yes
ECDH-SS + HKDF-512	-28	ECDH SS w/ HKDF - generate key directly	[RFC8152]	Yes
ECDH-SS + HKDF-256	-27	ECDH SS w/ HKDF - generate key directly	[RFC8152]	Yes
ECDH-ES + HKDF-512	-26	ECDH ES w/ HKDF - generate key directly	[RFC8152]	Yes
ECDH-ES + HKDF-256	-25	ECDH ES w/ HKDF - generate key directly	[RFC8152]	Yes
Unassigned	-24 to -19			
SHAKE128 (TEMPORARY - registered 2019-08-13, extension registered 2020-06-19, expires 2021-08-13)	-18	128-bit SHAKE	[draft-ietf-cose-hash-algs]	Yes
SHA-512/256 (TEMPORARY - registered 2019-08-13, extension registered 2020-06-19, expires 2021-08-13)	-17	SHA-2 512-bit Hash truncated to 256-bits	[draft-ietf-cose-hash-algs]	Yes
SHA-256 (TEMPORARY - registered 2019-08-13, extension registered 2020-06-19, expires 2021-08-13)	-16	SHA-2 256-bit Hash	[draft-ietf-cose-hash-algs]	Yes
SHA-256/64 (TEMPORARY - registered 2019-08-13, extension registered 2020-06-19, expires 2021-08-13)	-15	SHA-2 256-bit Hash truncated to 64-bits	[draft-ietf-cose-hash-algs]	Filter Only
SHA-1 (TEMPORARY - registered 2019-08-13, extension registered 2020-06-19, expires 2021-08-13)	-14	SHA-1 Hash	[draft-ietf-cose-hash-algs]	Filter Only
direct+HKDF-AES-256	-13	Shared secret w/ AES-MAC 256-bit key	[RFC8152]	Yes
direct+HKDF-AES-128	-12	Shared secret w/ AES-MAC 128-bit key	[RFC8152]	Yes
direct+HKDF-SHA-512	-11	Shared secret w/ HKDF and SHA-512	[RFC8152]	Yes
direct+HKDF-SHA-256	-10	Shared secret w/ HKDF and SHA-256	[RFC8152]	Yes
Unassigned	-9			
EdDSA	-8	EdDSA	[RFC8152]	Yes
ES256	-7	ECDSA w/ SHA-256	[RFC8152]	Yes
direct	-6	Direct use of CEK	[RFC8152]	Yes
A256KW	-5	AES Key Wrap w/ 256-bit key	[RFC8152]	Yes
A192KW	-4	AES Key Wrap w/ 192-bit key	[RFC8152]	Yes
A128KW	-3	AES Key Wrap w/ 128-bit key	[RFC8152]	Yes
Unassigned	-2 to -1			
Reserved	0		[RFC8152]	No
A128GCM	1	AES-GCM mode w/ 128-bit key, 128-bit tag	[RFC8152]	Yes
A192GCM	2	AES-GCM mode w/ 192-bit key, 128-bit tag	[RFC8152]	Yes
A256GCM	3	AES-GCM mode w/ 256-bit key, 128-bit tag	[RFC8152]	Yes
HMAC 256/64	4	HMAC w/ SHA-256 truncated to 64 bits	[RFC8152]	Yes
HMAC 256/256	5	HMAC w/ SHA-256	[RFC8152]	Yes
HMAC 384/384	6	HMAC w/ SHA-384	[RFC8152]	Yes
HMAC 512/512	7	HMAC w/ SHA-512	[RFC8152]	Yes
Unassigned	8-9			
AES-CCM-16-64-128	10	AES-CCM mode 128-bit key, 64-bit tag, 13-byte nonce	[RFC8152]	Yes
AES-CCM-16-64-256	11	AES-CCM mode 256-bit key, 64-bit tag, 13-byte nonce	[RFC8152]	Yes
AES-CCM-64-64-128	12	AES-CCM mode 128-bit key, 64-bit tag, 7-byte nonce	[RFC8152]	Yes
AES-CCM-64-64-256	13	AES-CCM mode 256-bit key, 64-bit tag, 7-byte nonce	[RFC8152]	Yes
AES-MAC 128/64	14	AES-MAC 128-bit key, 64-bit tag	[RFC8152]	Yes
AES-MAC 256/64	15	AES-MAC 256-bit key, 64-bit tag	[RFC8152]	Yes
Unassigned	16-23			
ChaCha20/Poly1305	24	ChaCha20/Poly1305 w/ 256-bit key, 128-bit tag	[RFC8152]	Yes
AES-MAC 128/128	25	AES-MAC 128-bit key, 128-bit tag	[RFC8152]	Yes
AES-MAC 256/128	26	AES-MAC 256-bit key, 128-bit tag	[RFC8152]	Yes
Unassigned	27-29			
AES-CCM-16-128-128	30	AES-CCM mode 128-bit key, 128-bit tag, 13-byte nonce	[RFC8152]	Yes
AES-CCM-16-128-256	31	AES-CCM mode 256-bit key, 128-bit tag, 13-byte nonce	[RFC8152]	Yes
AES-CCM-64-128-128	32	AES-CCM mode 128-bit key, 128-bit tag, 7-byte nonce	[RFC8152]	Yes

- WebAuthn uses CBOR for many structures, including attestation statements and authenticator extensions
  - CBOR is a binary format with a data format based on JSON
- WebAuthn also makes use of COSE, which is functionally similar to the JOSE family of standards, but uses a CBOR-based data format instead of JSON
- Sometimes, dealing with CBOR and/or COSE can be weird, which is a pretty good reason to use a higher-level WebAuthn server library

futureimperfect commented on Jul 25, 2018 • edited by equalsJeffH

Member

Pardon the confusion, but I'm having trouble figuring out how, as a Relying Party, to reliably determine the length of the `attestedCredentialData` in authenticator data when authenticator extensions are present.

According to the spec, `attestedCredentialData`'s "length depends on the length of the credential ID and credential public key being attested." It also states the following regarding the length of both the `attestedCredentialData` and `extensions`:

"Note that the authenticator data describes its own length: If the AT and ED flags are not set, it is always 37 bytes long. The attested credential data (which is only present if the AT flag is set) describes its own length. If the ED flag is set, then the total length is 37 bytes plus the length of the attested credential data, plus the length of the CBOR map that follows."

The `credentialPublicKey` within `attestedCredentialData` is also of variable length. In the past, before handling extensions, I just took the rest of the `authenticatorData` after `credentialId`, and assumed that was the `credentialPublicKey`. I've noticed a handful of other implementations making this same assumption. However, if the ED flag is set, how does the Relying Party know the length of the `credentialPublicKey`? Is the solution just to figure out where the `credentialPublicKey` CBOR data structure ends? If so, maybe this should be made clear in the spec?

Thanks!

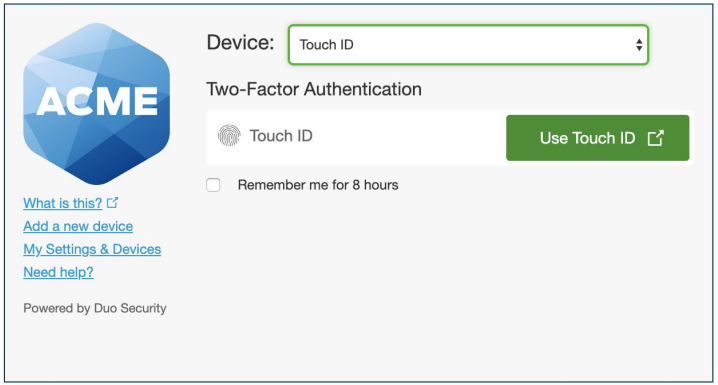
User Onboarding and Education

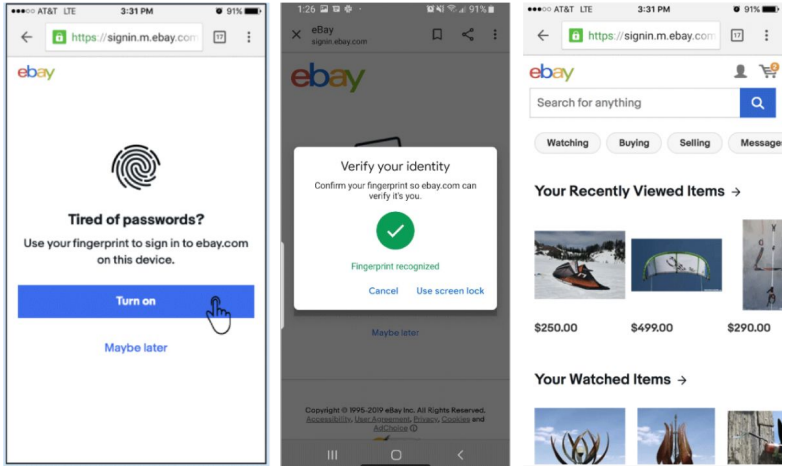
- Sometimes, the hardest part about deploying a new authentication workflow has nothing to do with the technology or security properties it provides

- Ensure end-users feel comfortable when enrolling authenticators, especially when biometrics are at play
  - Conveying to them that their biometric data stays safely on their device and isn't shared with the website is **tough to get right**, but **pays off in the long run**
  - That way, they won't have to worry about their fingers being chopped off in the middle of the night and used to steal their identity

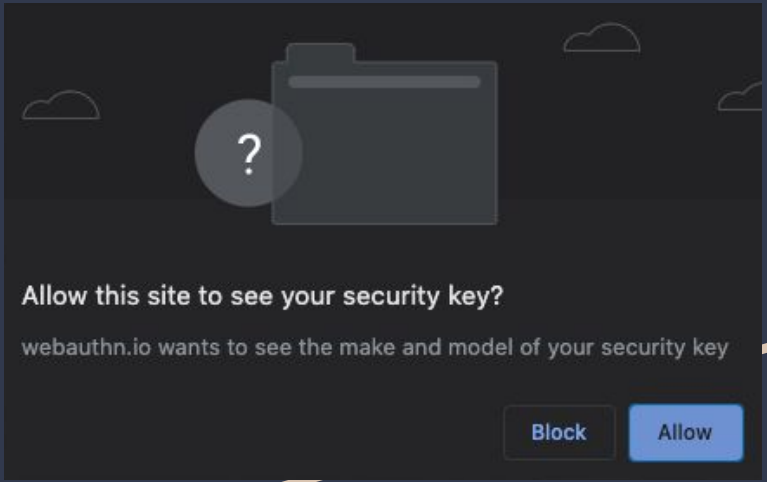
- Conduct user interviews, put new authentication workflows in front of new and existing customers

What do you even call it?!





Attestation



- Unless you're a large **financial institution**, **government** agency, or other extremely **high-value target**, you probably shouldn't be using attestation
  - In addition to the technical hurdles of implementing attestation, there are also privacy concerns
    - Additional dialogs for users to jump through
- What do you lose by not attesting to the provenance of authenticators?
  - Users could be using untrustworthy or software-based authenticators
  - Malicious authenticators become a more realistic threat
  - ...but the **benefits of using WebAuthn/FIDO2** as opposed to other forms of authentication still **outweigh these drawbacks** in many cases

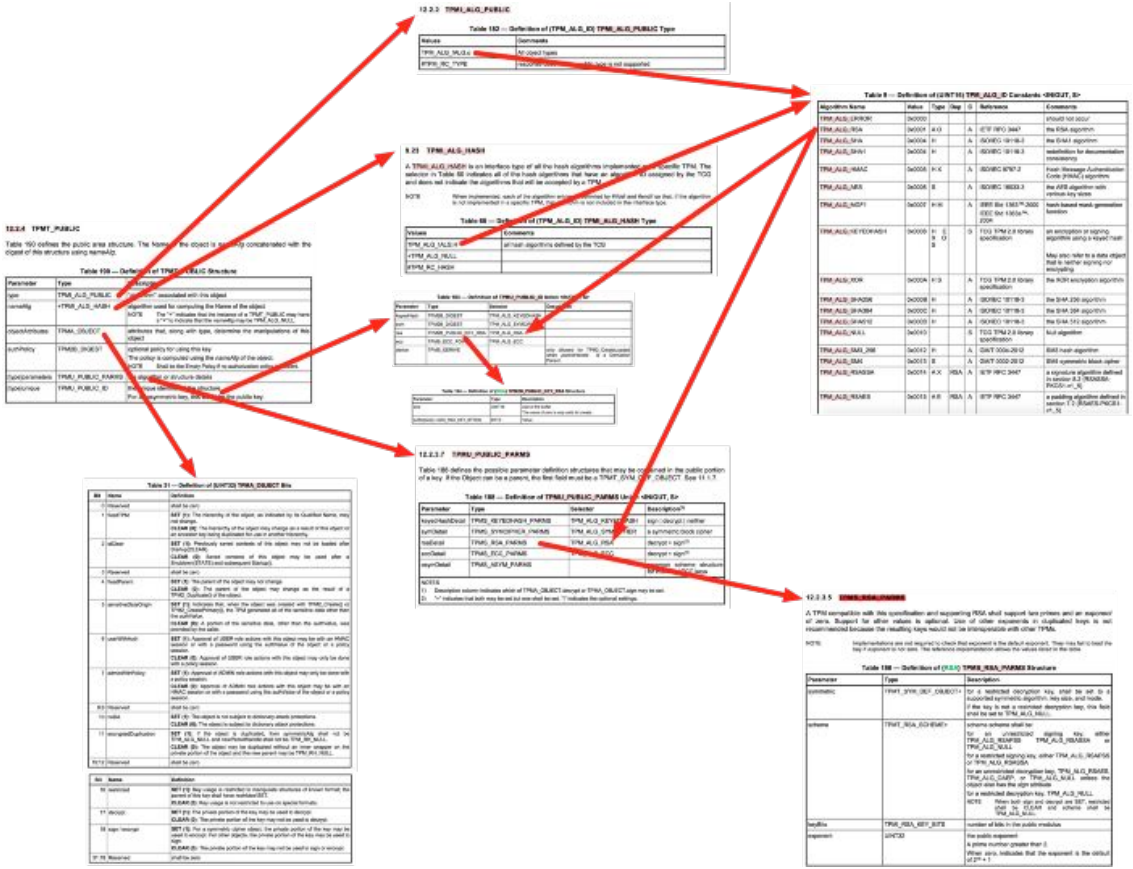
Attestation

```
from OpenSSL.crypto import load_certificate, load_privatekey
from OpenSSL.crypto import X509Store, X509StoreContext
from six import u, b, binary_type, PY3

root_cert_pem = b("""<snip>""")
intermediate_cert_pem = b("""<snip>""")
leaf_cert_pem = b("""<snip>""")

root_cert = load_certificate(FILETYPE_PEM, root_cert_pem)
intermediate_cert = load_certificate(FILETYPE_PEM,
intermediate_cert_pem)
leaf_cert = load_certificate(FILETYPE_PEM, leaf_cert_pem)
store = X509Store()
store.add_cert(root_cert)
store.add_cert(intermediate_cert)
store_ctx = X509StoreContext(store, leaf_cert)

# Will succeed if the intermediate signed the leaf, even if
# the root didn't sign the intermediate.
print(store_ctx.verify_certificate())
```

- Certificate chain validation is hard to get right!
  - You can mess this up with WebAuthn attestation pretty easily
- The FIDO Metadata Service is a great idea, but can be difficult to get started with
  - Instead of maintaining your own list of trusted attestation root certificates, you can look these up by `AAGUID` in the MDS

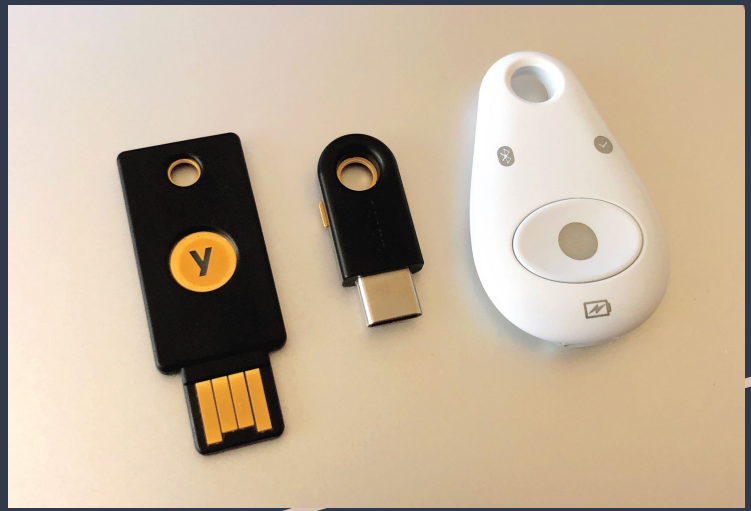
duo.com/labs/research/chain-of-fools

Verification of attestation objects requires that the Relying Party has a trusted method of determining acceptable trust anchors in step 15 above. Also, if certificates are being used, the Relying Party **MUST have access to certificate status information** for the intermediate CA certificates. The Relying Party **MUST also be able to build the attestation certificate chain if the client did not provide this chain in the attestation information**.



Parsing pubArea, the algorithm.
Author: Adam Powers

Account Recovery



- Dealing with account recovery can be one of the hardest parts of deploying WebAuthn

- It's recommended that you read through the FIDO Recommended Account Recovery Practices document, however the guidance essentially boils down to
  - Require or encourage users to enroll multiple authenticators, so that if one is lost or stolen they don't lose access

  - Or, resort to identity proofing
    - The FIDO Login.gov Webinar is one of best practical resources we've encountered so far
    - They discuss both manual and semi-automated approaches for this

Account Recovery, but with more cryptography

"...a new primitive, called **Asynchronous Remote Key Generation** (ARKG), which allows some primary authenticator to generate unlinkable public keys for which the backup authenticator may later recover corresponding private keys."

eprint.iacr.org/2020/1004.pdf

Thank you

@codekaiju && @futureimperfect