Web Authentication

From Spec to Product

Suby Raman

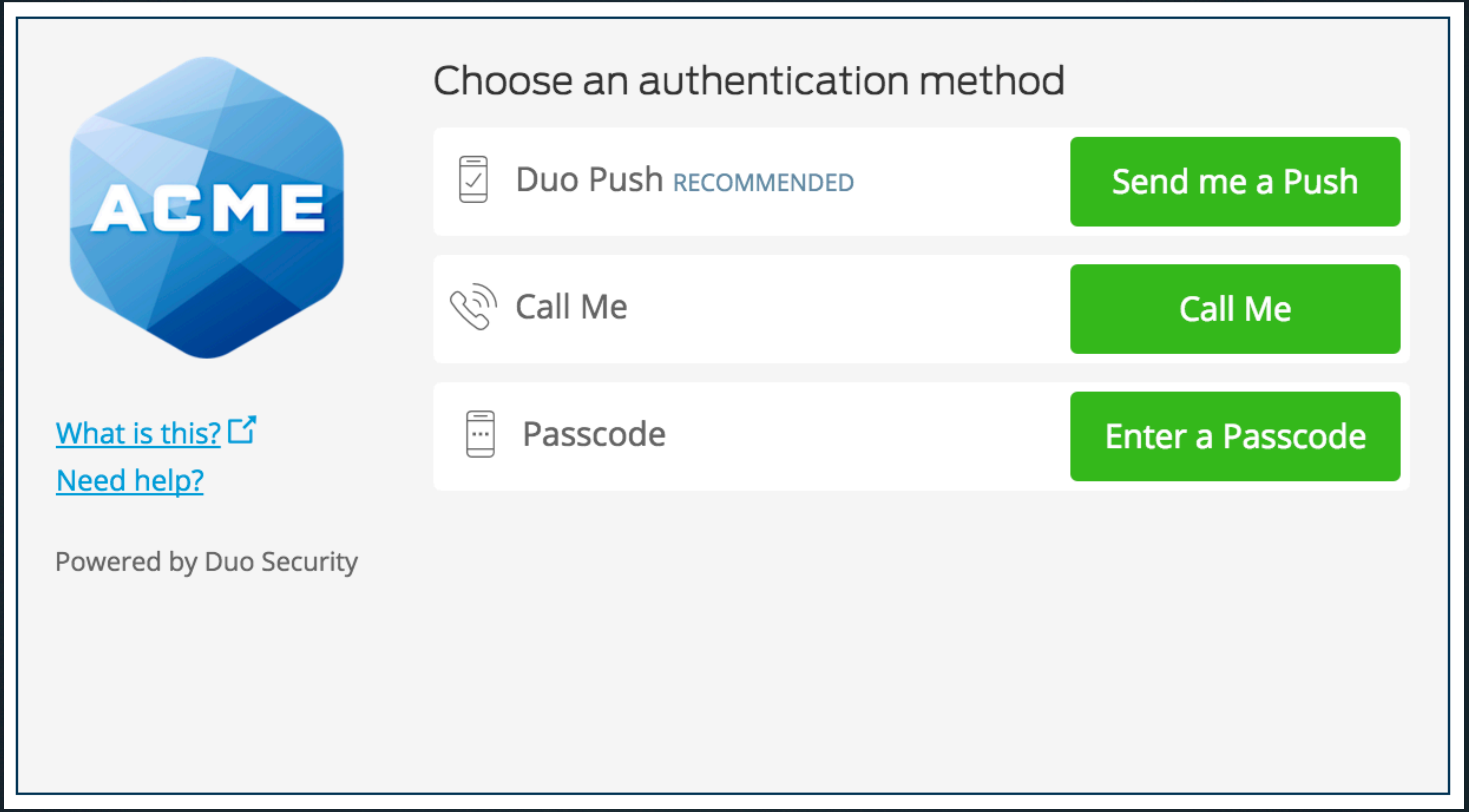
@subyraman

Why should your organization care about WebAuthn?

Coming up with a development plan for WebAuthn

The design challenges of WebAuthn

Looking Ahead



Choose an authentication method

Why are we excited about WebAuthn?

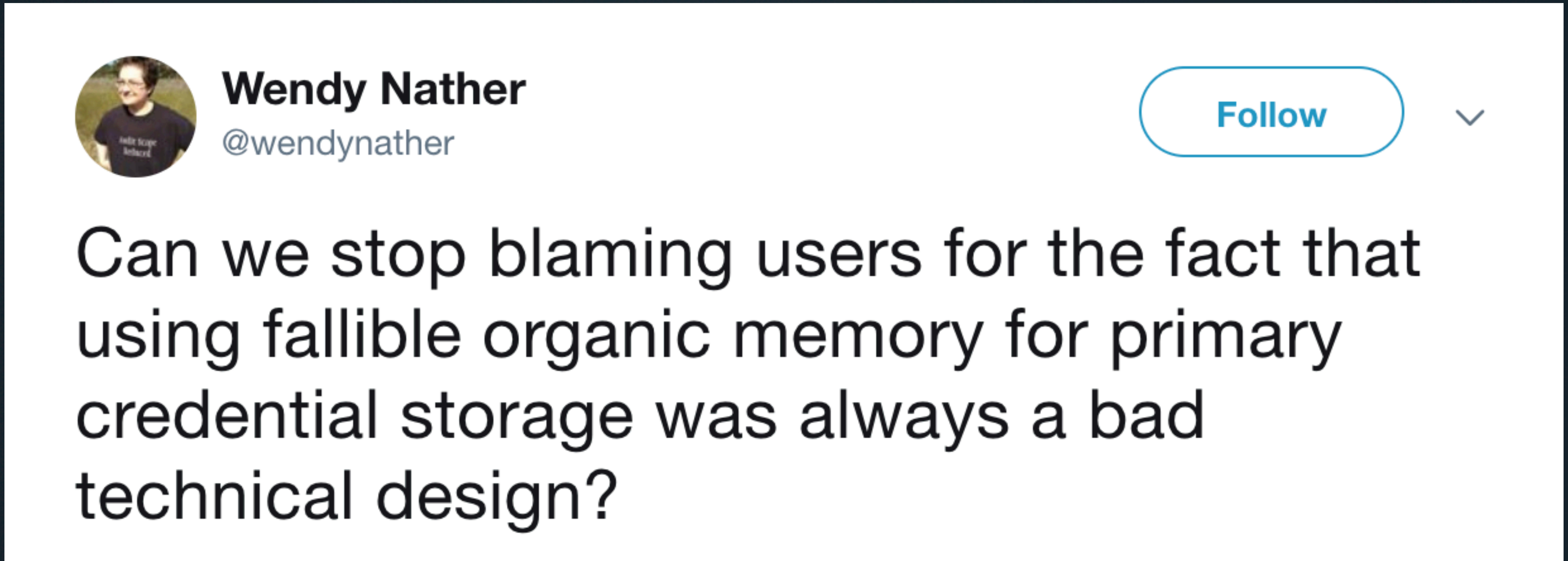
Why can’t users just remember random passwords?

Why are our dumb users re-using passwords?

Why are our dumb users losing their passwords?

Why can’t the dumb developers just be smarter about handling passwords?

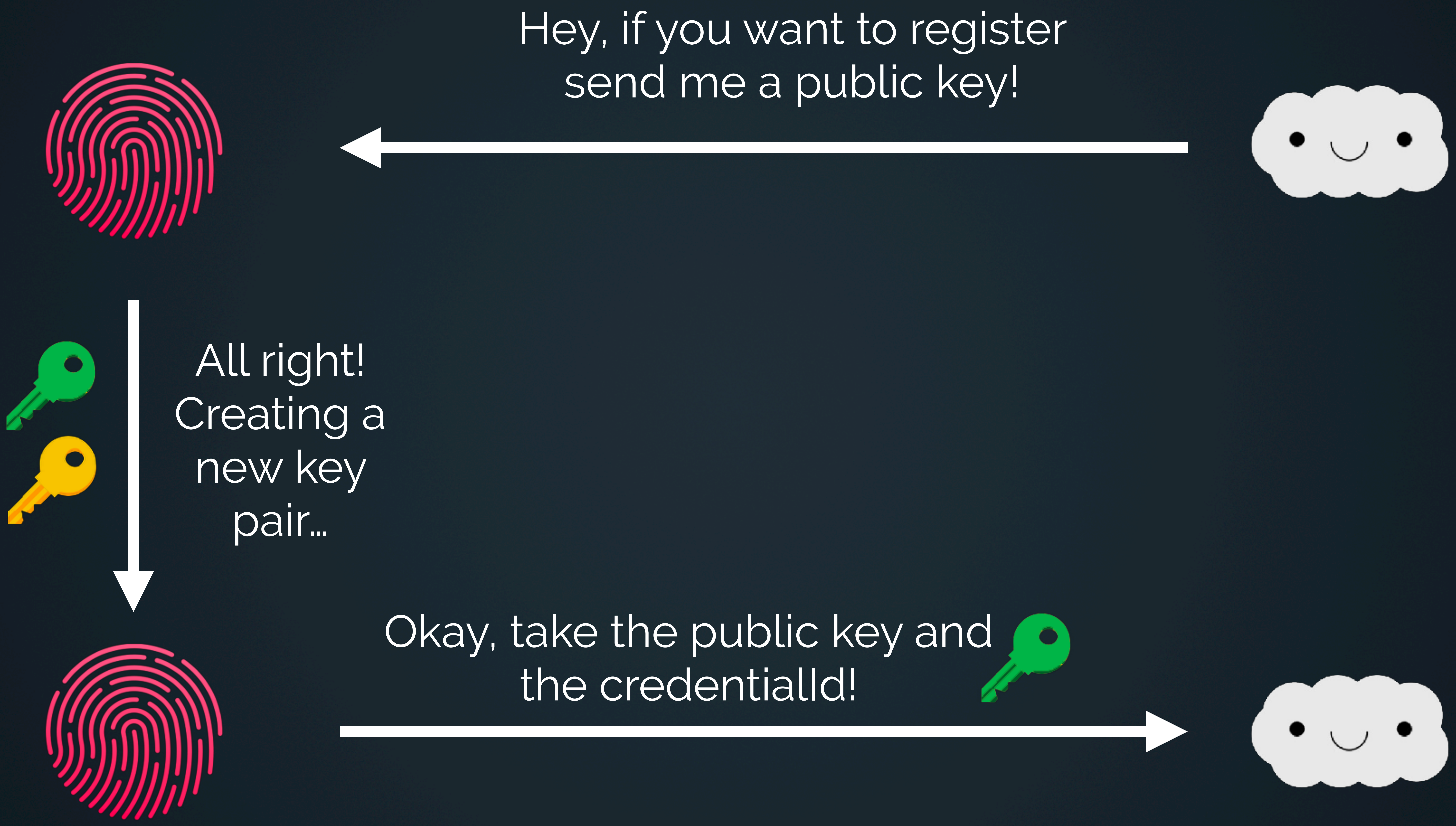
Empathy.



Wendy Nather
@wendynather

Can we stop blaming users for the fact that using fallible organic memory for primary credential storage was always a bad technical design?

Web Authentication allows us to authenticate our users using **public key cryptography.**



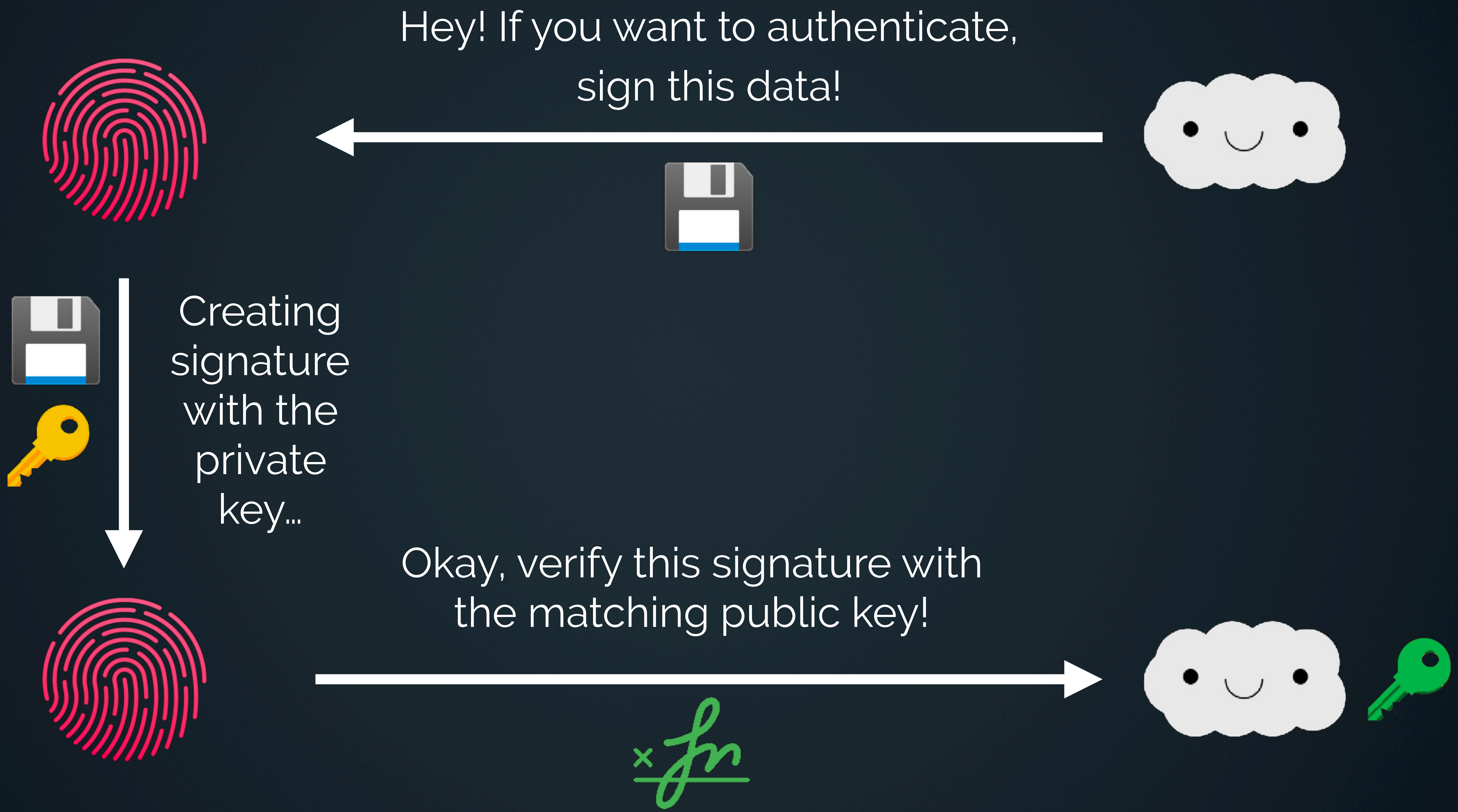
Hey, if you want to register send me a public key!

All right! Creating a new key pair...

Okay, take the public key and the credentialId!

The user creates a key pair and gives us the public key.

```
await navigator.credentials.create({
    publicKey: {…}
});
```



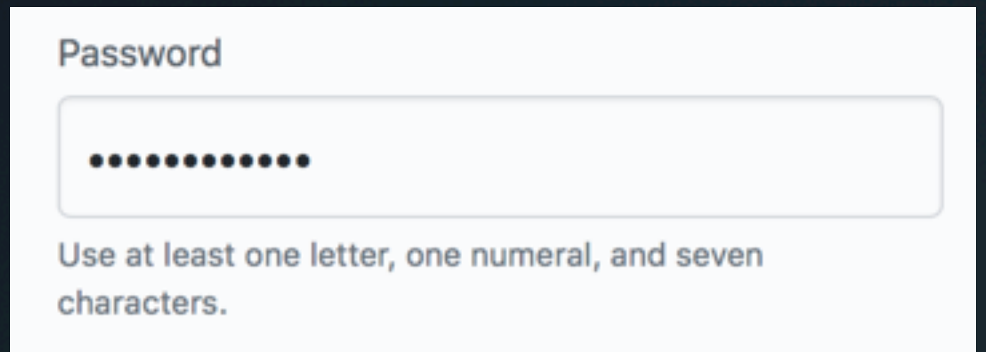
Hey! If you want to authenticate, sign this data!

Creating signature with the private key…

Okay, verify this signature with the matching public key!

The website requests an “assertion” from the user’s authenticator:

```
await navigator.credentials.get({
    publicKey: {…}
});
```



Password

•••••••••••••

Use at least one letter, one numeral, and seven characters.

Passwords are a "shared secret."

Passwords are hard to create and remember.

Passwords are easily stolen.

Passwords encourage unsafe re-use.

Passwords are hard to secure.

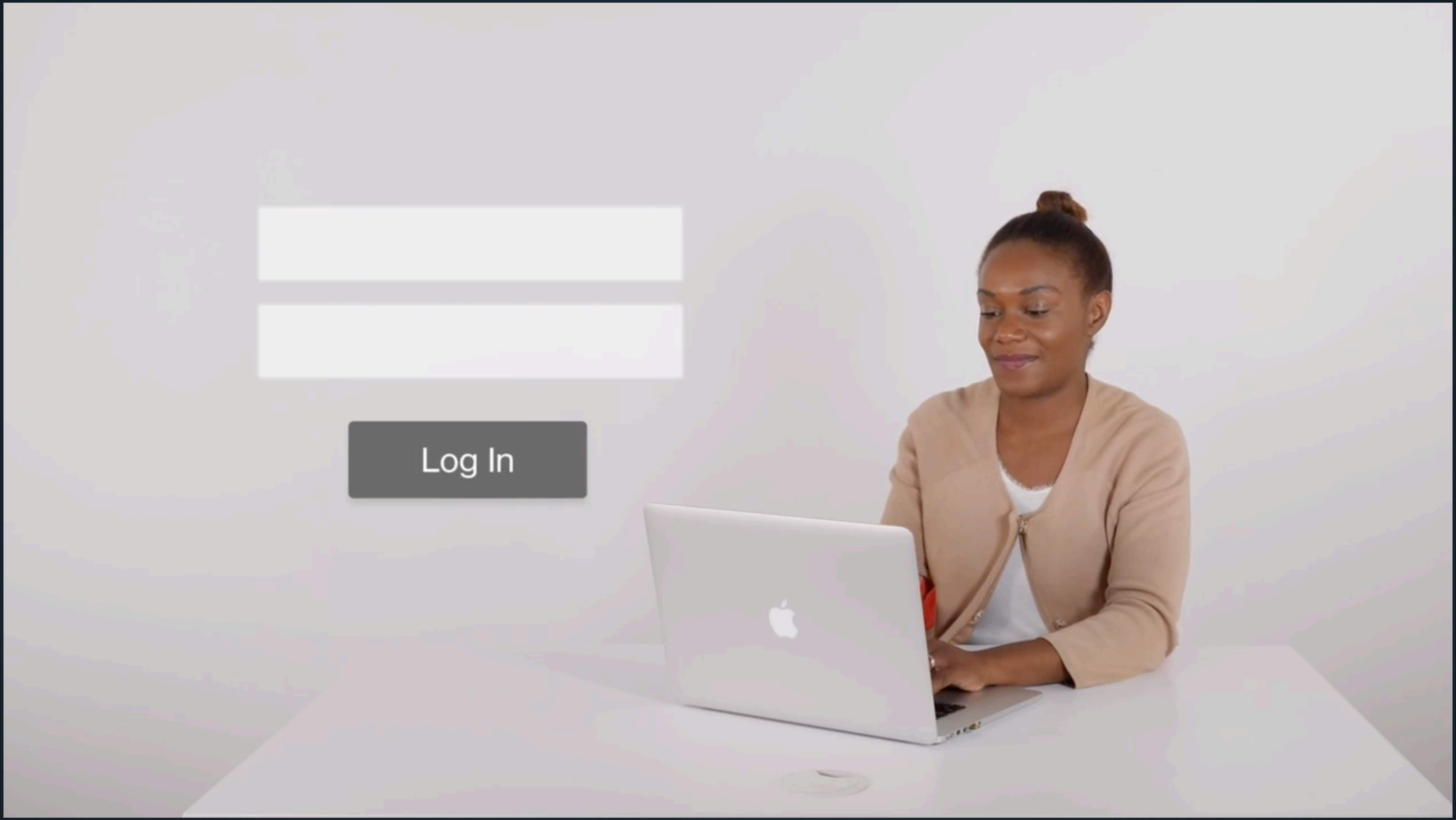
The credential public key is not secret.

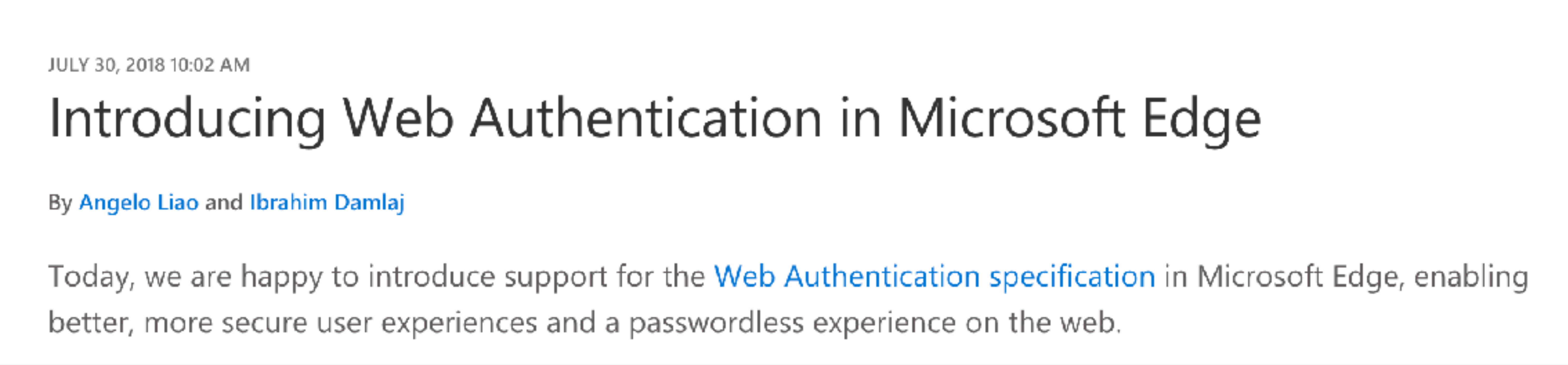
The authenticator creates a random and secure credential.

Secure hardware on devices makes credential theft difficult.

Credentials are scoped to an origin, making re-use impossible.

The credential public key is not secret.


Log In

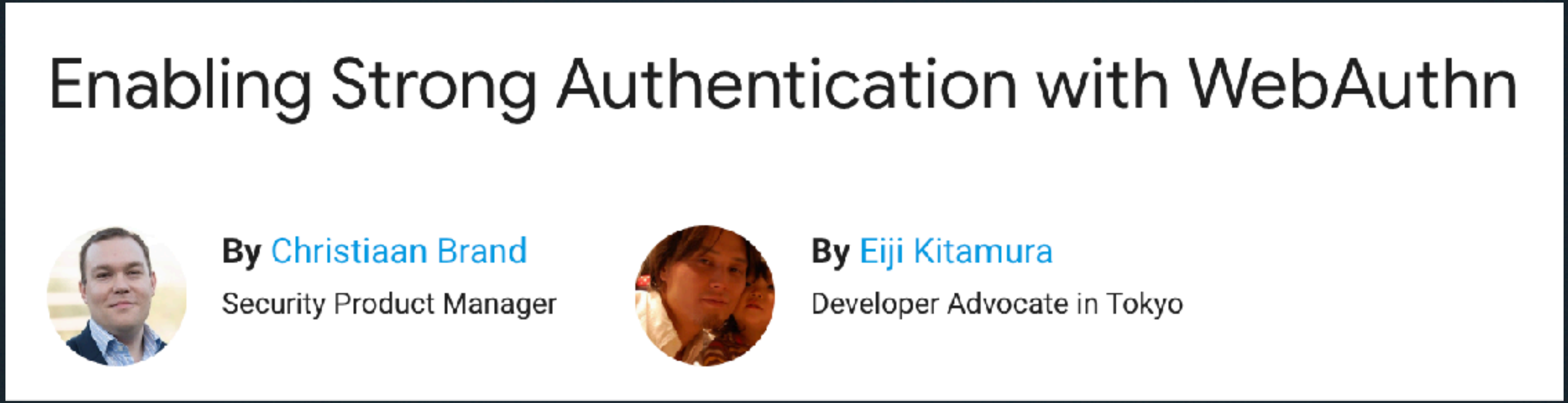


JULY 30, 2018 10:02 AM

Introducing Web Authentication in Microsoft Edge

By Angelo Liao and Ibrahim Damlaj

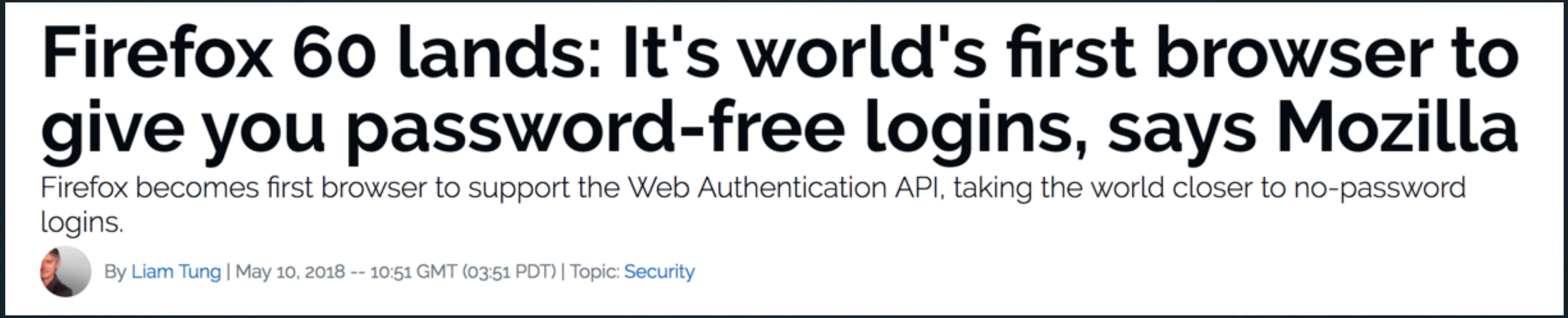
Today, we are happy to introduce support for the Web Authentication specification in Microsoft Edge, enabling better, more secure user experiences and a passwordless experience on the web.



Enabling Strong Authentication with WebAuthn

By Christiaan Brand
Security Product Manager

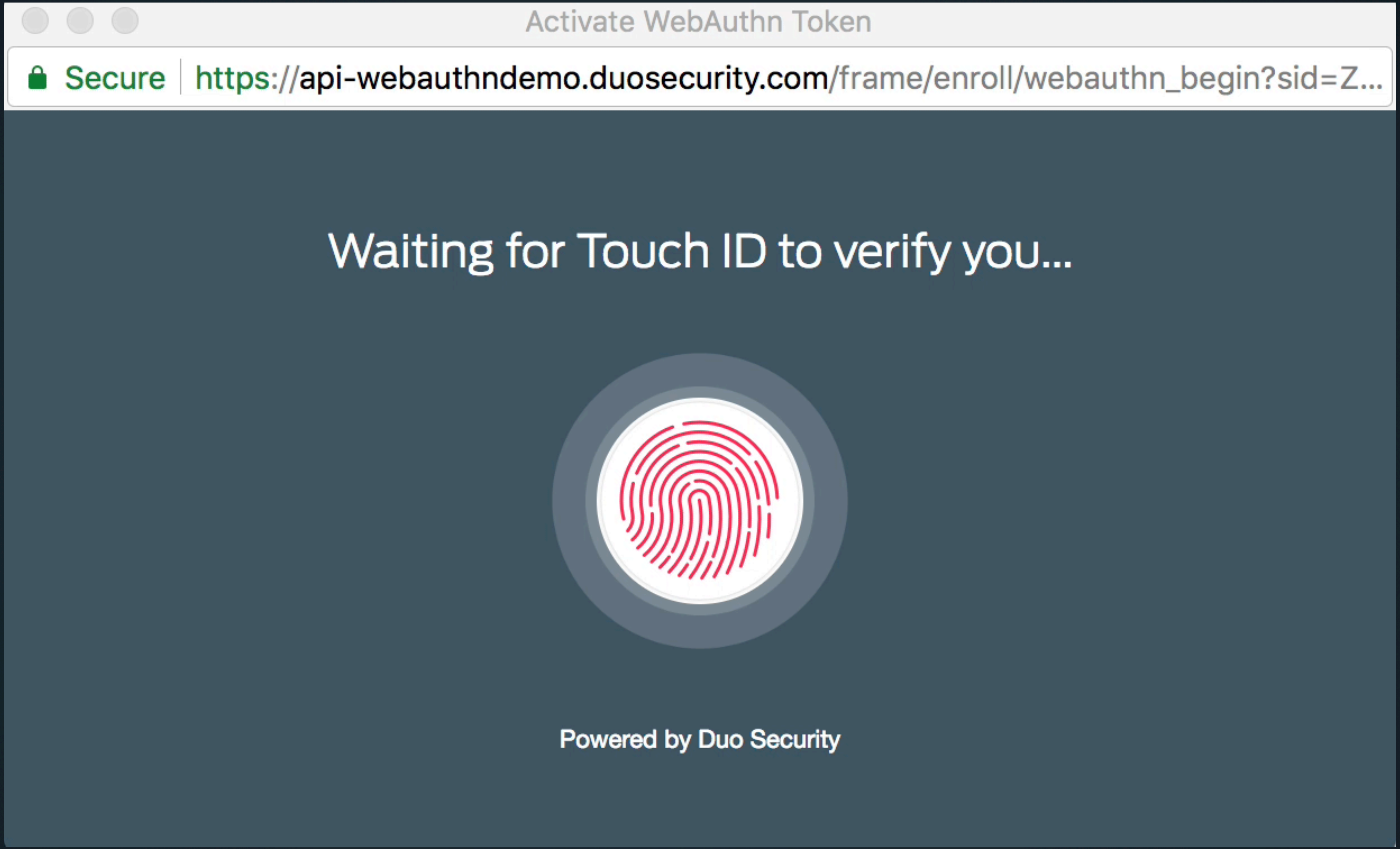
By Eiji Kitamura
Developer Advocate in Tokyo



Firefox 60 lands: It's world's first browser to give you password-free logins, says Mozilla

Firefox becomes first browser to support the Web Authentication API, taking the world closer to no-password logins.

By Liam Tung | May 10, 2018 -- 10:51 GMT (03:51 PDT) | Topic: Security

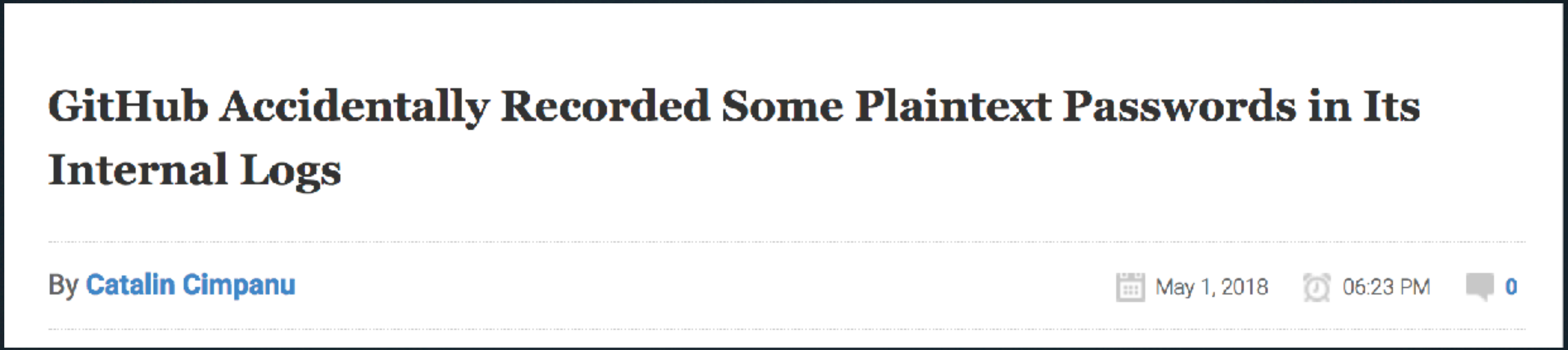


Activate WebAuthn Token

Secure https://api-webauthndemo.duosecurity.com/frame/enroll/webauthn_begin?sid=Z...

Waiting for Touch ID to verify you...

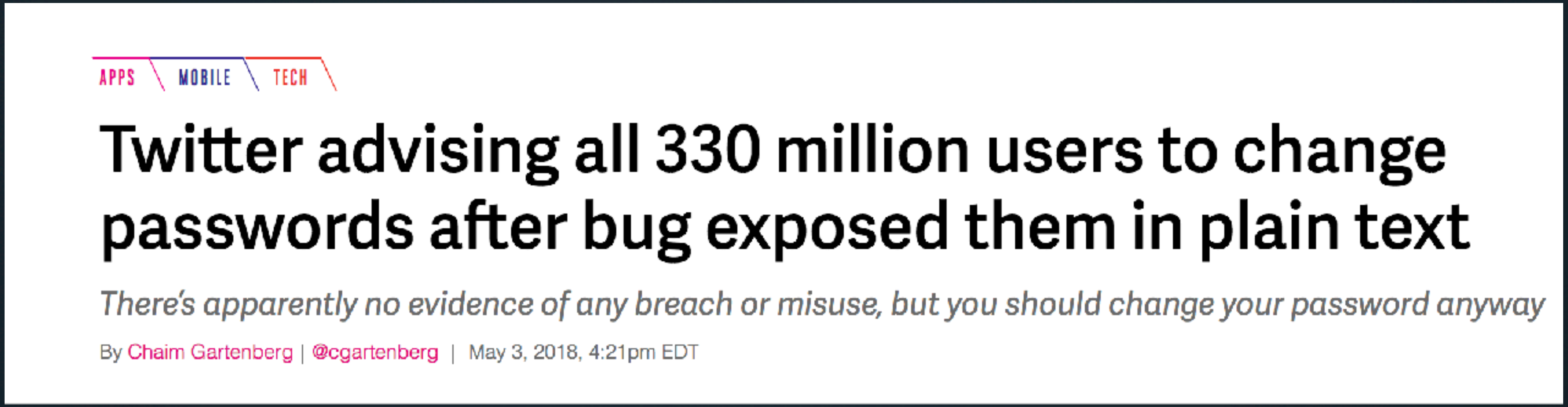
Powered by Duo Security



GitHub Accidentally Recorded Some Plaintext Passwords in Its Internal Logs

By **Catalin Cimpanu**

May 1, 2018 06:23 PM 0



APPS MOBILE TECH

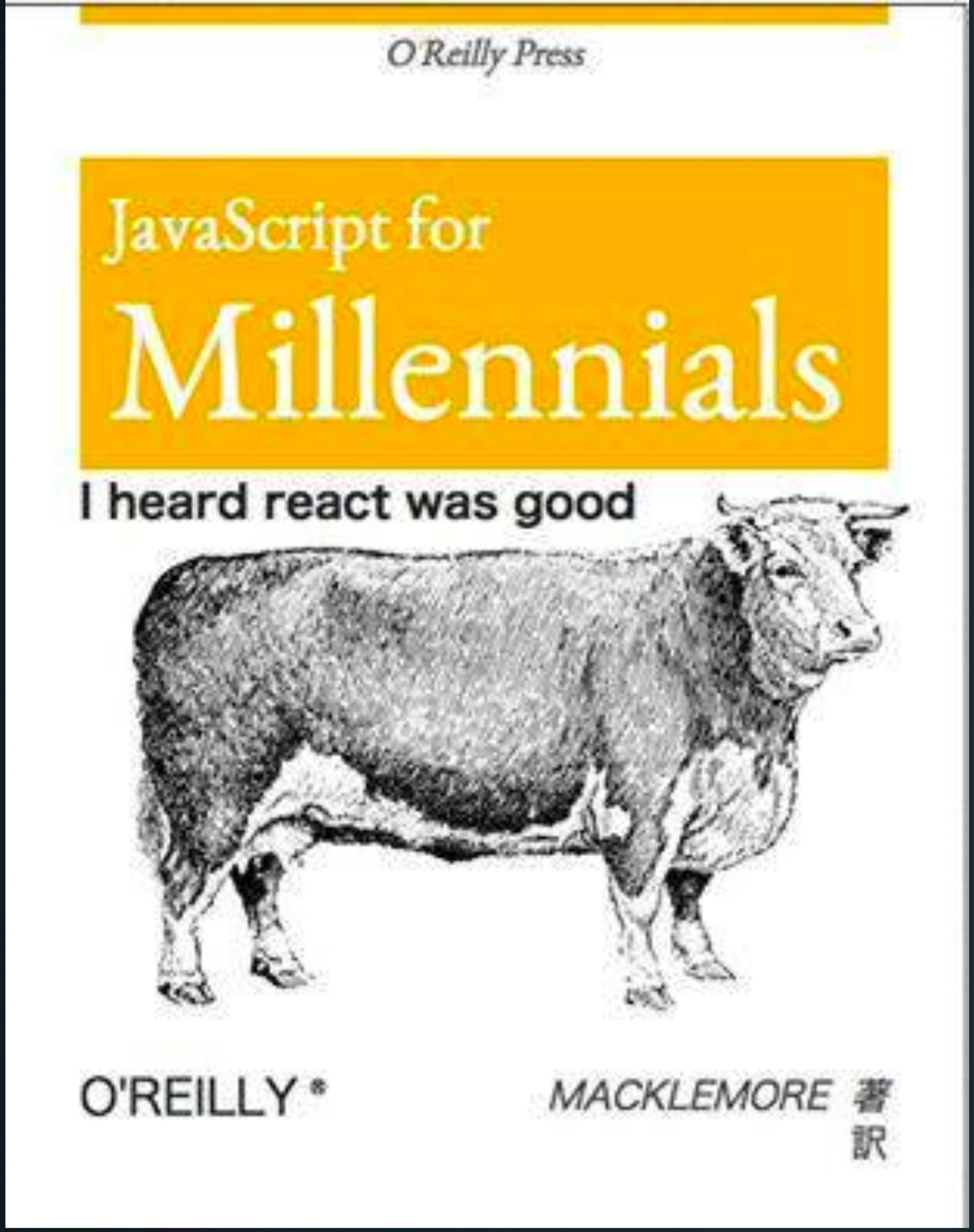
Twitter advising all 330 million users to change passwords after bug exposed them in plain text

There's apparently no evidence of any breach or misuse, but you should change your password anyway

By Chaim Gartenberg | @cgartenberg | May 3, 2018, 4:21pm EDT

Building a plan to integrate WebAuthn

Development



O'Reilly Press

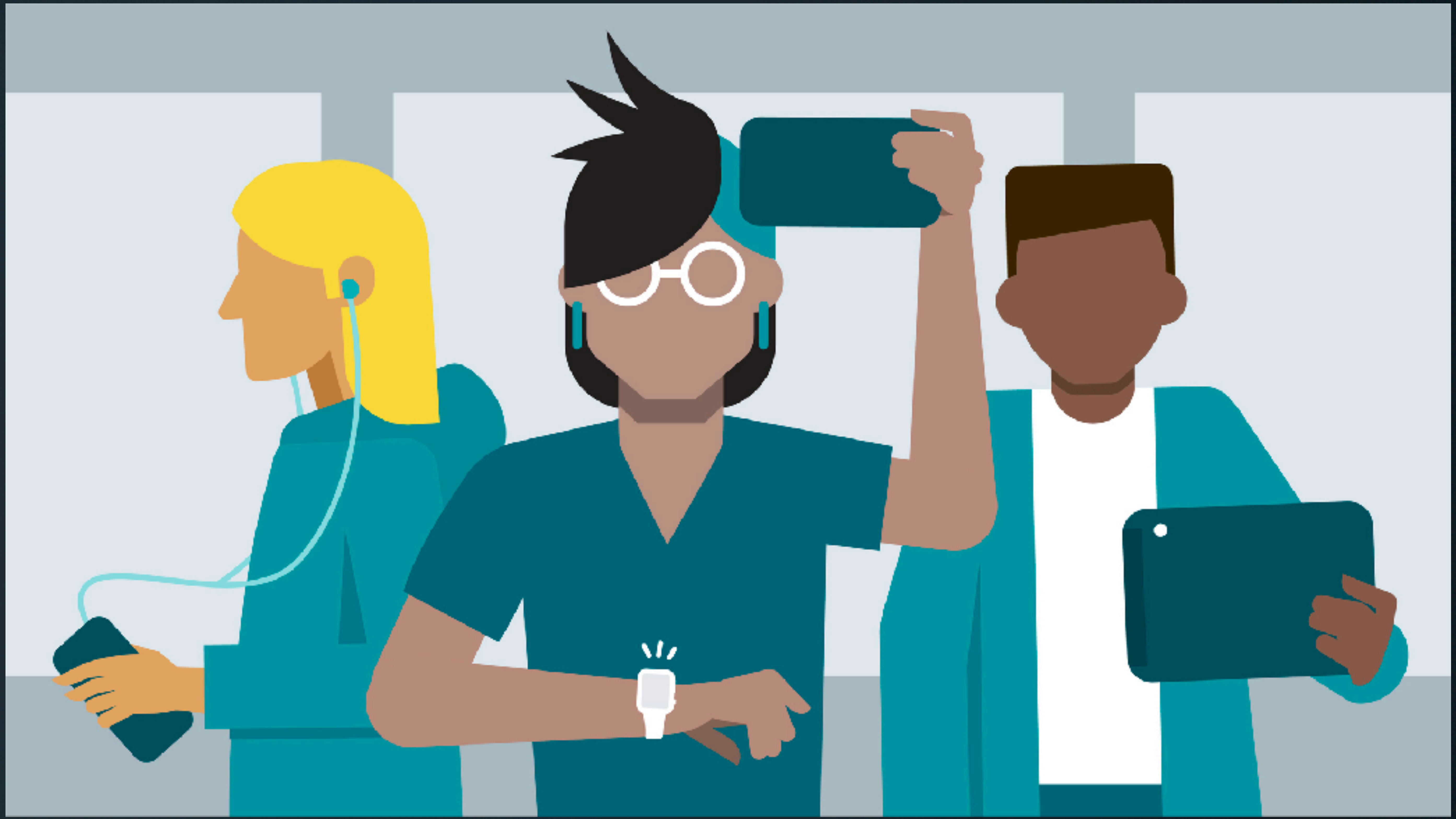
JavaScript for

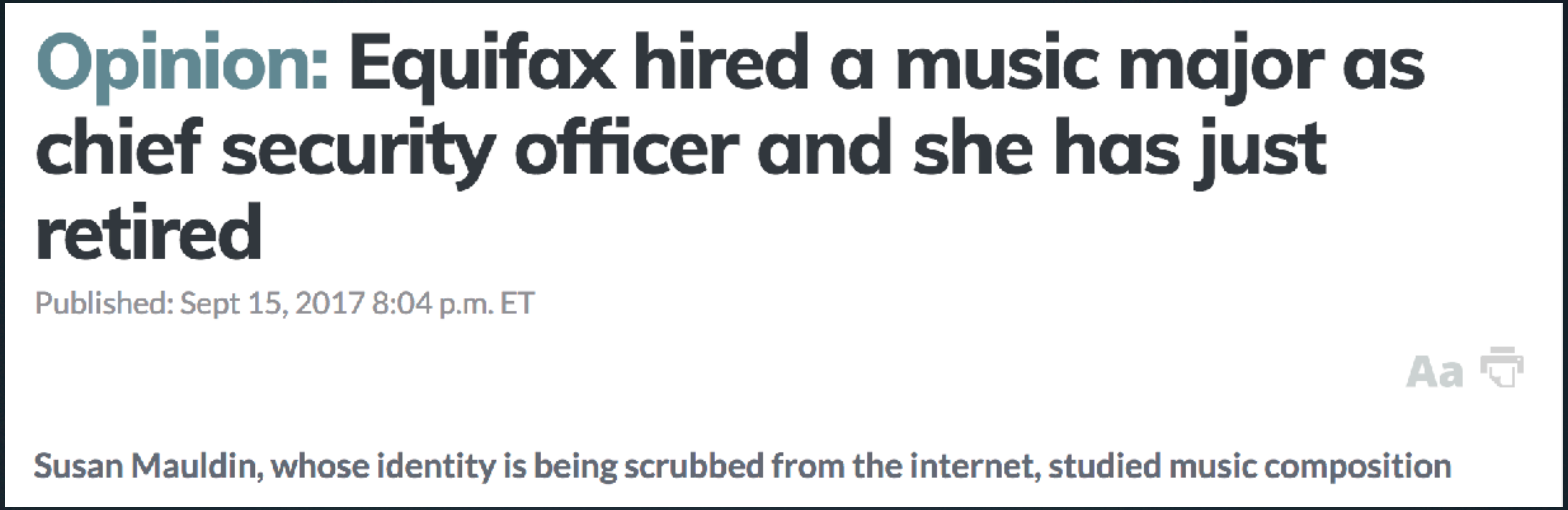
Millennials

I heard react was good

O'REILLY®

MACKLEMORE 著
訳

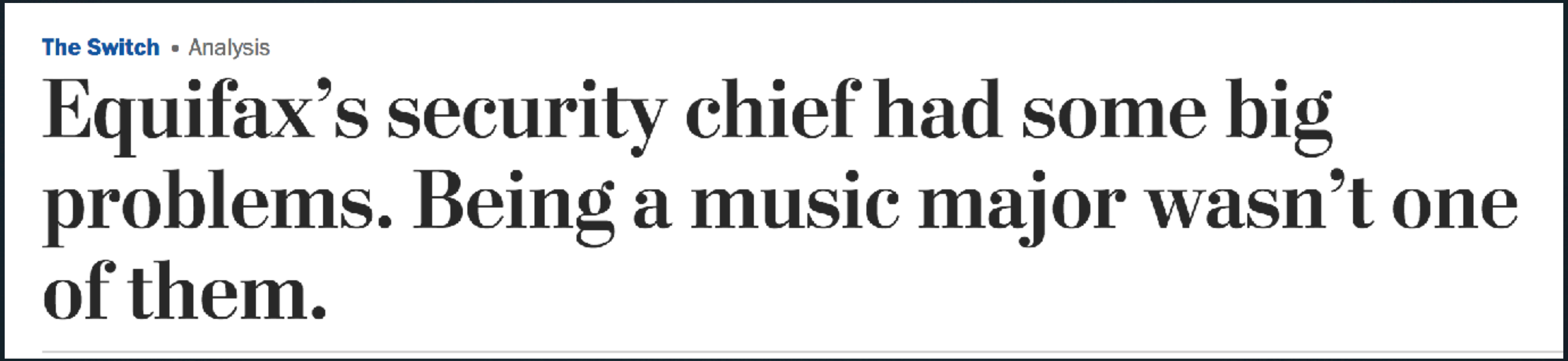




Opinion: Equifax hired a music major as chief security officer and she has just retired

Published: Sept 15, 2017 8:04 p.m. ET

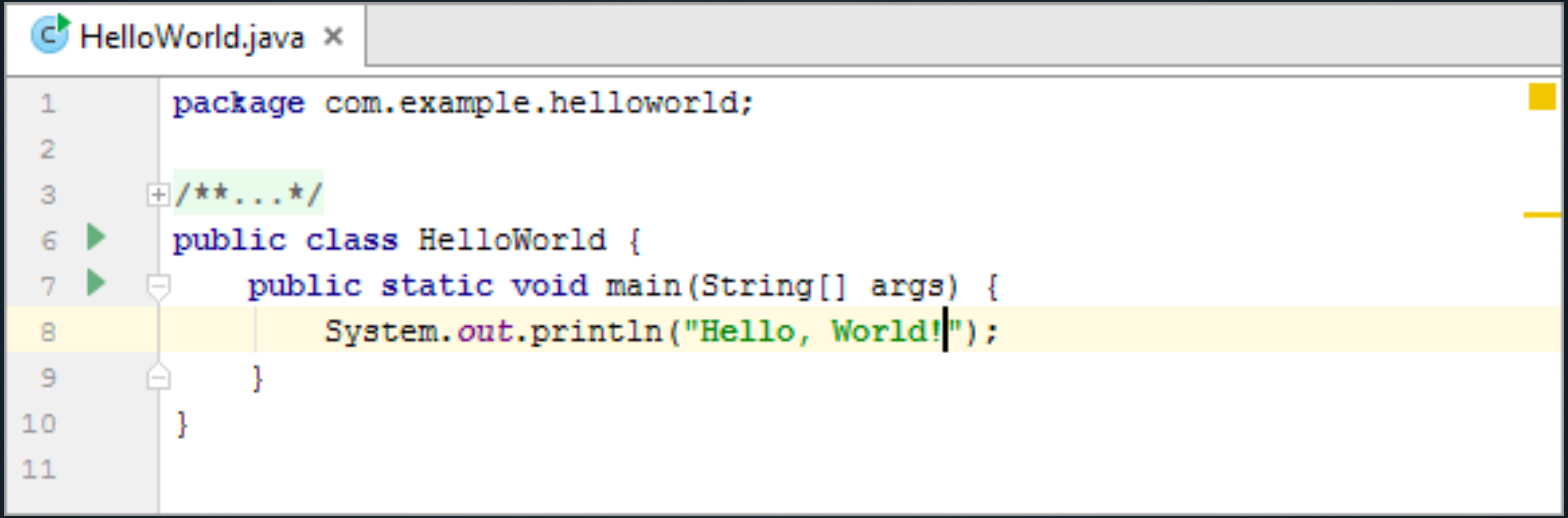
Susan Mauldin, whose identity is being scrubbed from the internet, studied music composition



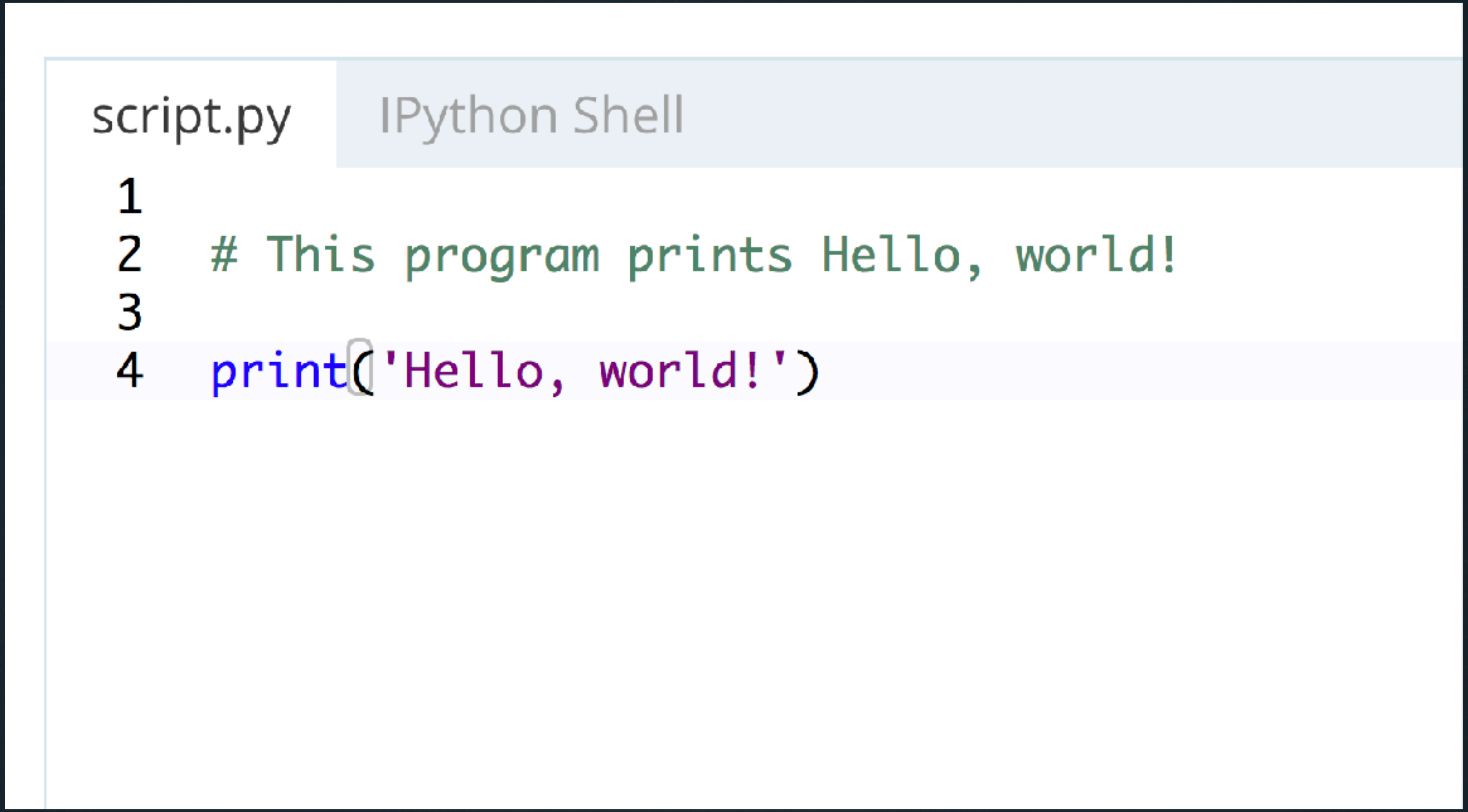
The Switch • Analysis

Equifax’s security chief had some big problems. Being a music major wasn’t one of them.

Programming: First Impressions

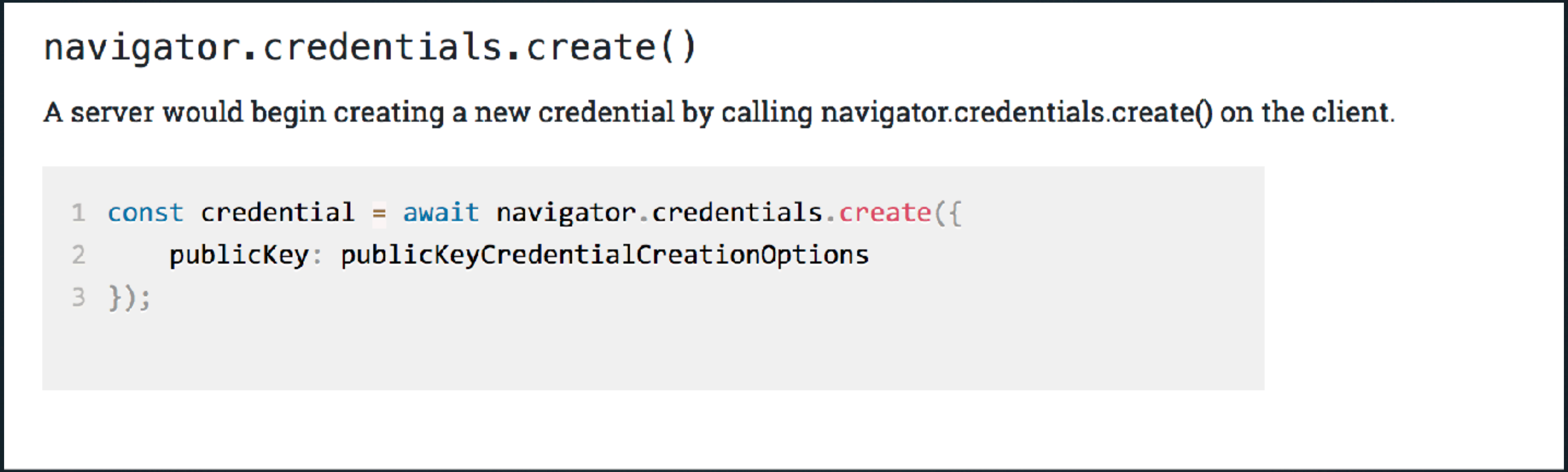


```
package com.example.helloworld;

/**...*/
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}

```



```python

# This program prints Hello, world!

print('Hello, world!')
```

The WebAuthn First Impression

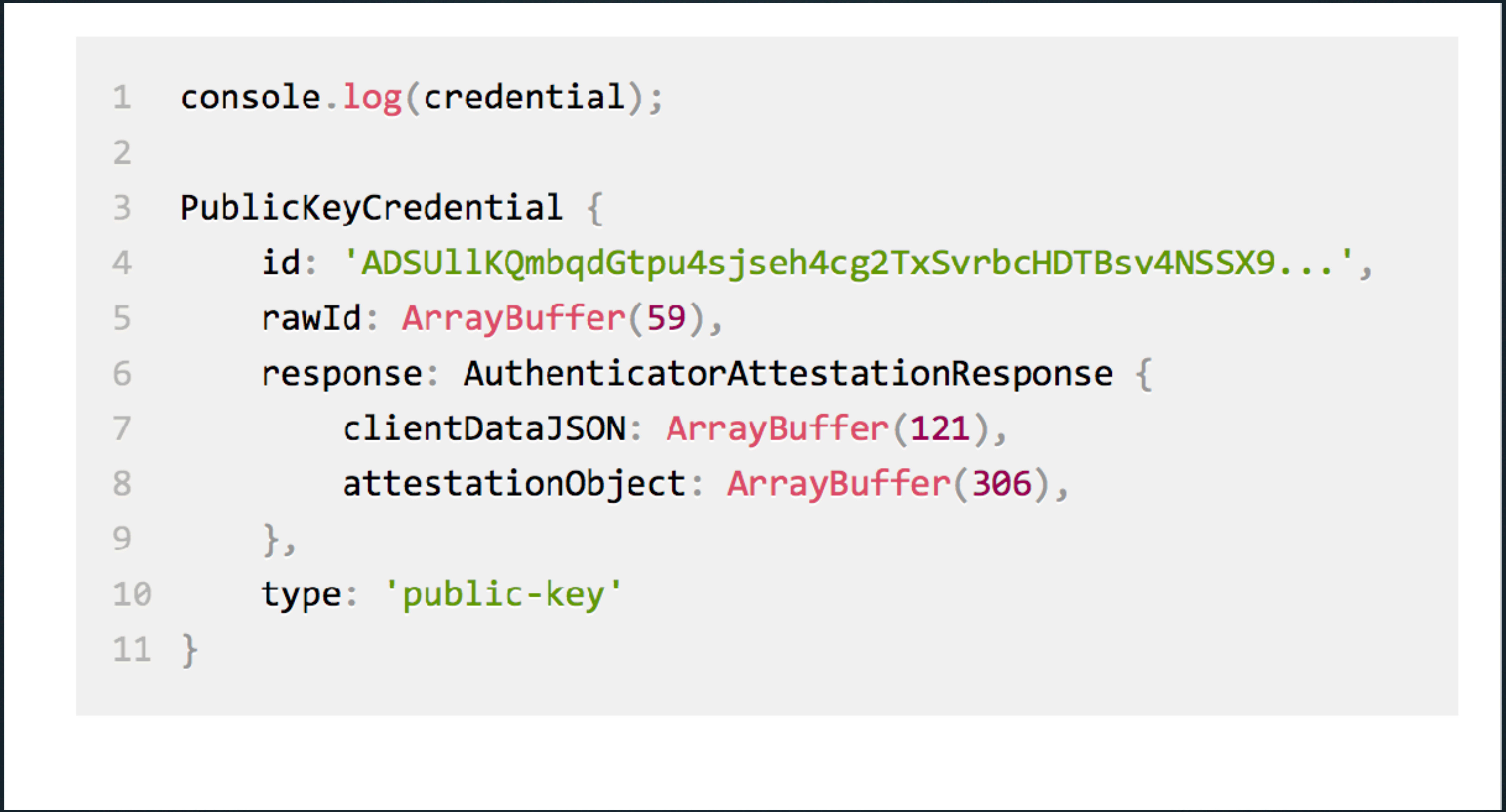


navigator.credentials.create()

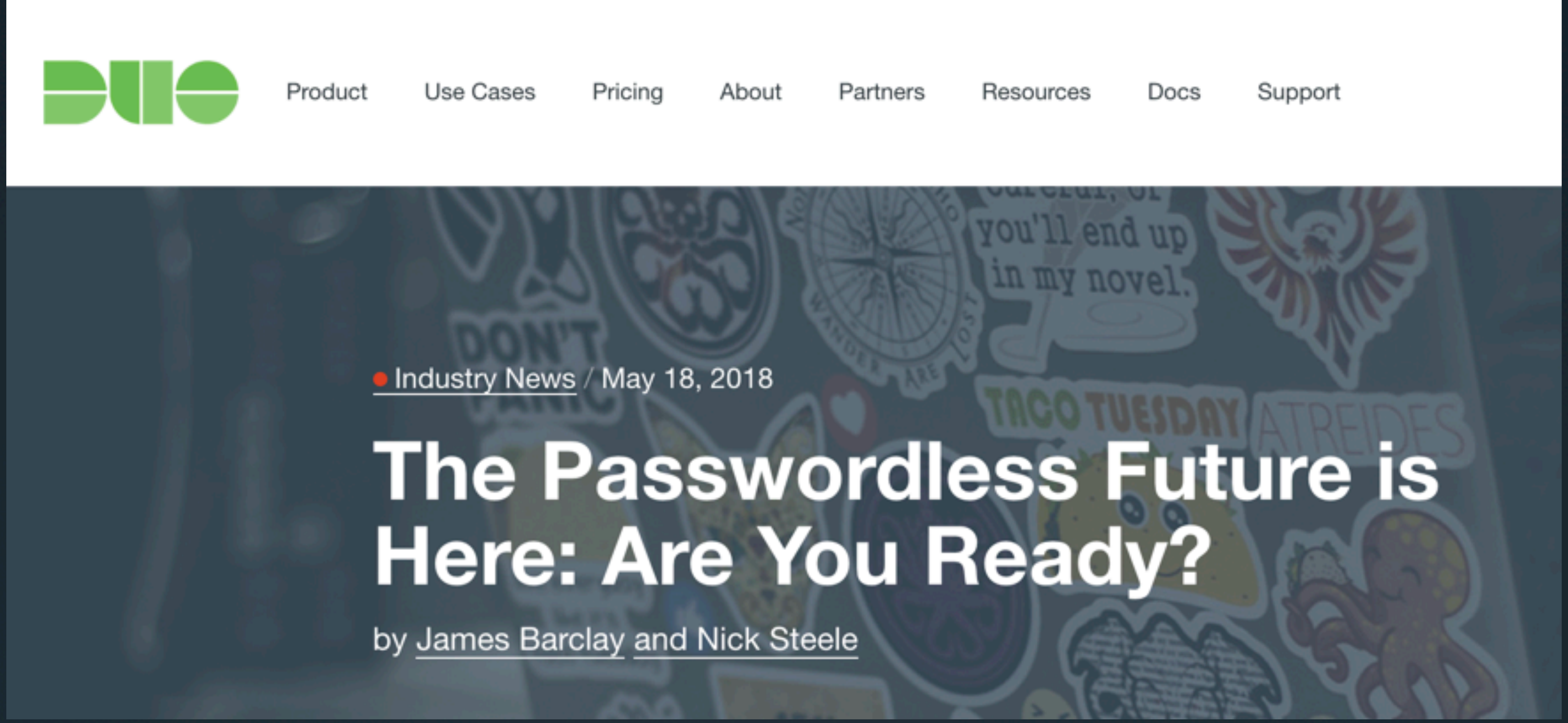
A server would begin creating a new credential by calling navigator.credentials.create() on the client.

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

The WebAuthn First Impression



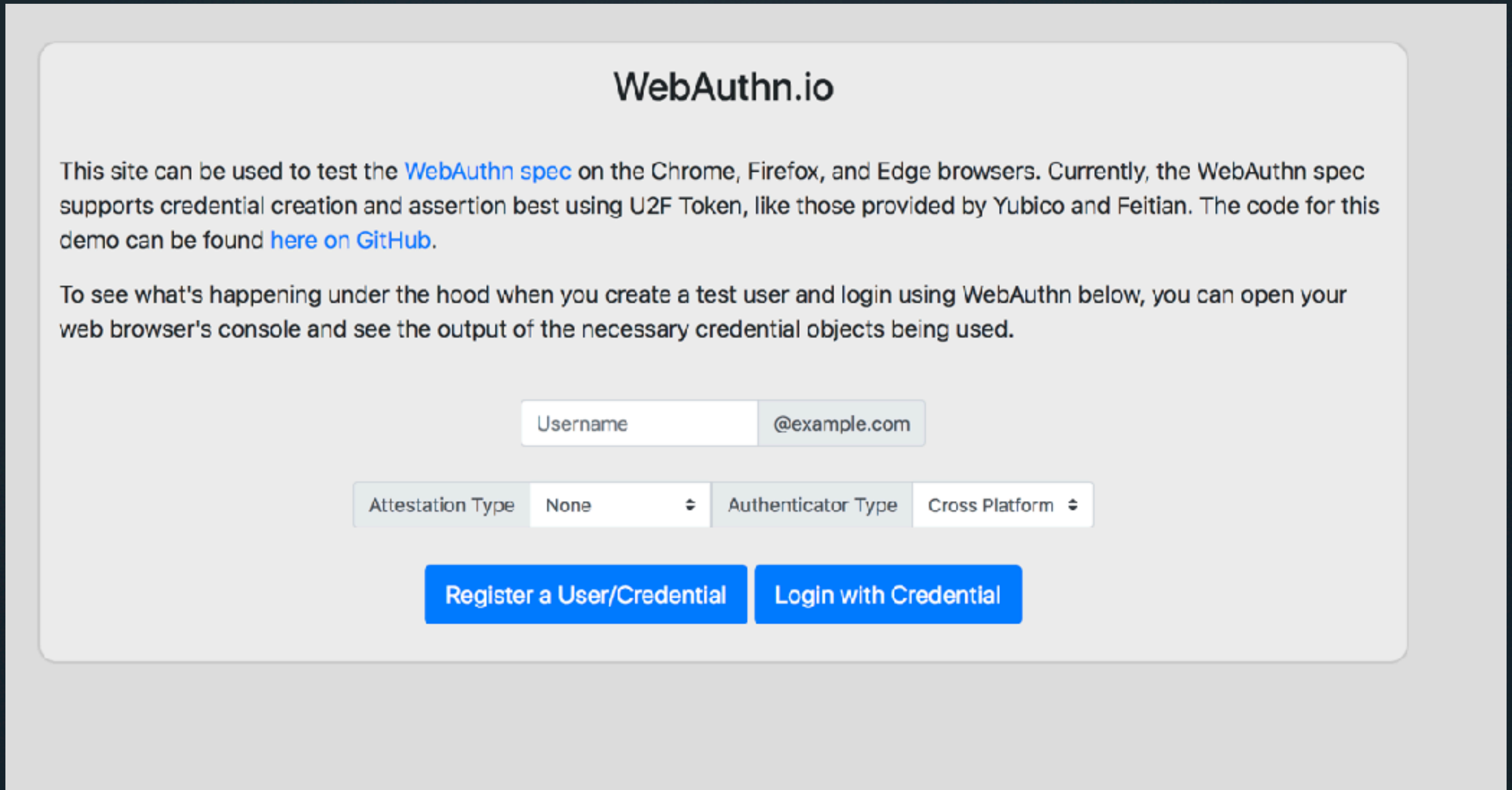
```
console.log(credential);

PublicKeyCredential {
    id: 'ADSUllKQmbqdGtpu4sjseh4cg2TxSvrbcHDTBsv4NSSX9...',
    rawId: ArrayBuffer(59),
    response: AuthenticatorAttestationResponse {
        clientDataJSON: ArrayBuffer(121),
        attestationObject: ArrayBuffer(306),
    },
    type: 'public-key'
}
```



● Industry News / May 18, 2018

The Passwordless Future is Here: Are You Ready?

by James Barclay and Nick Steele



WebAuthn.io

This site can be used to test the WebAuthn spec on the Chrome, Firefox, and Edge browsers. Currently, the WebAuthn spec supports credential creation and assertion best using U2F Token, like those provided by Yubico and Feitian. The code for this demo can be found here on GitHub.

To see what's happening under the hood when you create a test user and login using WebAuthn below, you can open your web browser's console and see the output of the necessary credential objects being used.

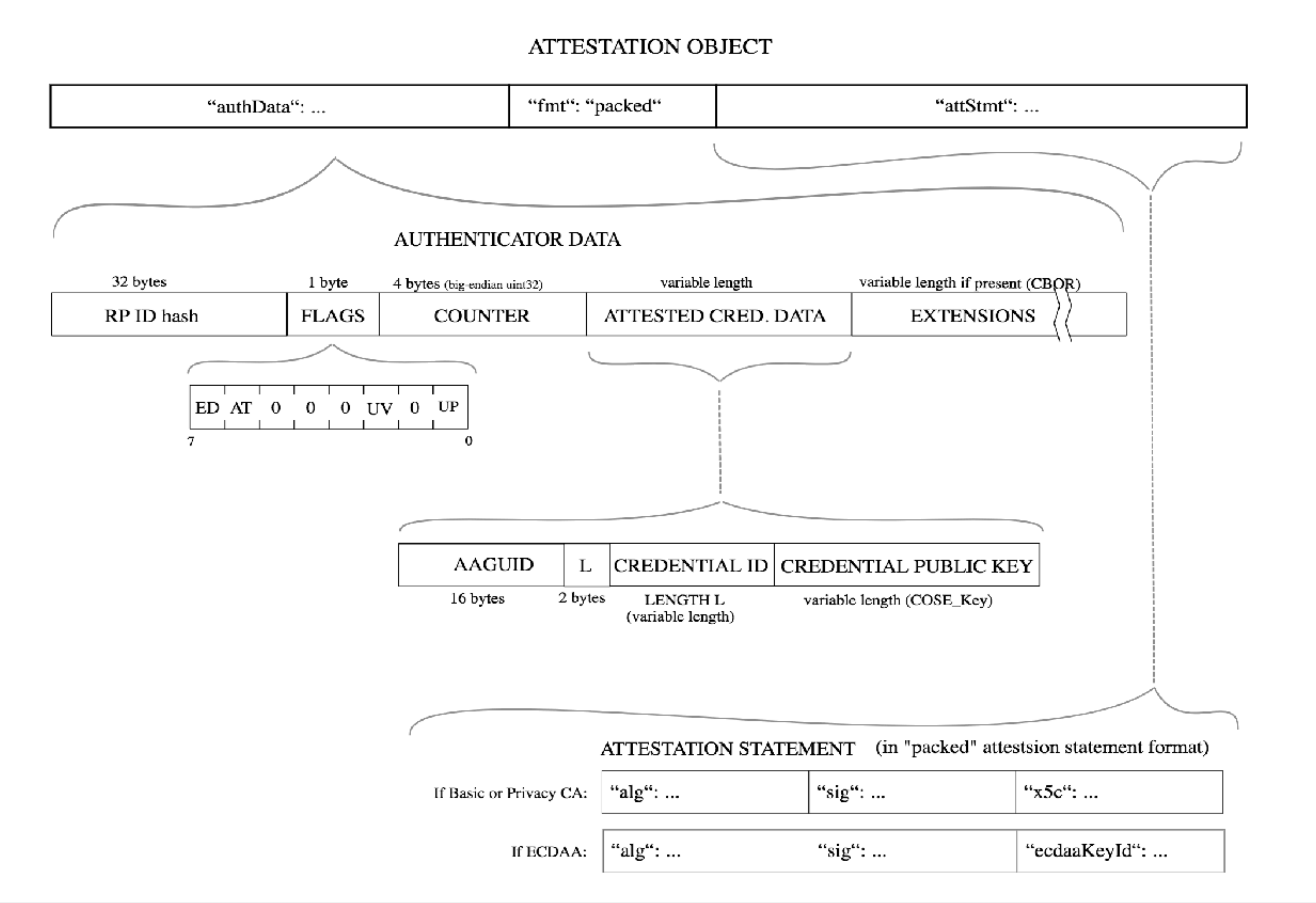
Username @example.com

Attestation Type: None

Authenticator Type: Cross Platform

Breaking down WebAuthn

attestationObject



ATTESTATION OBJECT

"authData": ... | "fmt": "packed" | "attStmt": ...

AUTHENTICATOR DATA

32 bytes	1 byte	4 bytes (big-endian uint32)	variable length	variable length if present (CBOR)
RP ID hash	FLAGS	COUNTER	ATTESTED CRED. DATA	EXTENSIONS

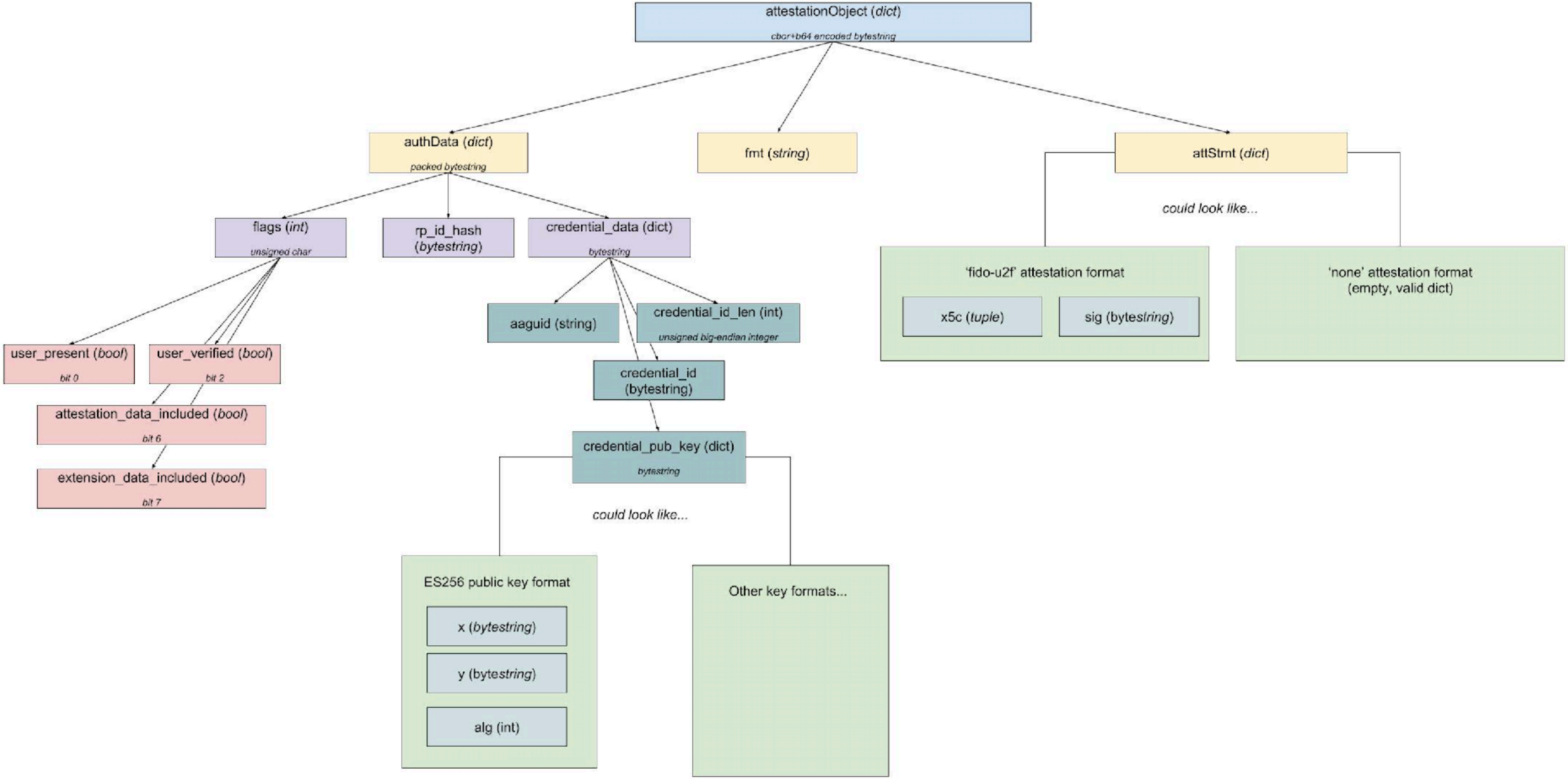
ED	AT	0	0	0	UV	0	UP

7 … 0

AAGUID	L	CREDENTIAL ID	CREDENTIAL PUBLIC KEY
16 bytes	2 bytes	LENGTH L (variable length)	variable length (COSE_Key)

ATTESTATION STATEMENT (in "packed" attestsion statement format)

If Basic or Privacy CA:	"alg": ...	"sig": ...	"x5c": ...
If ECDAA:	"alg": ...	"sig": ...	"ecdaaKeyId": ...



attestationObject (*dict*)
cbor+b64 encoded bytestring

authData (*dict*)
packed bytestring

fmt (*string*)

attStmt (*dict*)

flags (*int*)
unsigned char

rp_id_hash (*bytestring*)

credential_data (dict)
bytestring

user_present (*bool*)
bit 0

user_verified (*bool*)
bit 2

attestation_data_included (*bool*)
bit 6

extension_data_included (*bool*)
bit 7

aaguid (string)

credential_id_len (int)
unsigned big-endian integer

credential_id (bytestring)

credential_pub_key (dict)
bytestring

could look like...

ES256 public key format

x (*bytestring*)

y (bytes*tring*)

alg (int)

Other key formats...

could look like...

'fido-u2f' attestation format

x5c (*tuple*)

sig (bytes*tring*)

'none' attestation format
(empty, valid dict)

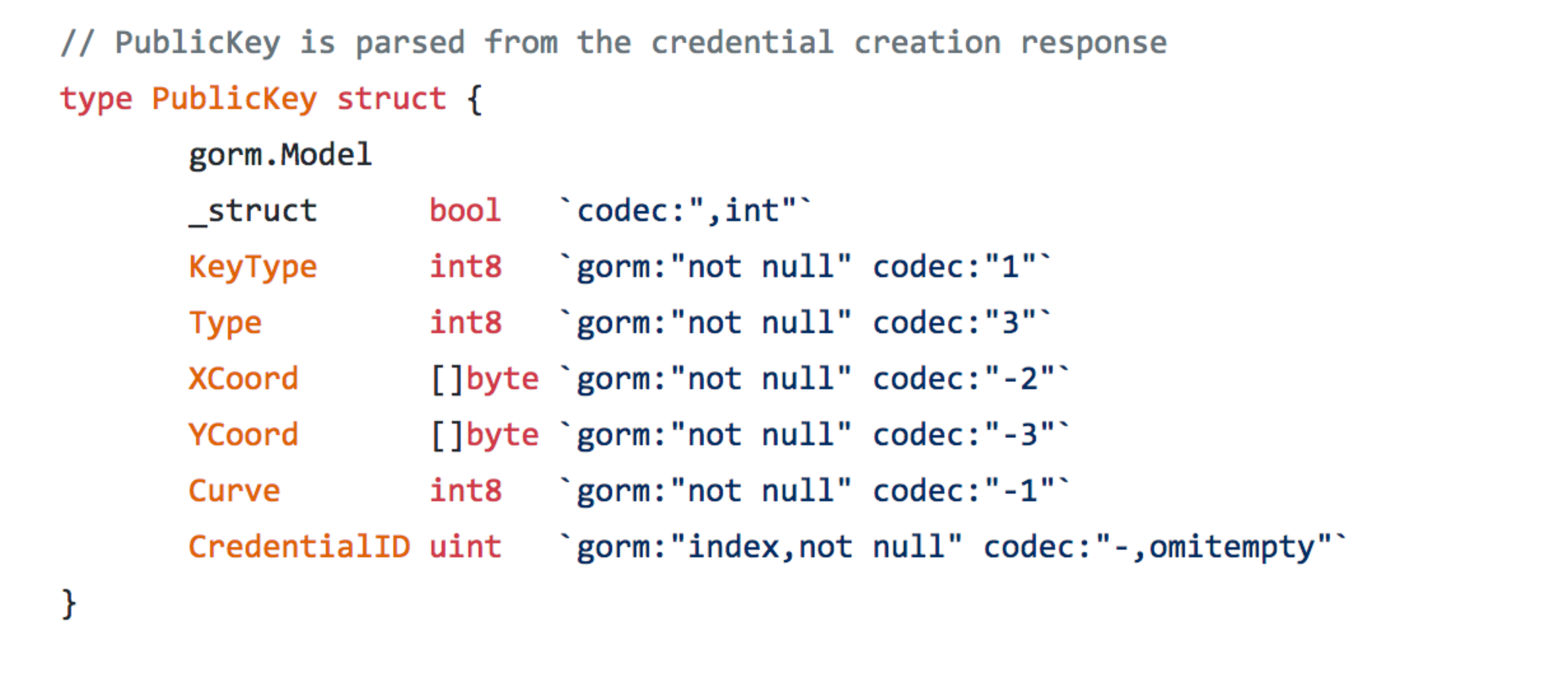
Recommendation:

Use an extensible library for processing WebAuthn data

Type coercion

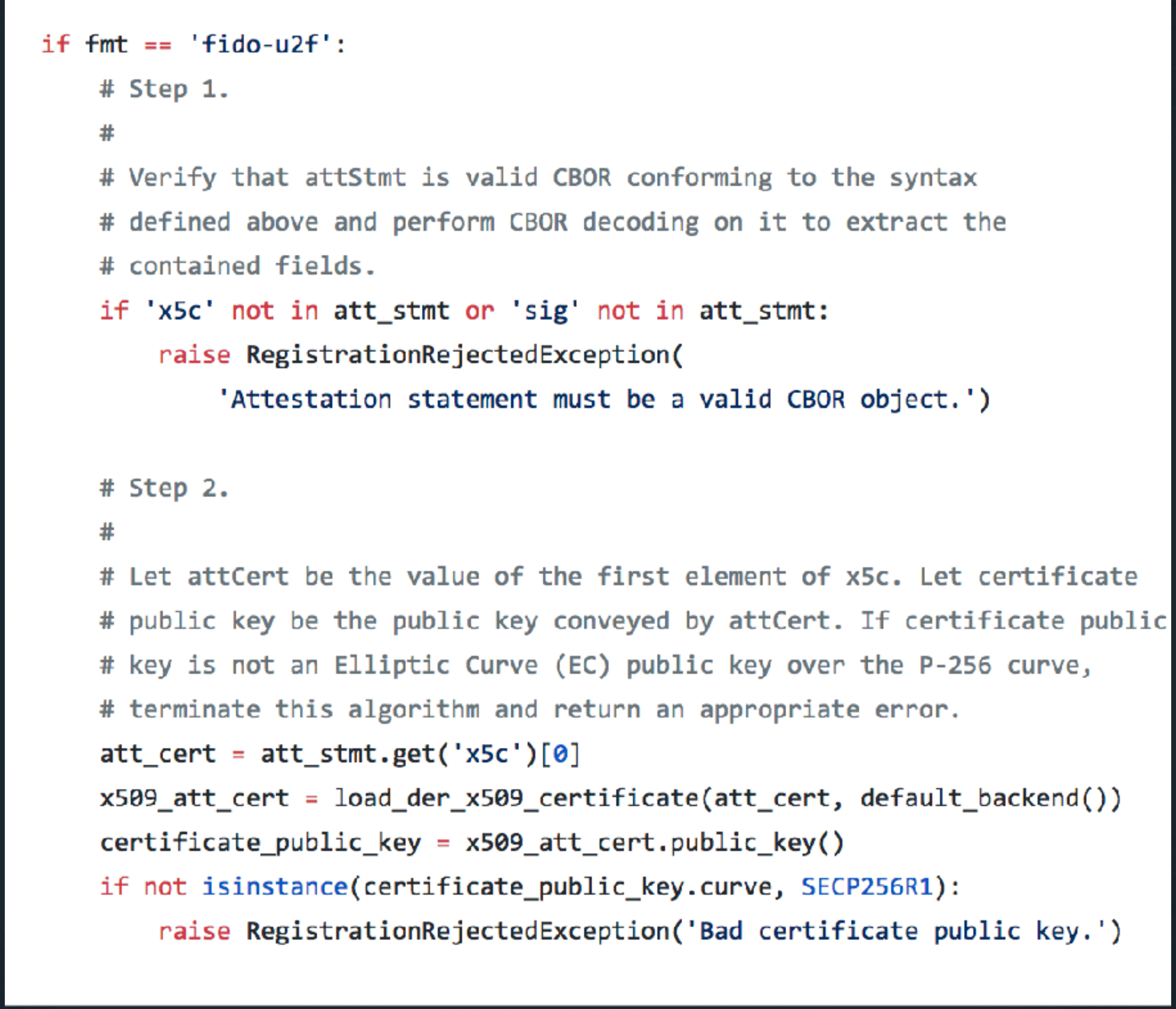
validation

https://github.com/duo-labs/webauthn/



```go
// PublicKey is parsed from the credential creation response
type PublicKey struct {
	gorm.Model
	_struct      bool   `codec:",int"`
	KeyType      int8   `gorm:"not null" codec:"1"`
	Type         int8   `gorm:"not null" codec:"3"`
	XCoord       []byte `gorm:"not null" codec:"-2"`
	YCoord       []byte `gorm:"not null" codec:"-3"`
	Curve        int8   `gorm:"not null" codec:"-1"`
	CredentialID uint   `gorm:"index,not null" codec:"-,omitempty"`
}
```

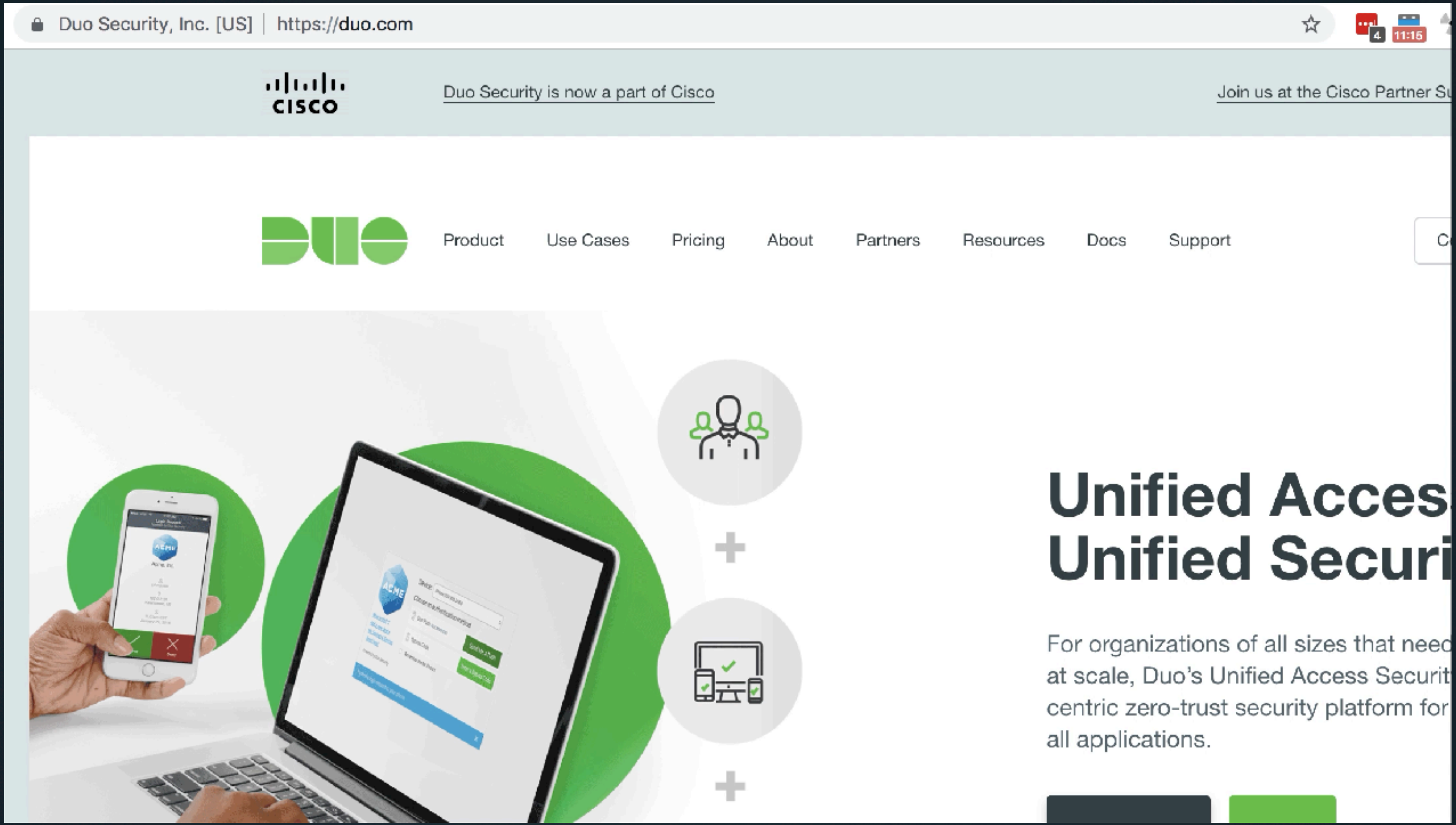
https://github.com/duo-labs/py_webauthn



```
if fmt == 'fido-u2f':
    # Step 1.
    #
    # Verify that attStmt is valid CBOR conforming to the syntax
    # defined above and perform CBOR decoding on it to extract the
    # contained fields.
    if 'x5c' not in att_stmt or 'sig' not in att_stmt:
        raise RegistrationRejectedException(
            'Attestation statement must be a valid CBOR object.')

    # Step 2.
    #
    # Let attCert be the value of the first element of x5c. Let certificate
    # public key be the public key conveyed by attCert. If certificate public
    # key is not an Elliptic Curve (EC) public key over the P-256 curve,
    # terminate this algorithm and return an appropriate error.
    att_cert = att_stmt.get('x5c')[0]
    x509_att_cert = load_der_x509_certificate(att_cert, default_backend())
    certificate_public_key = x509_att_cert.public_key()
    if not isinstance(certificate_public_key.curve, SECP256R1):
        raise RegistrationRejectedException('Bad certificate public key.')
```

Recommendation:
Start with Chrome's Touch ID implementation.

Built into user's device
Simple data verification process



Duo Security, Inc. [US] | https://duo.com

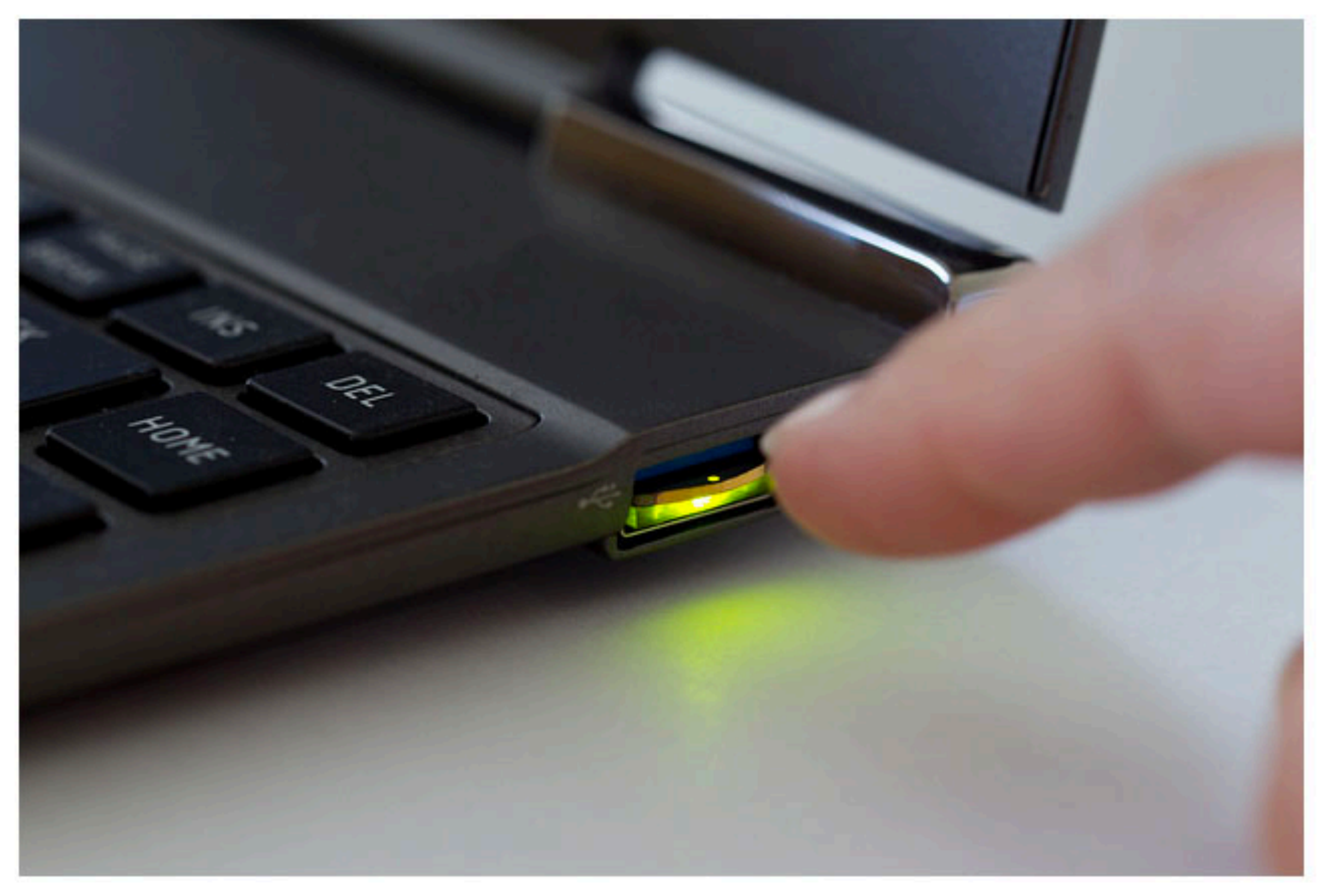
Duo Security is now a part of Cisco

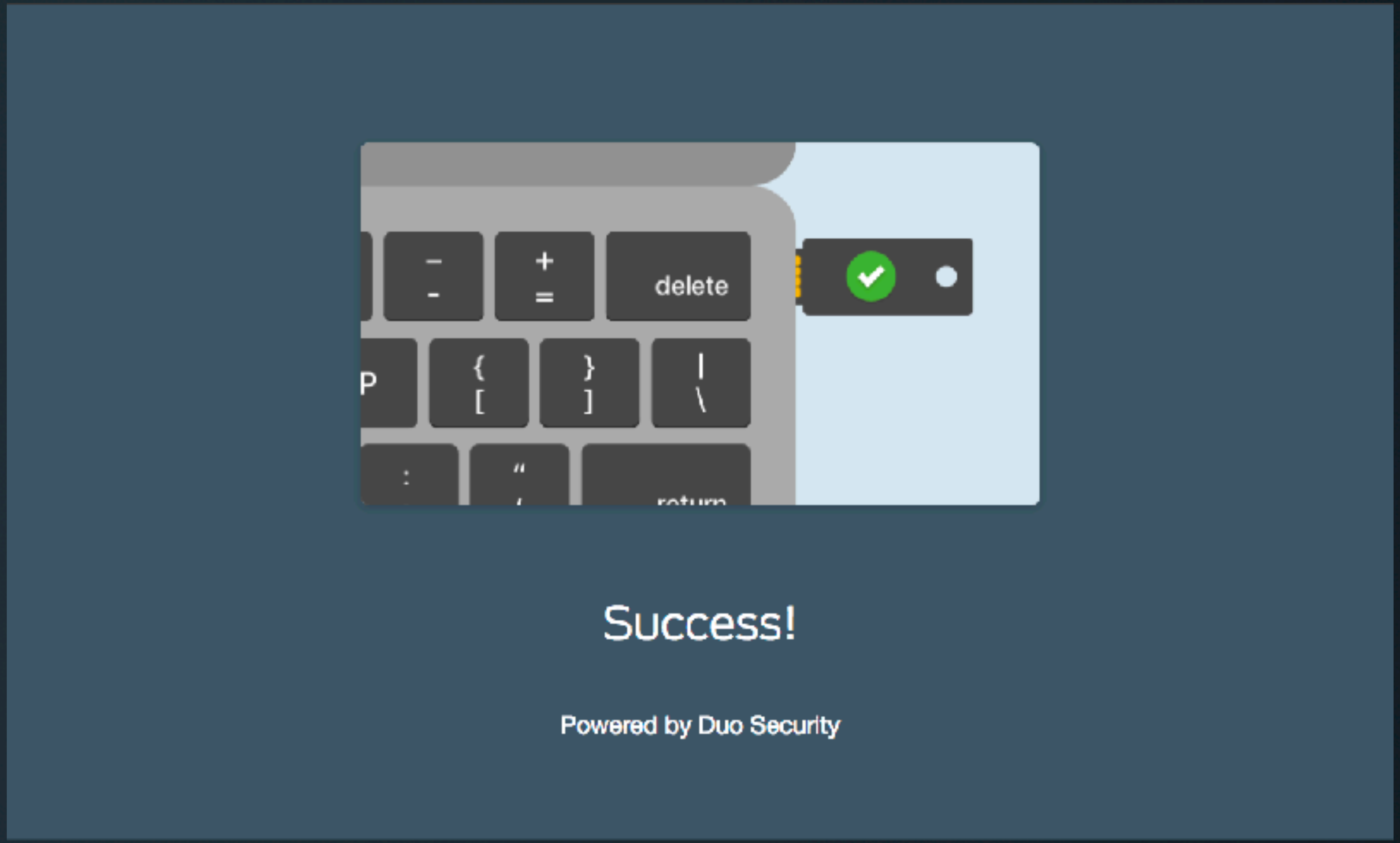
Join us at the Cisco Partner Su

Unified Acces
Unified Securi

For organizations of all sizes that need
at scale, Duo's Unified Access Securit
centric zero-trust security platform for
all applications.

Design

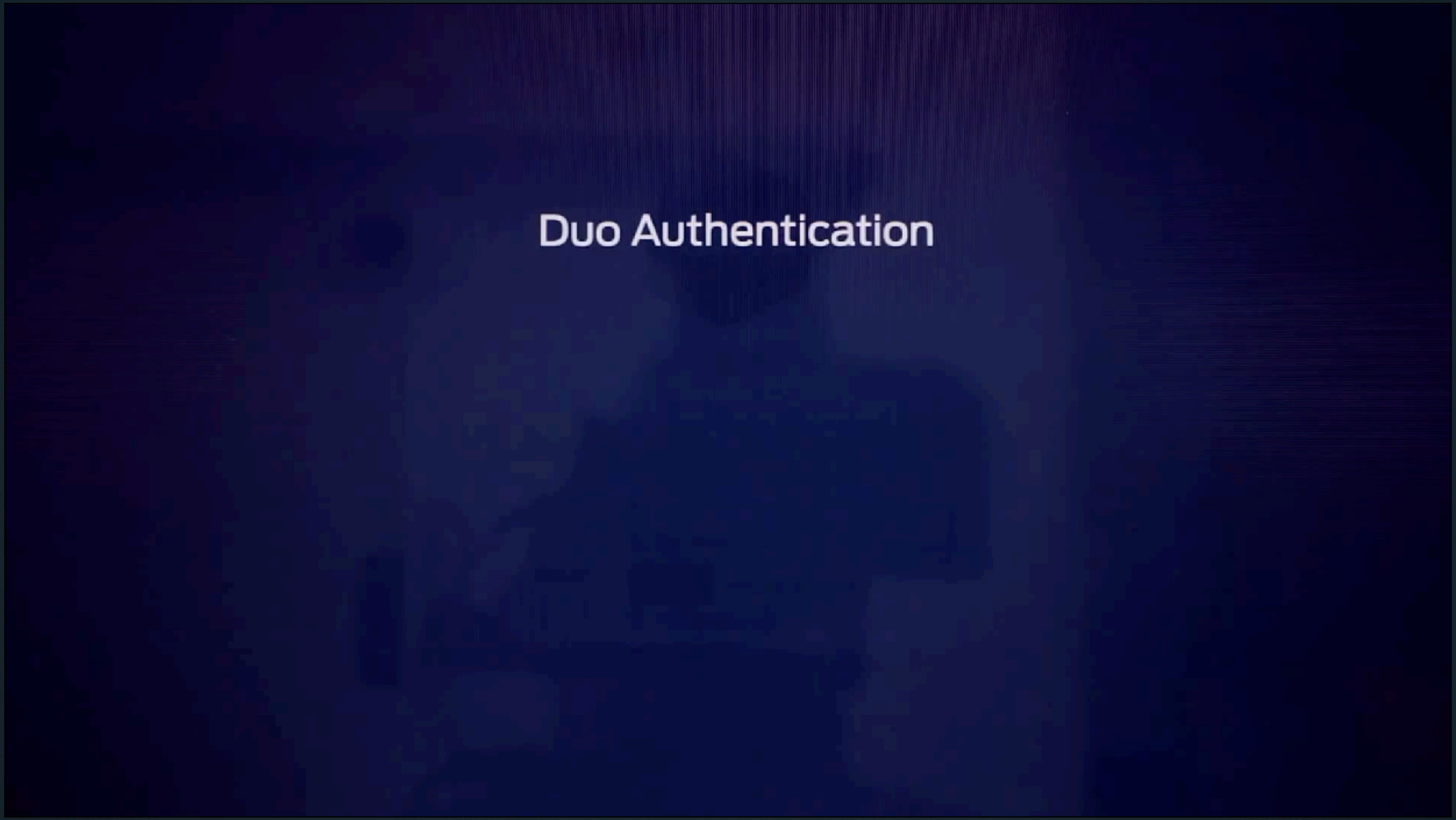




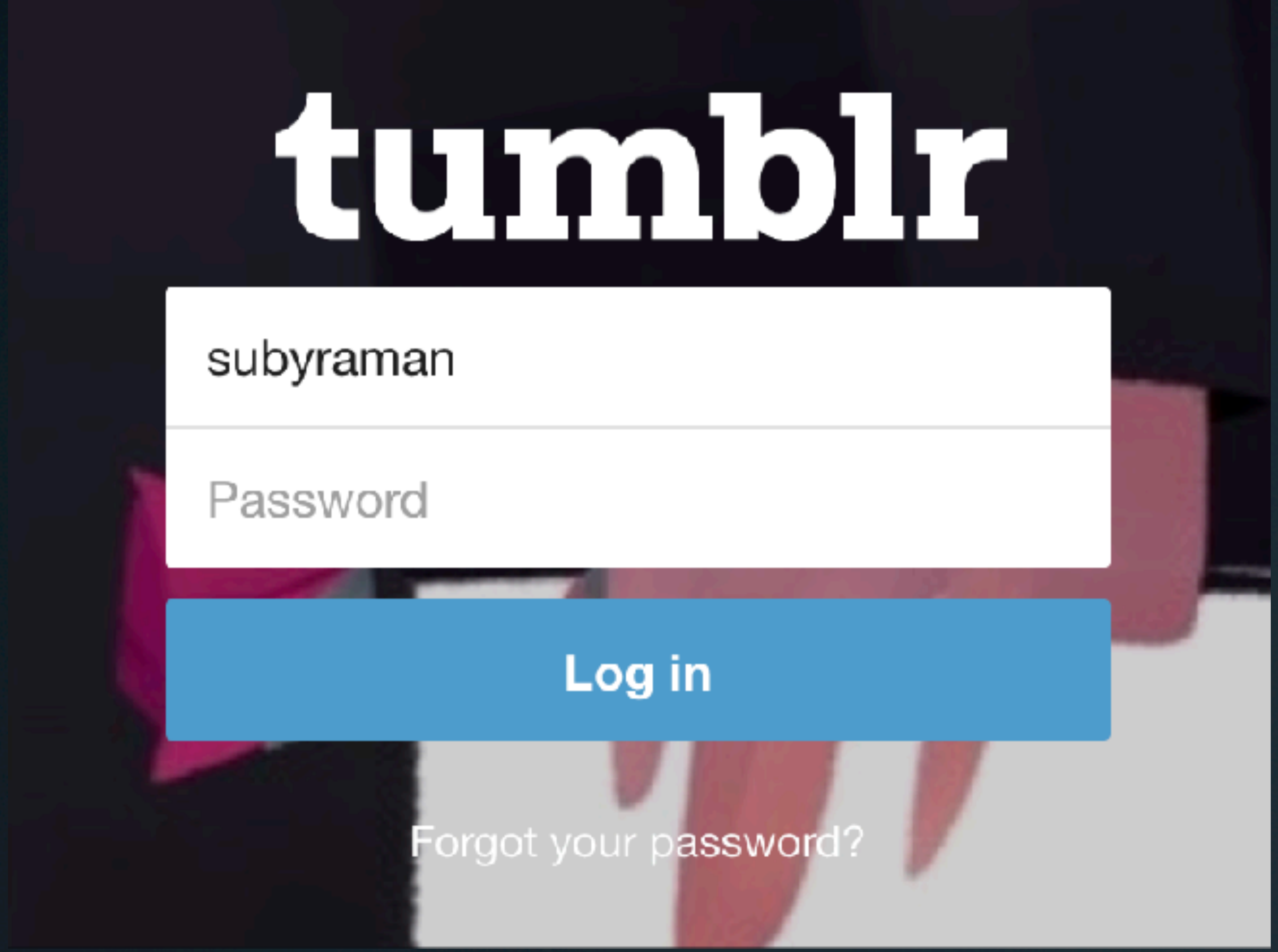
Success!

Powered by Duo Security

What do we even call this thing?



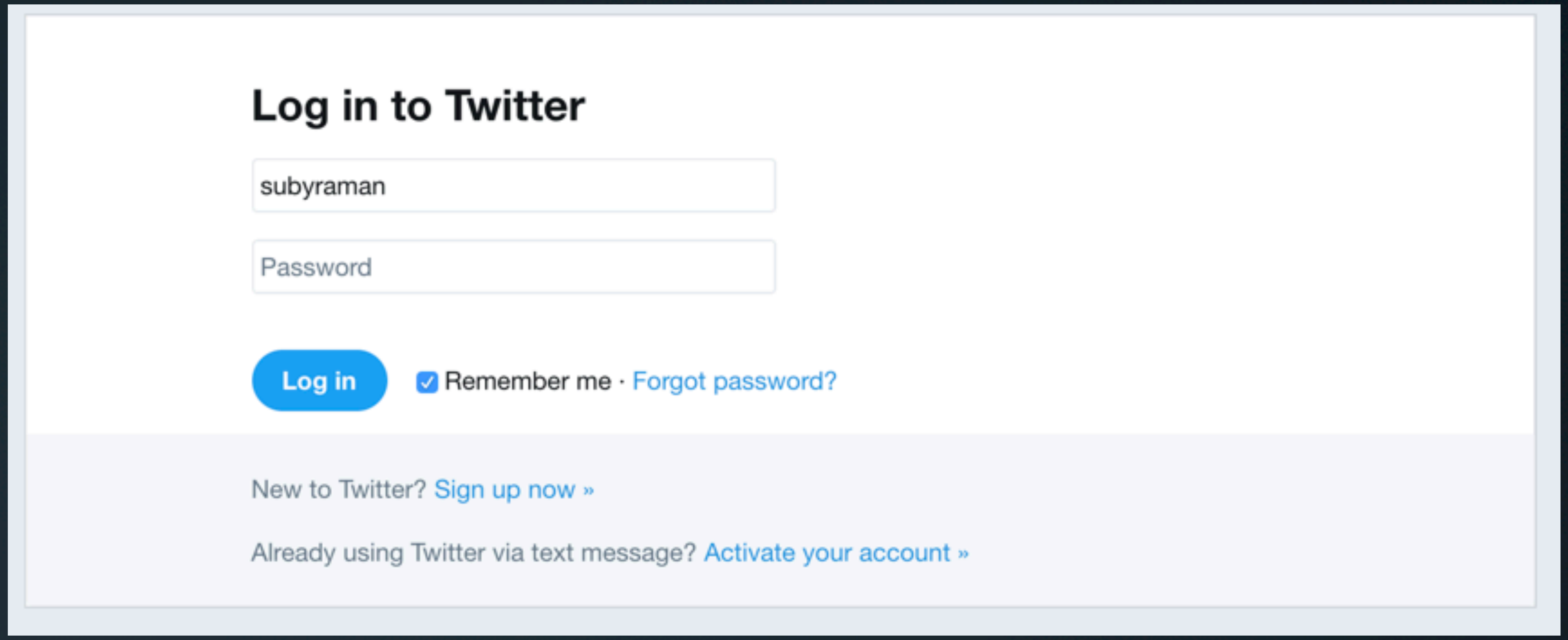
Duo Authentication



tumblr

subyraman

Password



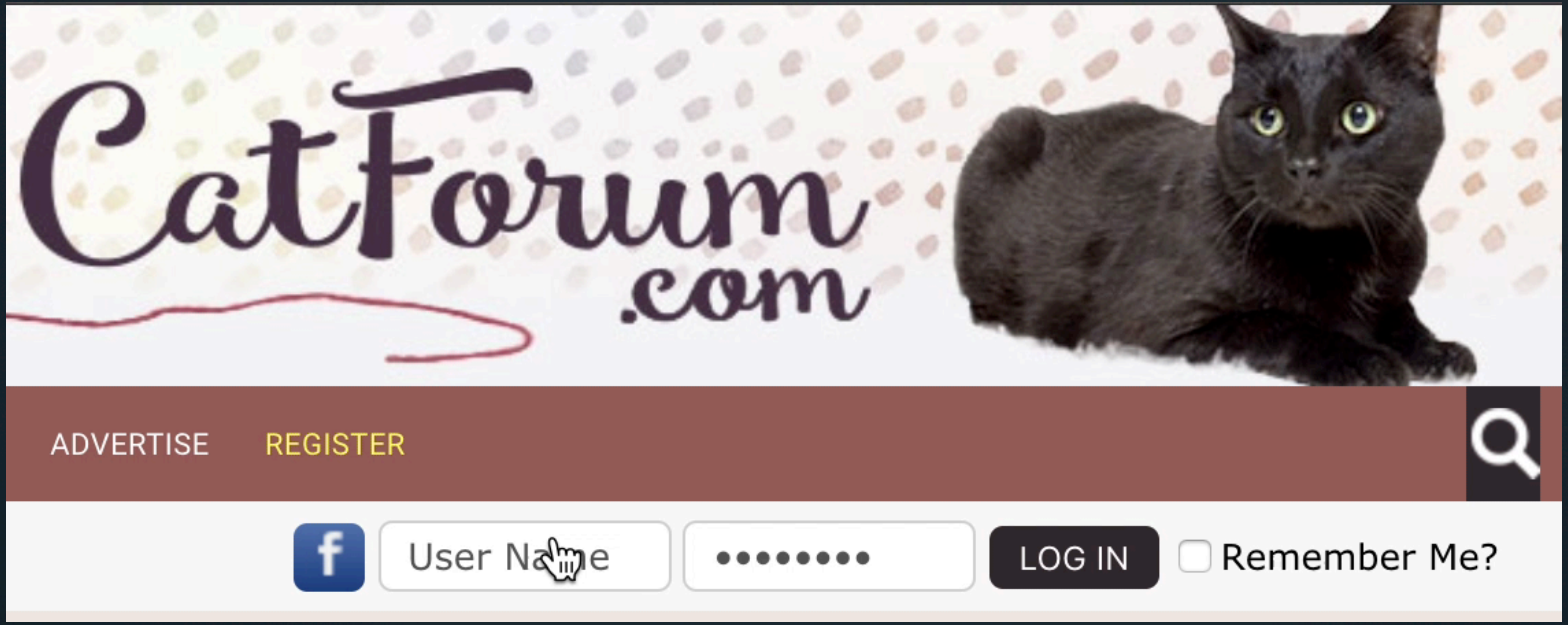
Log in to Twitter

subyraman

Password

New to Twitter? Sign up now »

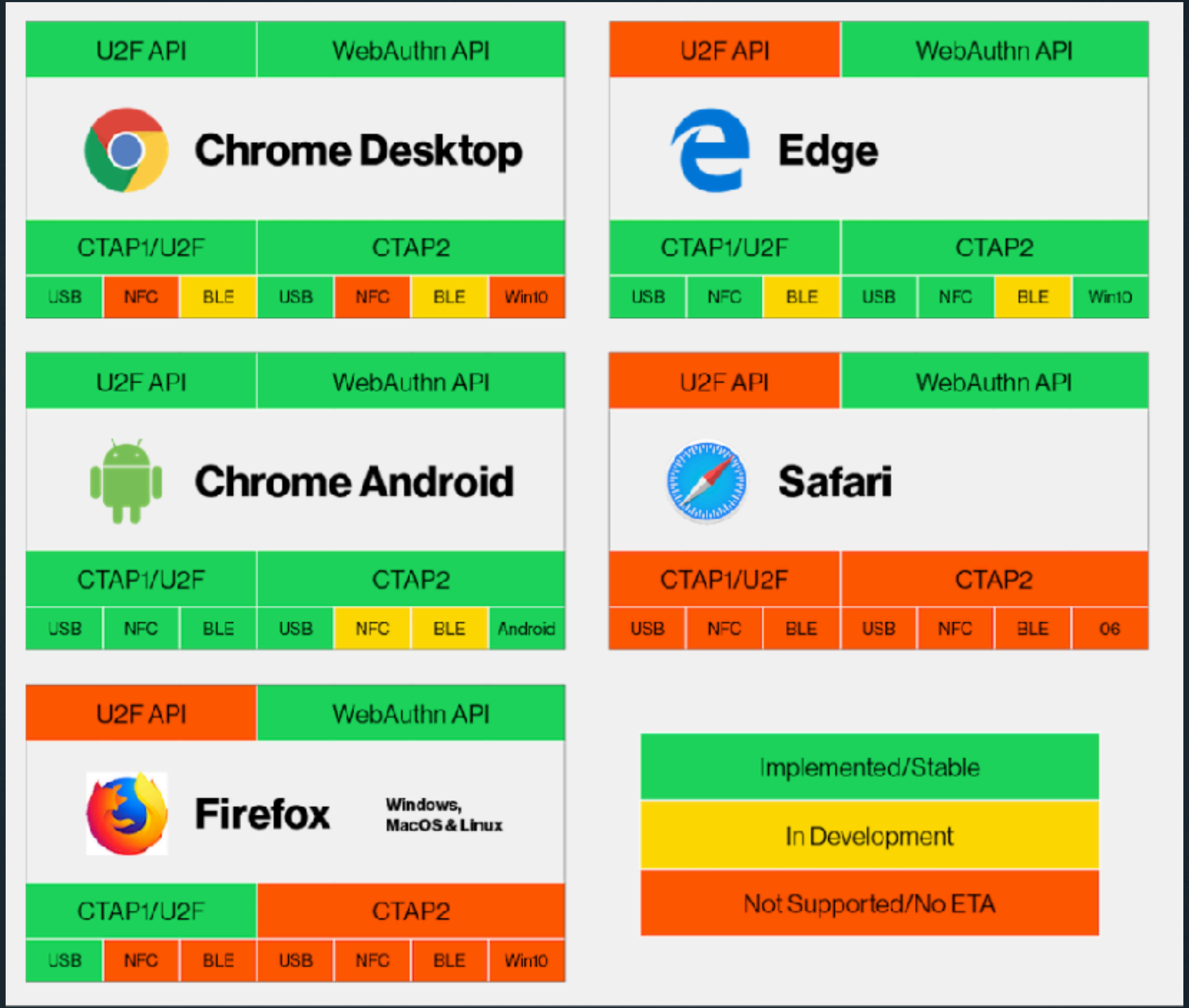
Already using Twitter via text message? Activate your account »



User Name

••••••••

User Agent Implementation Differences



Chrome Desktop

🟩 U2F API			🟩 WebAuthn API			
🟩 CTAP1/U2F			🟩 CTAP2			
🟩 USB	🟧 NFC	🟨 BLE	🟩 USB	🟧 NFC	🟨 BLE	🟧 Win10

Edge

🟧 U2F API			🟩 WebAuthn API			
🟩 CTAP1/U2F			🟩 CTAP2			
🟩 USB	🟩 NFC	🟨 BLE	🟩 USB	🟩 NFC	🟨 BLE	🟩 Win10

Chrome Android

🟩 U2F API			🟩 WebAuthn API			
🟩 CTAP1/U2F			🟩 CTAP2			
🟩 USB	🟩 NFC	🟩 BLE	🟩 USB	🟨 NFC	🟨 BLE	🟩 Android

Safari

🟧 U2F API			🟩 WebAuthn API			
🟧 CTAP1/U2F			🟧 CTAP2			
🟧 USB	🟧 NFC	🟧 BLE	🟧 USB	🟧 NFC	🟧 BLE	🟧 06

Firefox Windows, MacOS & Linux

🟧 U2F API			🟩 WebAuthn API			
🟩 CTAP1/U2F			🟧 CTAP2			
🟩 USB	🟧 NFC	🟧 BLE	🟧 USB	🟧 NFC	🟧 BLE	🟧 Win10

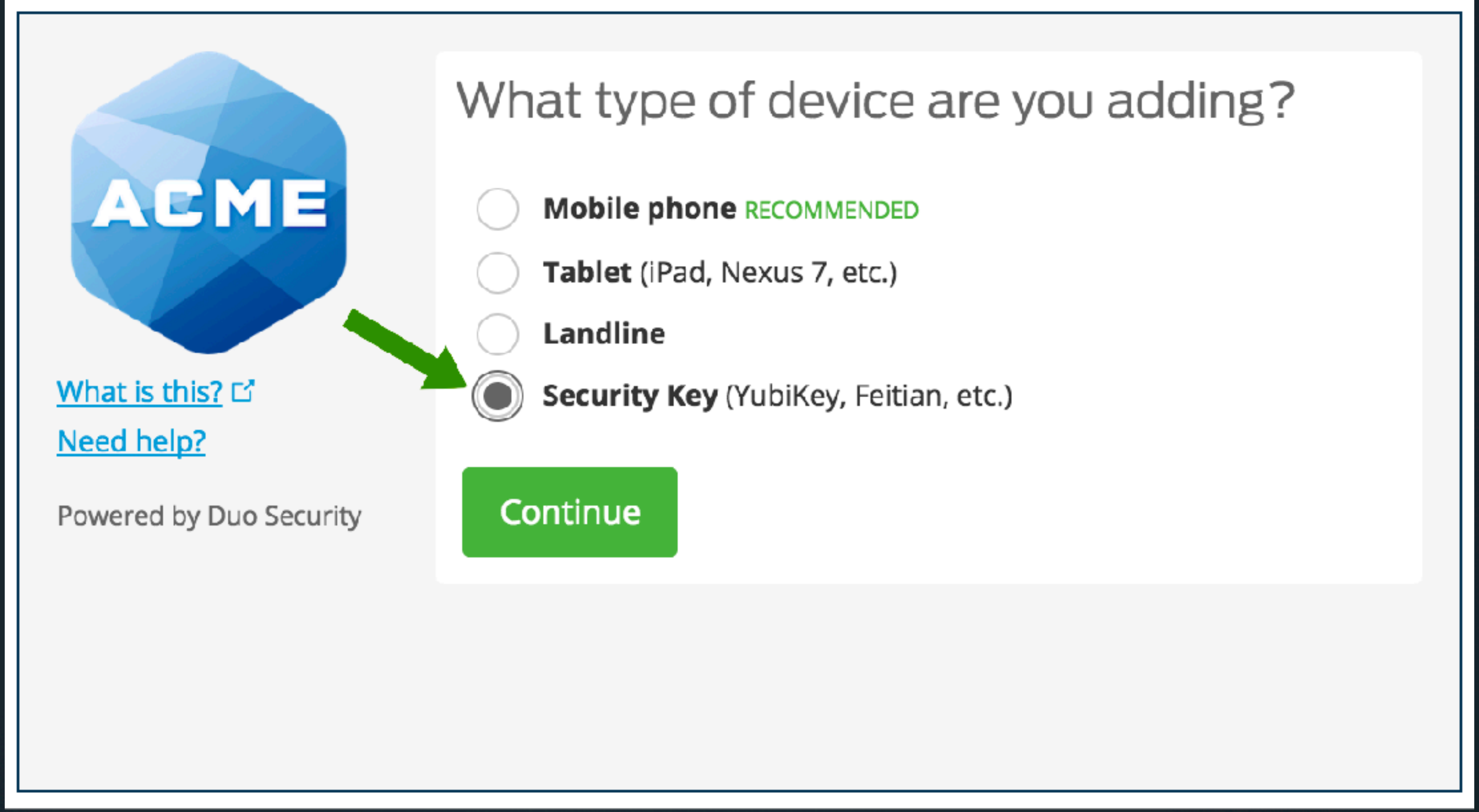
🟩 Implemented/Stable

🟨 In Development

🟧 Not Supported/No ETA

Thanks to Adam Powers

User Agent Implementation Differences



What type of device are you adding?

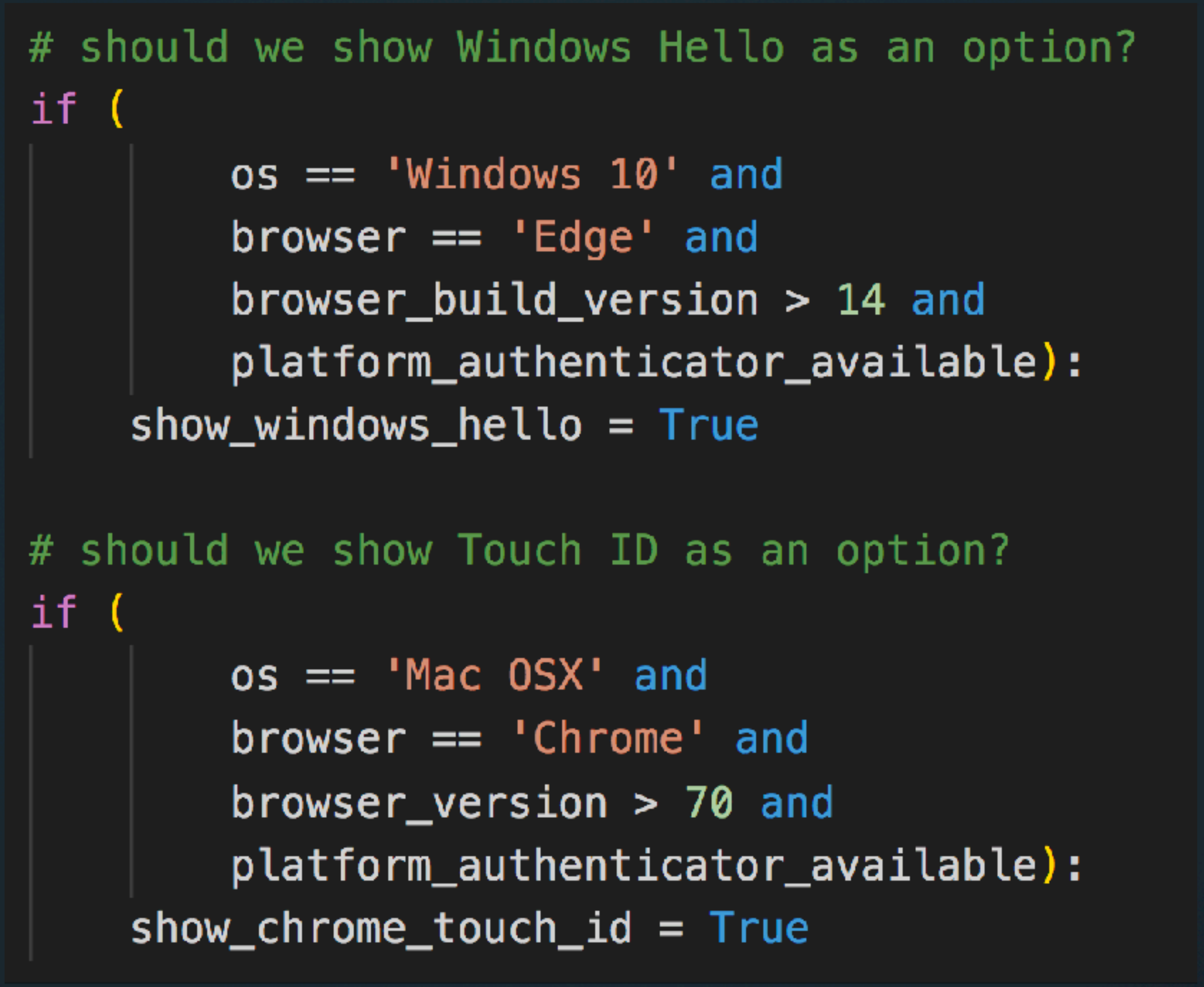
○ **Mobile phone** RECOMMENDED

○ **Tablet** (iPad, Nexus 7, etc.)

○ **Landline**

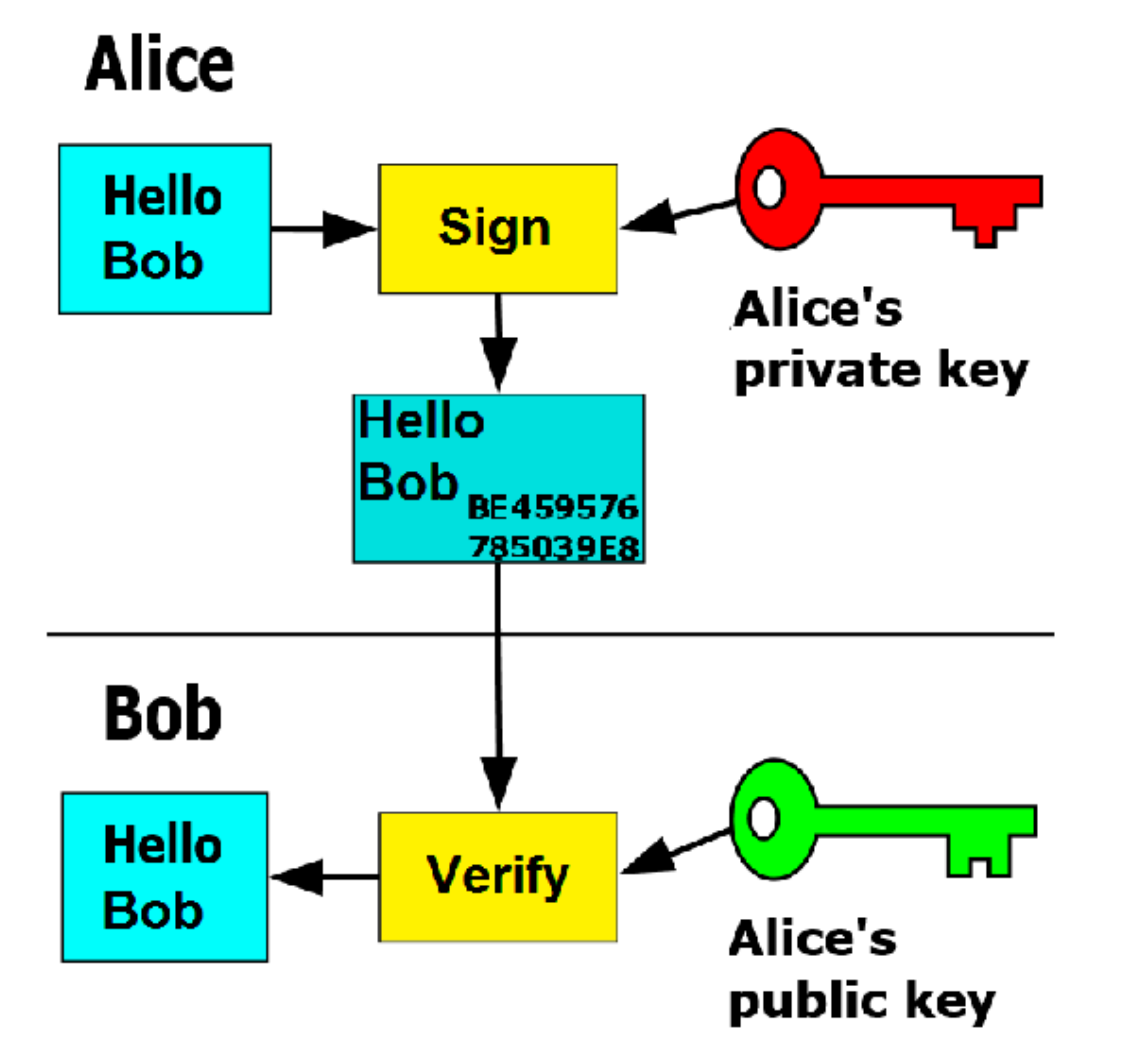
● **Security Key** (YubiKey, Feitian, etc.)

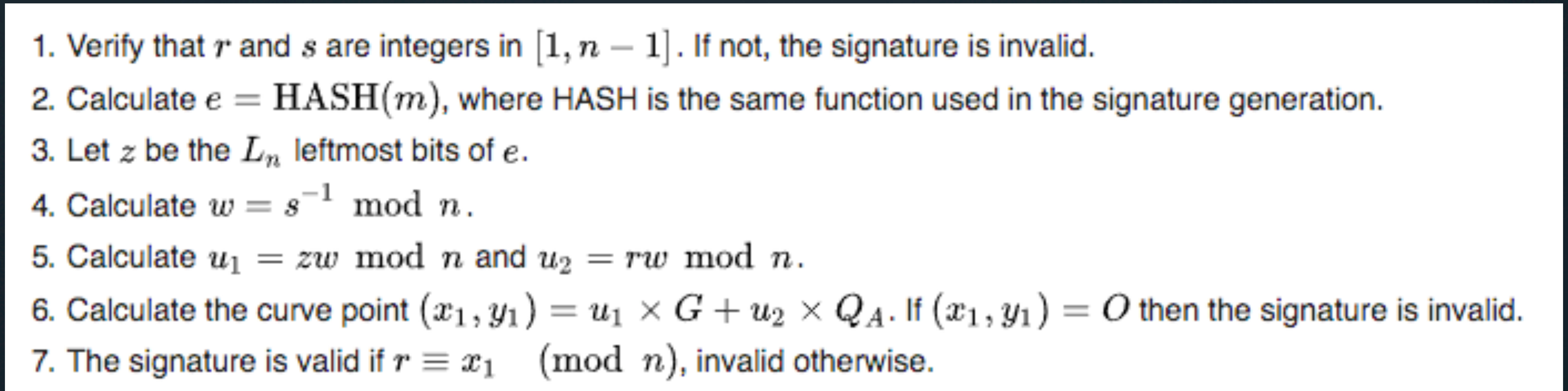
Building a decision engine to help guide users:



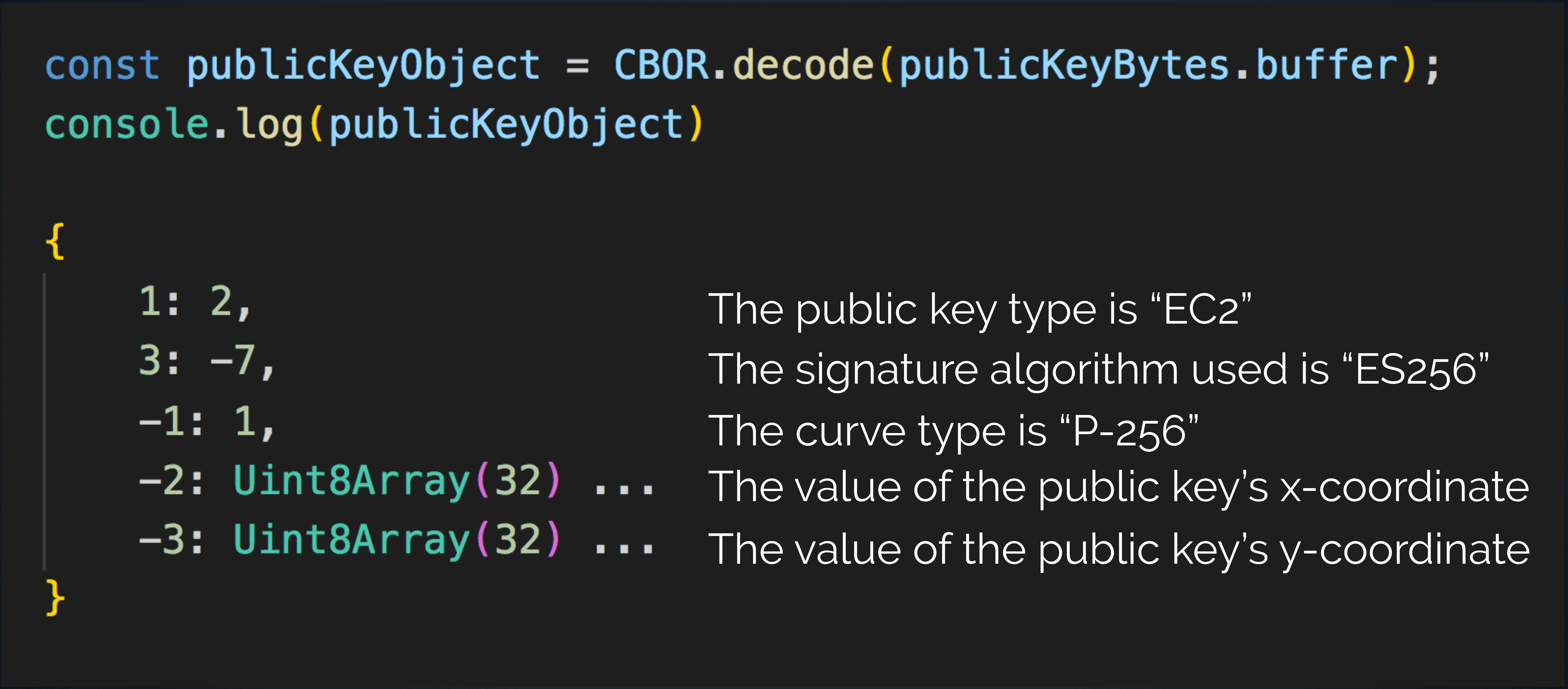
```
# should we show Windows Hello as an option?
if (
        os == 'Windows 10' and
        browser == 'Edge' and
        browser_build_version > 14 and
        platform_authenticator_available):
    show_windows_hello = True

# should we show Touch ID as an option?
if (
        os == 'Mac OSX' and
        browser == 'Chrome' and
        browser_version > 70 and
        platform_authenticator_available):
    show_chrome_touch_id = True
```

Foreseen and unforeseen challenges





1. Verify that r and s are integers in $[1, n-1]$. If not, the signature is invalid.
2. Calculate $e = \mathrm{HASH}(m)$, where HASH is the same function used in the signature generation.
3. Let z be the L_n leftmost bits of e.
4. Calculate $w = s^{-1} \mod n$.
5. Calculate $u_1 = zw \mod n$ and $u_2 = rw \mod n$.
6. Calculate the curve point $(x_1, y_1) = u_1 \times G + u_2 \times Q_A$. If $(x_1, y_1) = O$ then the signature is invalid.
7. The signature is valid if $r \equiv x_1 \pmod{n}$, invalid otherwise.



```
const publicKeyObject = CBOR.decode(publicKeyBytes.buffer);
console.log(publicKeyObject)

{
    1: 2,
    3: -7,
    -1: 1,
    -2: Uint8Array(32) ...
    -3: Uint8Array(32) ...
}
```

The public key type is “EC2”

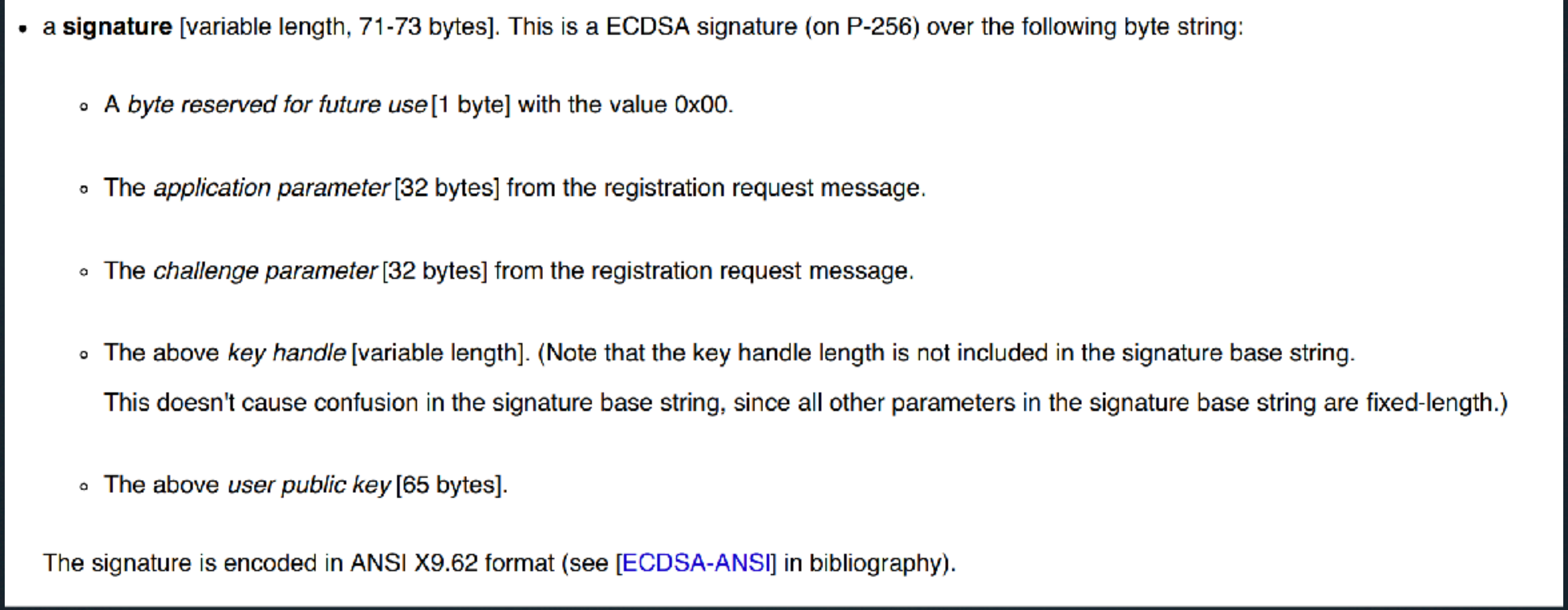
The signature algorithm used is “ES256”

The curve type is “P-256”

The value of the public key’s x-coordinate

The value of the public key’s y-coordinate

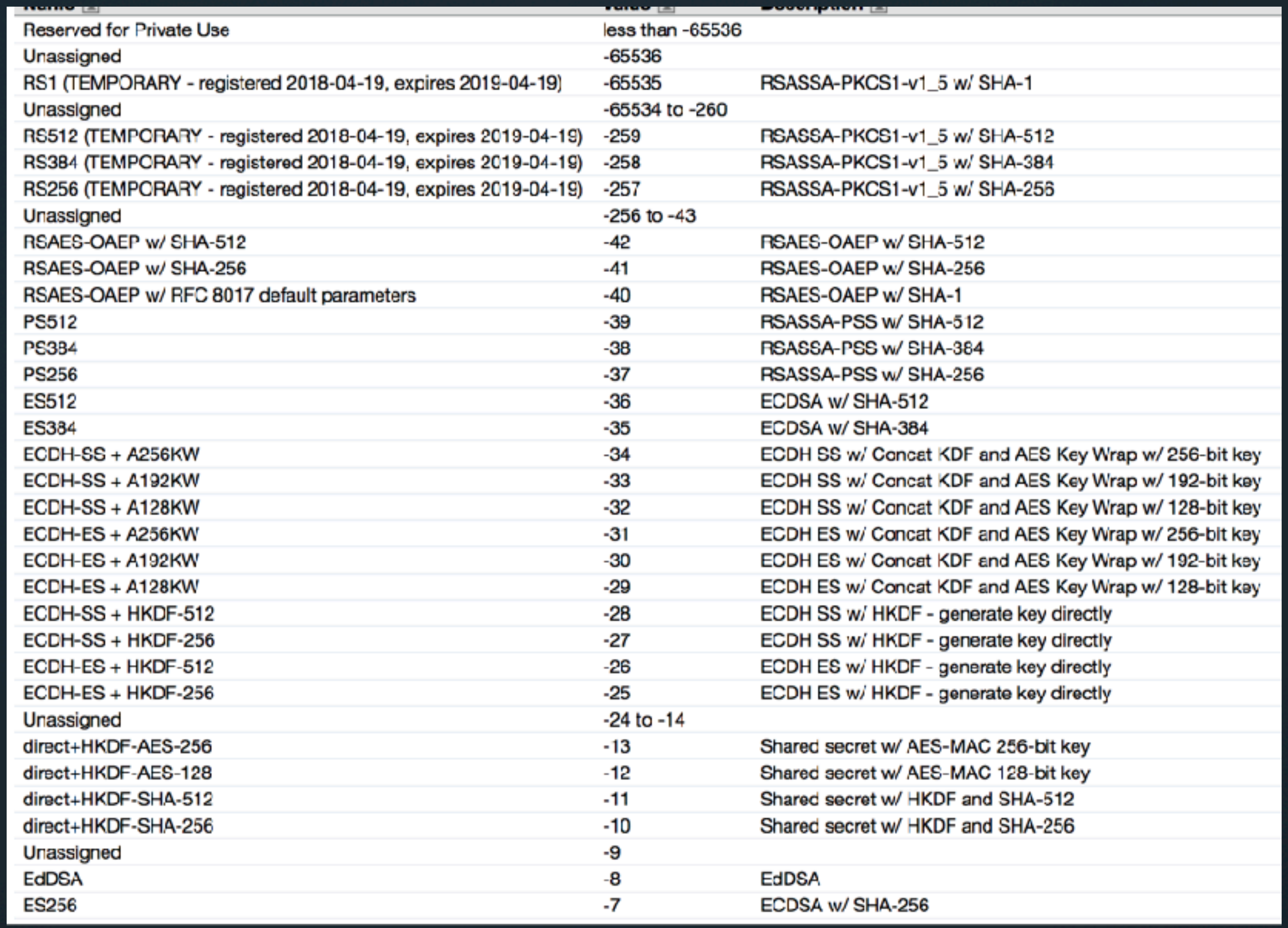
With U2F:
One Signature Algorithm



- a **signature** [variable length, 71-73 bytes]. This is a ECDSA signature (on P-256) over the following byte string:
    - A *byte reserved for future use* [1 byte] with the value 0x00.
    - The *application parameter* [32 bytes] from the registration request message.
    - The *challenge parameter* [32 bytes] from the registration request message.
    - The above *key handle* [variable length]. (Note that the key handle length is not included in the signature base string. This doesn't cause confusion in the signature base string, since all other parameters in the signature base string are fixed-length.)
    - The above *user public key* [65 bytes].

  The signature is encoded in ANSI X9.62 format (see [ECDSA-ANSI] in bibliography).

With WebAuthn:
Dozens of Signature Algorithms

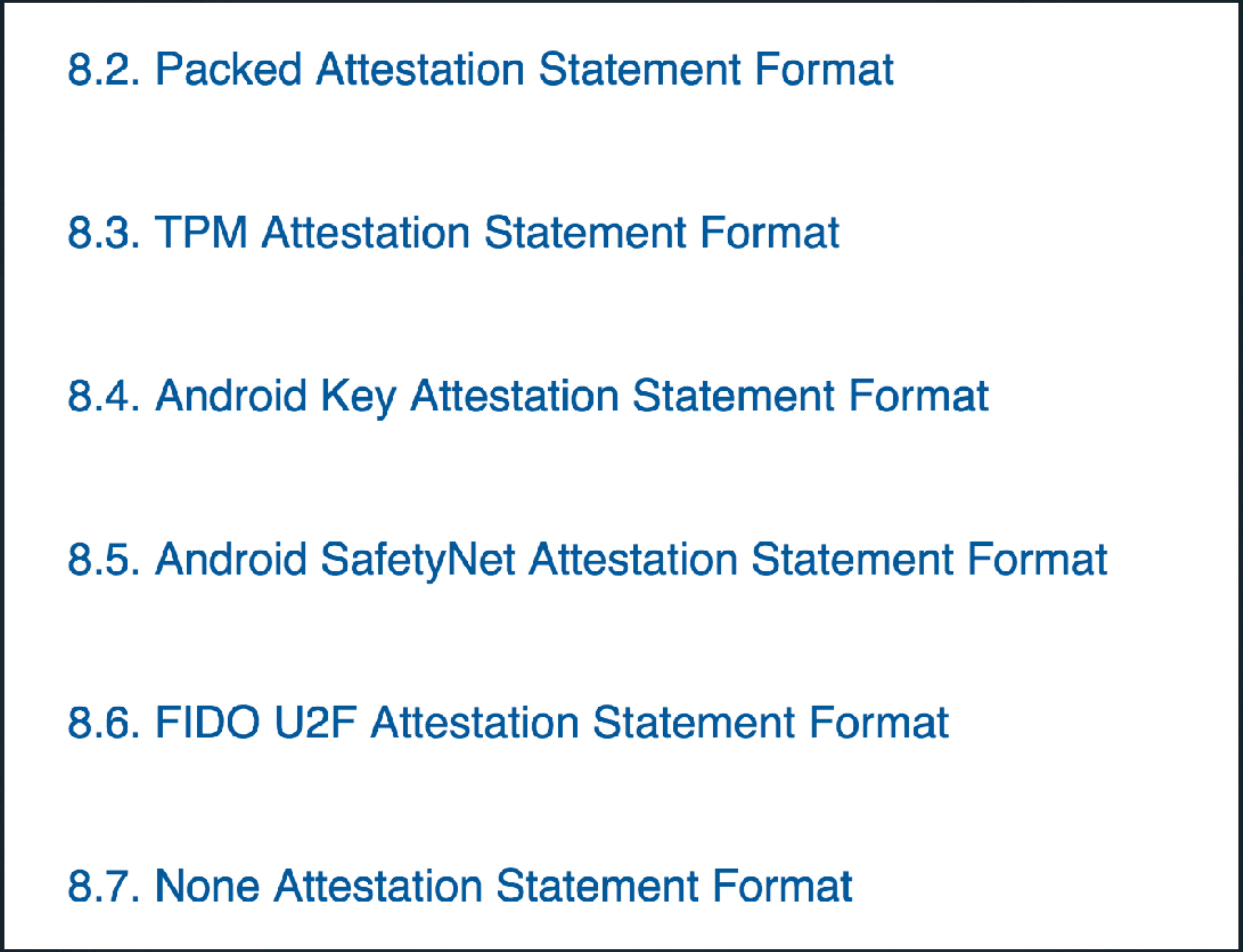


Name	Value	Description
Reserved for Private Use	less than -65536	
Unassigned	-65536	
RS1 (TEMPORARY - registered 2018-04-19, expires 2019-04-19)	-65535	RSASSA-PKCS1-v1_5 w/ SHA-1
Unassigned	-65534 to -260	
RS512 (TEMPORARY - registered 2018-04-19, expires 2019-04-19)	-259	RSASSA-PKCS1-v1_5 w/ SHA-512
RS384 (TEMPORARY - registered 2018-04-19, expires 2019-04-19)	-258	RSASSA-PKCS1-v1_5 w/ SHA-384
RS256 (TEMPORARY - registered 2018-04-19, expires 2019-04-19)	-257	RSASSA-PKCS1-v1_5 w/ SHA-256
Unassigned	-256 to -43	
RSAES-OAEP w/ SHA-512	-42	RSAES-OAEP w/ SHA-512
RSAES-OAEP w/ SHA-256	-41	RSAES-OAEP w/ SHA-256
RSAES-OAEP w/ RFC 8017 default parameters	-40	RSAES-OAEP w/ SHA-1
PS512	-39	RSASSA-PSS w/ SHA-512
PS384	-38	RSASSA-PSS w/ SHA-384
PS256	-37	RSASSA-PSS w/ SHA-256
ES512	-36	ECDSA w/ SHA-512
ES384	-35	ECDSA w/ SHA-384
ECDH-SS + A256KW	-34	ECDH SS w/ Concat KDF and AES Key Wrap w/ 256-bit key
ECDH-SS + A192KW	-33	ECDH SS w/ Concat KDF and AES Key Wrap w/ 192-bit key
ECDH-SS + A128KW	-32	ECDH SS w/ Concat KDF and AES Key Wrap w/ 128-bit key
ECDH-ES + A256KW	-31	ECDH ES w/ Concat KDF and AES Key Wrap w/ 256-bit key
ECDH-ES + A192KW	-30	ECDH ES w/ Concat KDF and AES Key Wrap w/ 192-bit key
ECDH-ES + A128KW	-29	ECDH ES w/ Concat KDF and AES Key Wrap w/ 128-bit key
ECDH-SS + HKDF-512	-28	ECDH SS w/ HKDF - generate key directly
ECDH-SS + HKDF-256	-27	ECDH SS w/ HKDF - generate key directly
ECDH-ES + HKDF-512	-26	ECDH ES w/ HKDF - generate key directly
ECDH-ES + HKDF-256	-25	ECDH ES w/ HKDF - generate key directly
Unassigned	-24 to -14	
direct+HKDF-AES-256	-13	Shared secret w/ AES-MAC 256-bit key
direct+HKDF-AES-128	-12	Shared secret w/ AES-MAC 128-bit key
direct+HKDF-SHA-512	-11	Shared secret w/ HKDF and SHA-512
direct+HKDF-SHA-256	-10	Shared secret w/ HKDF and SHA-256
Unassigned	-9	
EdDSA	-8	EdDSA
ES256	-7	ECDSA w/ SHA-256

Attestation

🏷️

Attestation is a way to cryptographically prove that a keypair came from secure hardware.



8.2. Packed Attestation Statement Format

8.3. TPM Attestation Statement Format

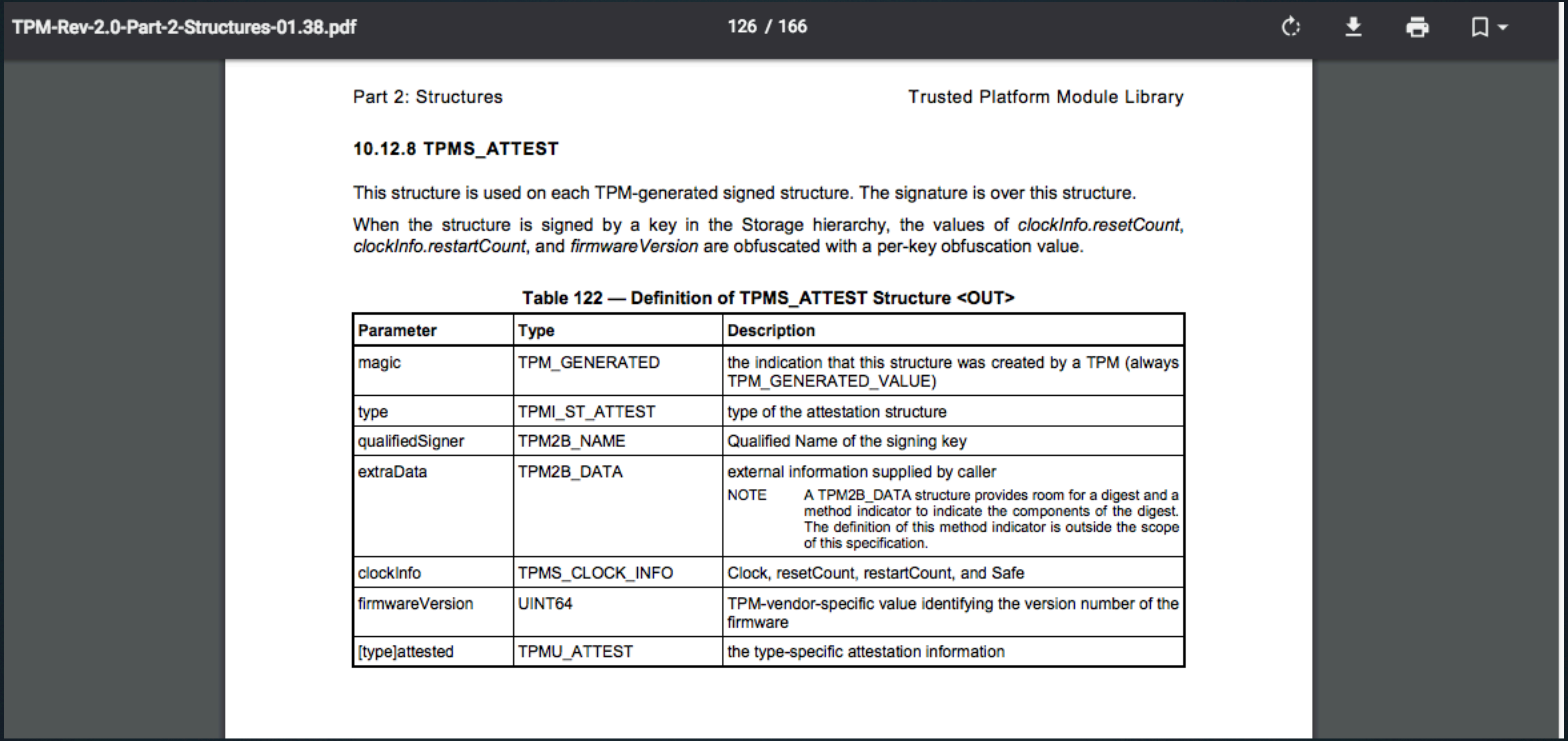
8.4. Android Key Attestation Statement Format

8.5. Android SafetyNet Attestation Statement Format

8.6. FIDO U2F Attestation Statement Format

8.7. None Attestation Statement Format

TPM Attestation



TPM-Rev-2.0-Part-2-Structures-01.38.pdf

126 / 166

Part 2: Structures   Trusted Platform Module Library

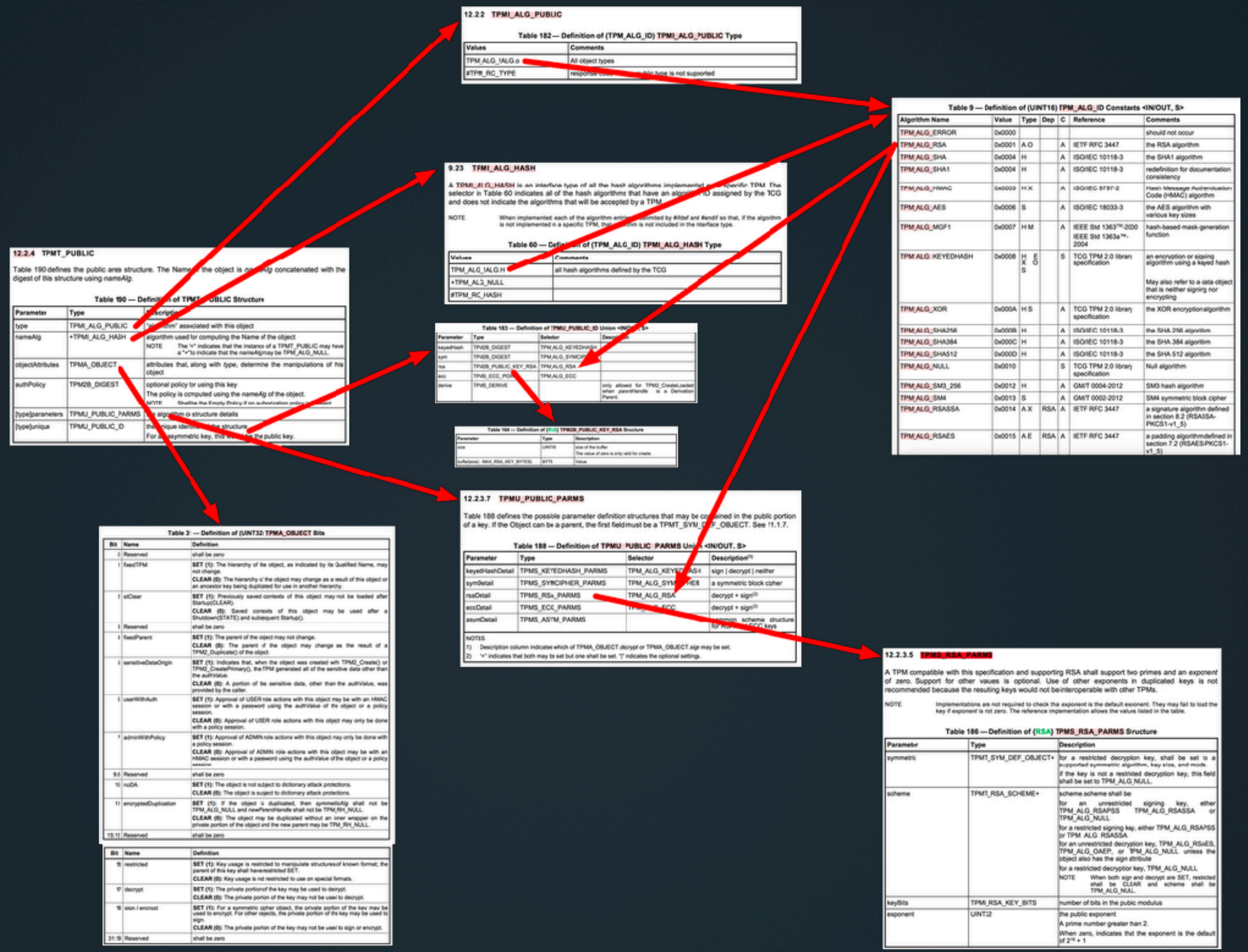
10.12.8 TPMS_ATTEST

This structure is used on each TPM-generated signed structure. The signature is over this structure.

When the structure is signed by a key in the Storage hierarchy, the values of *clockInfo.resetCount*, *clockInfo.restartCount*, and *firmwareVersion* are obfuscated with a per-key obfuscation value.

Table 122 — Definition of TPMS_ATTEST Structure <OUT>

Parameter	Type	Description
magic	TPM_GENERATED	the indication that this structure was created by a TPM (always TPM_GENERATED_VALUE)
type	TPMI_ST_ATTEST	type of the attestation structure
qualifiedSigner	TPM2B_NAME	Qualified Name of the signing key
extraData	TPM2B_DATA	external information supplied by caller NOTE A TPM2B_DATA structure provides room for a digest and a method indicator to indicate the components of the digest. The definition of this method indicator is outside the scope of this specification.
clockInfo	TPMS_CLOCK_INFO	Clock, resetCount, restartCount, and Safe
firmwareVersion	UINT64	TPM-vendor-specific value identifying the version number of the firmware
[type]attested	TPMU_ATTEST	the type-specific attestation information



12.2.4 TPMT_PUBLIC

Table 190 defines the public area structure. The Name of the object is nameAlg concatenated with the digest of this structure using nameAlg.

Table 190 — Definition of TPMT_PUBLIC Structure

Parameter	Type	Description
type	TPMI_ALG_PUBLIC	"algorithm" associated with this object
nameAlg	+TPMI_ALG_HASH	algorithm used for computing the Name of the object NOTE The "+" indicates that the instance of a TPMT_PUBLIC may have a "+" to indicate that the nameAlg may be TPM_ALG_NULL.
objectAttributes	TPMA_OBJECT	attributes that, along with type, determine the manipulations of this object
authPolicy	TPM2B_DIGEST	optional policy for using this key The policy is computed using the nameAlg of the object. NOTE Shall be the Empty Policy if no authorization policy is present.
[type]parameters	TPMU_PUBLIC_PARMS	the algorithm or structure details
[type]unique	TPMU_PUBLIC_ID	the unique identifier of the structure. For an asymmetric key, this would be the public key.

12.2.2 TPMI_ALG_PUBLIC

Table 182 — Definition of (TPM_ALG_ID) TPMI_ALG_?UBLIC Type

Values	Comments
TPM_ALG_!ALG.o	All object types
#TPM_RC_TYPE	response code when a public type is not supported

9.23 TPMI_ALG_HASH

A TPMI_ALG_HASH is an interface type of all the hash algorithms implemented on a specific TPM. The selector in Table 60 indicates all of the hash algorithms that have an algorithm ID assigned by the TCG and does not indicate the algorithms that will be accepted by a TPM.

NOTE When implemented each of the algorithm entries is delimited by #ifdef and #endif so that, if the algorithm is not implemented in a specific TPM, that algorithm is not included in the interface type.

Table 60 — Definition of (TPM_ALG_ID) TPMI_ALG_HASH Type

Values	Comments
TPM_ALG_!ALG.H	all hash algorithms defined by the TCG
+TPM_AL3_NULL	
#TPM_RC_HASH	

Table 183 — Definition of TPMU_PUBLIC_ID Union <IN/OUT, S>

Parameter	Type	Selector	Description
keyedHash	TPM2B_DIGEST	TPM_ALG_KEYEDHASH	
sym	TPM2B_DIGEST	TPM_ALG_SYMCIPHER	
rsa	TPM2B_PUBLIC_KEY_RSA	TPM_ALG_RSA	
ecc	TPMS_ECC_POINT	TPM_ALG_ECC	
derive	TPMS_DERIVE		only allowed for TPM2_CreateLoaded when parentHandle is a Derivation Parent.

Table 184 — Definition of (RSA) TPM2B_PUBLIC_KEY_RSA Structure

Parameter	Type	Description
size	UINT16	size of the buffer The value of zero is only valid for create.
buffer[size] {: MAX_RSA_KEY_BYTES}	BYTE	Value

12.2.3.7 TPMU_PUBLIC_PARMS

Table 188 defines the possible parameter definition structures that may be contained in the public portion of a key. If the Object can be a parent, the first field must be a TPMT_SYM_DEF_OBJECT. See 11.1.7.

Table 188 — Definition of TPMU_PUBLIC_PARMS Union <IN/OUT, S>

Parameter	Type	Selector	Description[1]
keyedHashDetail	TPMS_KEYEDHASH_PARMS	TPM_ALG_KEYEDHASH	sign | decrypt | neither
symDetail	TPMS_SYMCIPHER_PARMS	TPM_ALG_SYMCIPHER	a symmetric block cipher
rsaDetail	TPMS_RSa_PARMS	TPM_ALG_RSA	decrypt + sign[2]
eccDetail	TPMS_ECC_PARMS	TPM_ALG_ECC	decrypt + sign[2]
asymDetail	TPMS_ASYM_PARMS		common scheme structure for RSA and ECC keys

NOTES

1) Description column indicates which of TPMA_OBJECT.decrypt or TPMA_OBJECT.sign may be set.

2) "+" indicates that both may be set but one shall be set. "|" indicates the optional setting.

Table 3' — Definition of (UINT32) TPMA_OBJECT Bits

Bit	Name	Definition
0	Reserved	shall be zero
1	fixedTPM	**SET (1):** The hierarchy of the object, as indicated by its Qualified Name, may not change. **CLEAR (0):** The hierarchy of the object may change as a result of this object or an ancestor key being duplicated for use in another hierarchy.
2	stClear	**SET (1):** Previously saved contexts of this object may not be loaded after Startup(CLEAR). **CLEAR (0):** Saved contexts of this object may be used after a Shutdown(STATE) and subsequent Startup().
3	Reserved	shall be zero
4	fixedParent	**SET (1):** The parent of the object may not change. **CLEAR (0):** The parent of the object may change as the result of a TPM2_Duplicate() of the object.
5	sensitiveDataOrigin	**SET (1):** Indicates that, when the object was created with TPM2_Create() or TPM2_CreatePrimary(), the TPM generated all of the sensitive data other than the authValue. **CLEAR (0):** A portion of the sensitive data, other than the authValue, was provided by the caller.
6	userWithAuth	**SET (1):** Approval of USER role actions with this object may be with an HMAC session or with a password using the authValue of the object or a policy session. **CLEAR (0):** Approval of USER role actions with this object may only be done with a policy session.
7	adminWithPolicy	**SET (1):** Approval of ADMIN role actions with this object may only be done with a policy session. **CLEAR (0):** Approval of ADMIN role actions with this object may be with an HMAC session or with a password using the authValue of the object or a policy session.
9:8	Reserved	shall be zero
10	noDA	**SET (1):** The object is not subject to dictionary attack protections. **CLEAR (0):** The object is subject to dictionary attack protections.
11	encryptedDuplication	**SET (1):** If the object is duplicated, then symmetricAlg shall not be TPM_ALG_NULL and newParentHandle shall not be TPM_RH_NULL. **CLEAR (0):** The object may be duplicated without an inner wrapper on the private portion of the object and the new parent may be TPM_RH_NULL.
15:12	Reserved	shall be zero
16	restricted	**SET (1):** Key usage is restricted to manipulate structures of known format; the parent of this key shall have restricted SET. **CLEAR (0):** Key usage is not restricted to use on special formats.
17	decrypt	**SET (1):** The private portion of the key may be used to decrypt. **CLEAR (0):** The private portion of the key may not be used to decrypt.
18	sign / encrypt	**SET (1):** For a symmetric cipher object, the private portion of the key may be used to encrypt. For other objects, the private portion of the key may be used to sign. **CLEAR (0):** The private portion of the key may not be used to sign or encrypt.
31:19	Reserved	shall be zero

Table 9 — Definition of (UINT16) TPM_ALG_ID Constants <IN/OUT, S>

Algorithm Name	Value	Type	Dep	C	Reference	Comments
TPM_ALG_ERROR	0x0000					should not occur
TPM_ALG_RSA	0x0001	A O		A	IETF RFC 3447	the RSA algorithm
TPM_ALG_SHA	0x0004	H		A	ISO/IEC 10118-3	the SHA1 algorithm
TPM_ALG_SHA1	0x0004	H		A	ISO/IEC 10118-3	redefinition for documentation consistency
TPM_ALG_HMAC	0x0005	H X		A	ISO/IEC 9797-2	Hash Message Authentication Code (HMAC) algorithm
TPM_ALG_AES	0x0006	S		A	ISO/IEC 18033-3	the AES algorithm with various key sizes
TPM_ALG_MGF1	0x0007	H M		A	IEEE Std 1363™-2000 IEEE Std 1363a™-2004	hash-based mask-generation function
TPM_ALG_KEYEDHASH	0x0008	H X S E O		S	TCG TPM 2.0 library specification	an encryption or signing algorithm using a keyed hash May also refer to a data object that is neither signing nor encrypting
TPM_ALG_XOR	0x000A	H S		A	TCG TPM 2.0 library specification	the XOR encryption algorithm
TPM_ALG_SHA256	0x000B	H		A	ISO/IEC 10118-3	the SHA 256 algorithm
TPM_ALG_SHA384	0x000C	H		A	ISO/IEC 10118-3	the SHA 384 algorithm
TPM_ALG_SHA512	0x000D	H		A	ISO/IEC 10118-3	the SHA 512 algorithm
TPM_ALG_NULL	0x0010			S	TCG TPM 2.0 library specification	Null algorithm
TPM_ALG_SM3_256	0x0012	H		A	GM/T 0004-2012	SM3 hash algorithm
TPM_ALG_SM4	0x0013	S		A	GM/T 0002-2012	SM4 symmetric block cipher
TPM_ALG_RSASSA	0x0014	A X	RSA	A	IETF RFC 3447	a signature algorithm defined in section 8.2 (RSASSA-PKCS1-v1_5)
TPM_ALG_RSAES	0x0015	A E	RSA	A	IETF RFC 3447	a padding algorithm defined in section 7.2 (RSAES-PKCS1-v1_5)

12.2.3.5 TPMS_RSA_PARMS

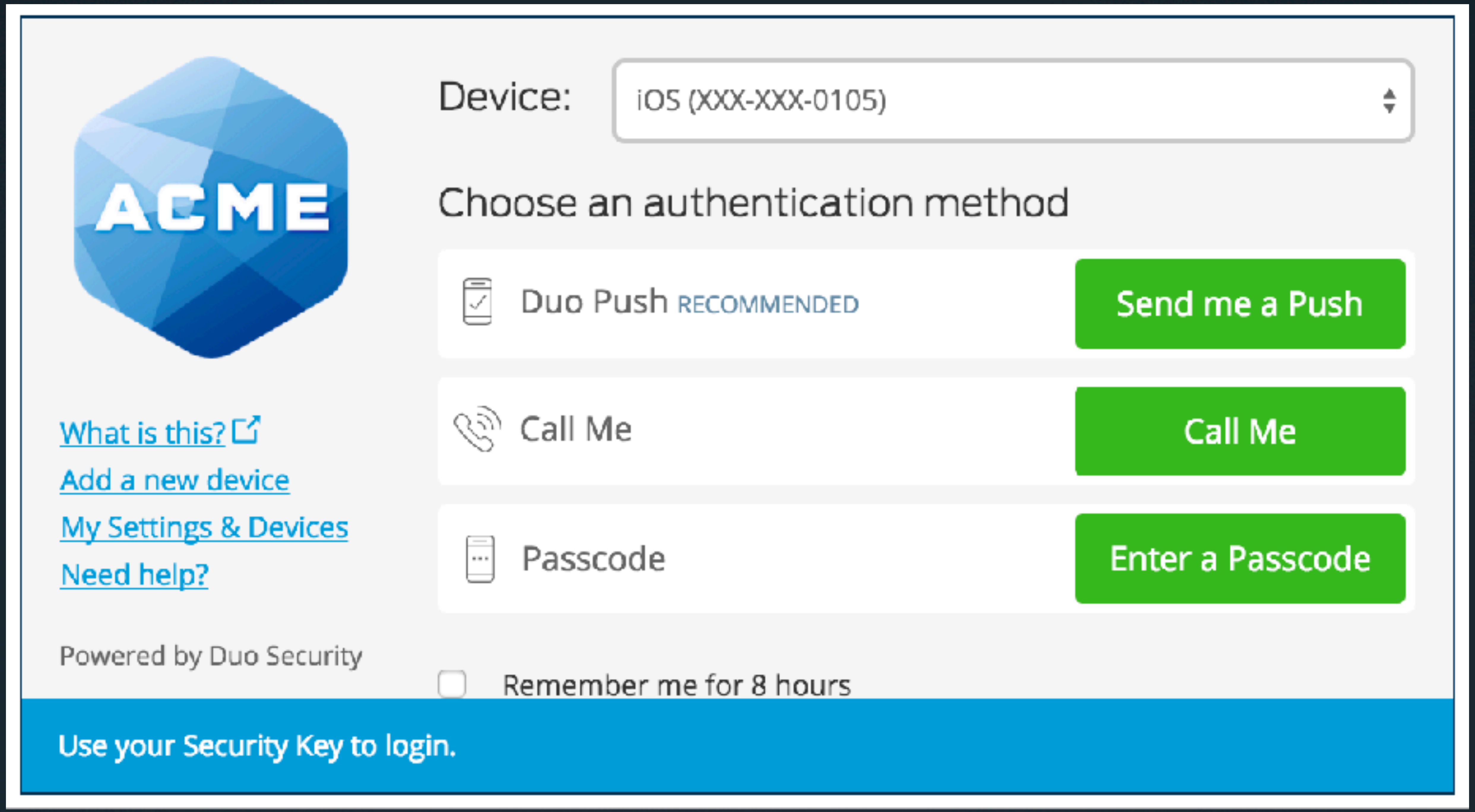
A TPM compatible with this specification and supporting RSA shall support two primes and an exponent of zero. Support for other values is optional. Use of other exponents in duplicated keys is not recommended because the resulting keys would not be interoperable with other TPMs.

NOTE Implementations are not required to check that exponent is the default exponent. They may fail to load the key if exponent is not zero. The reference implementation allows the values listed in the table.

Table 186 — Definition of (RSA) TPMS_RSA_PARMS Structure

Parameter	Type	Description
symmetric	TPMT_SYM_DEF_OBJECT+	for a restricted decryption key, shall be set to a supported symmetric algorithm, key size, and mode. if the key is not a restricted decryption key, this field shall be set to TPM_ALG_NULL.
scheme	TPMT_RSA_SCHEME+	scheme.scheme shall be for an unrestricted signing key, either TPM_ALG_RSAPSS TPM_ALG_RSASSA or TPM_ALG_NULL for a restricted signing key, either TPM_ALG_RSAPSS or TPM_ALG_RSASSA for an unrestricted decryption key, TPM_ALG_RSAES, TPM_ALG_OAEP, or TPM_ALG_NULL unless the object also has the sign attribute for a restricted decryption key, TPM_ALG_NULL NOTE When both sign and decrypt are SET, restricted shall be CLEAR and scheme shall be TPM_ALG_NULL.
keyBits	TPMI_RSA_KEY_BITS	number of bits in the public modulus
exponent	UINT32	the public exponent A prime number greater than 2. When zero, indicates that the exponent is the default of $2^{16} + 1$

Thanks to Adam Powers and Yuriy Ackermann

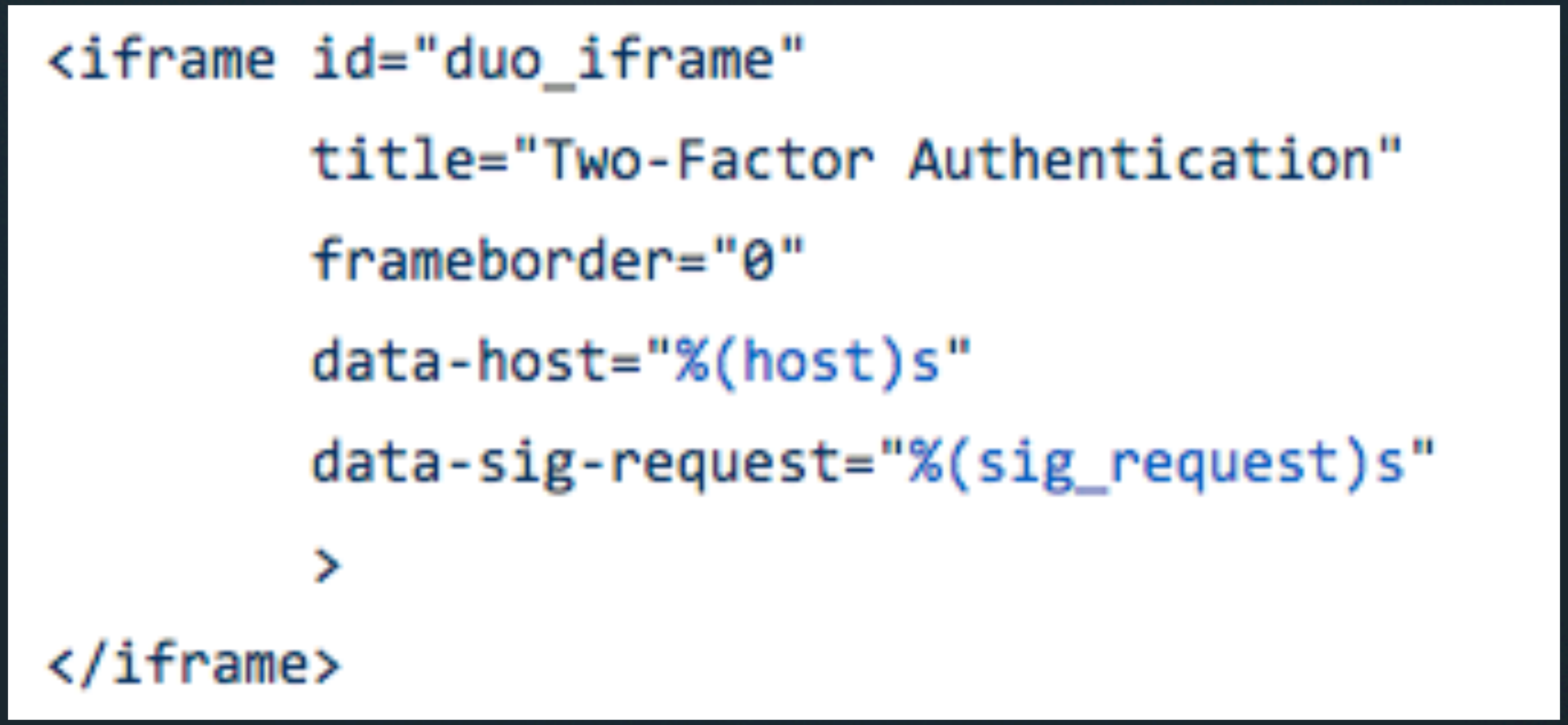


Device: iOS (XXX-XXX-0105)

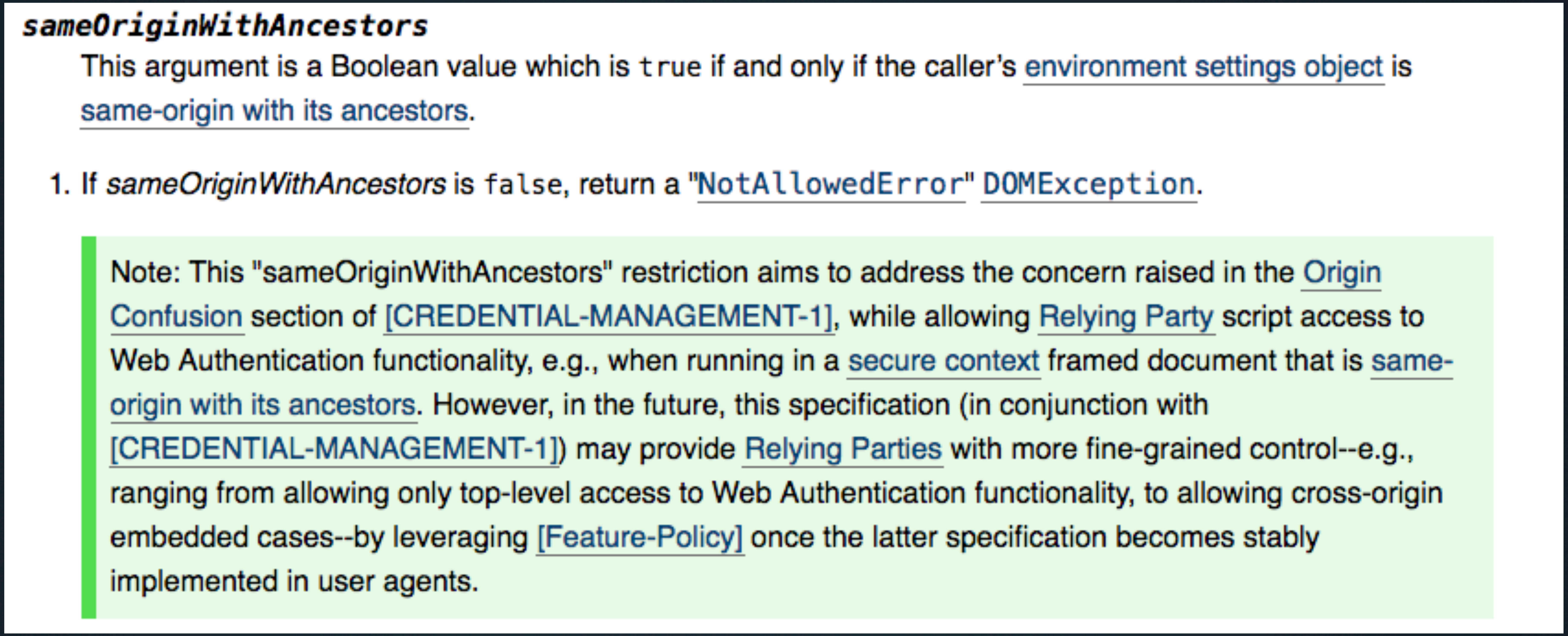
Choose an authentication method

Remember me for 8 hours

Use your Security Key to login.



```
<iframe id="duo_iframe"
        title="Two-Factor Authentication"
        frameborder="0"
        data-host="%(host)s"
        data-sig-request="%(sig_request)s"
        >
</iframe>
```

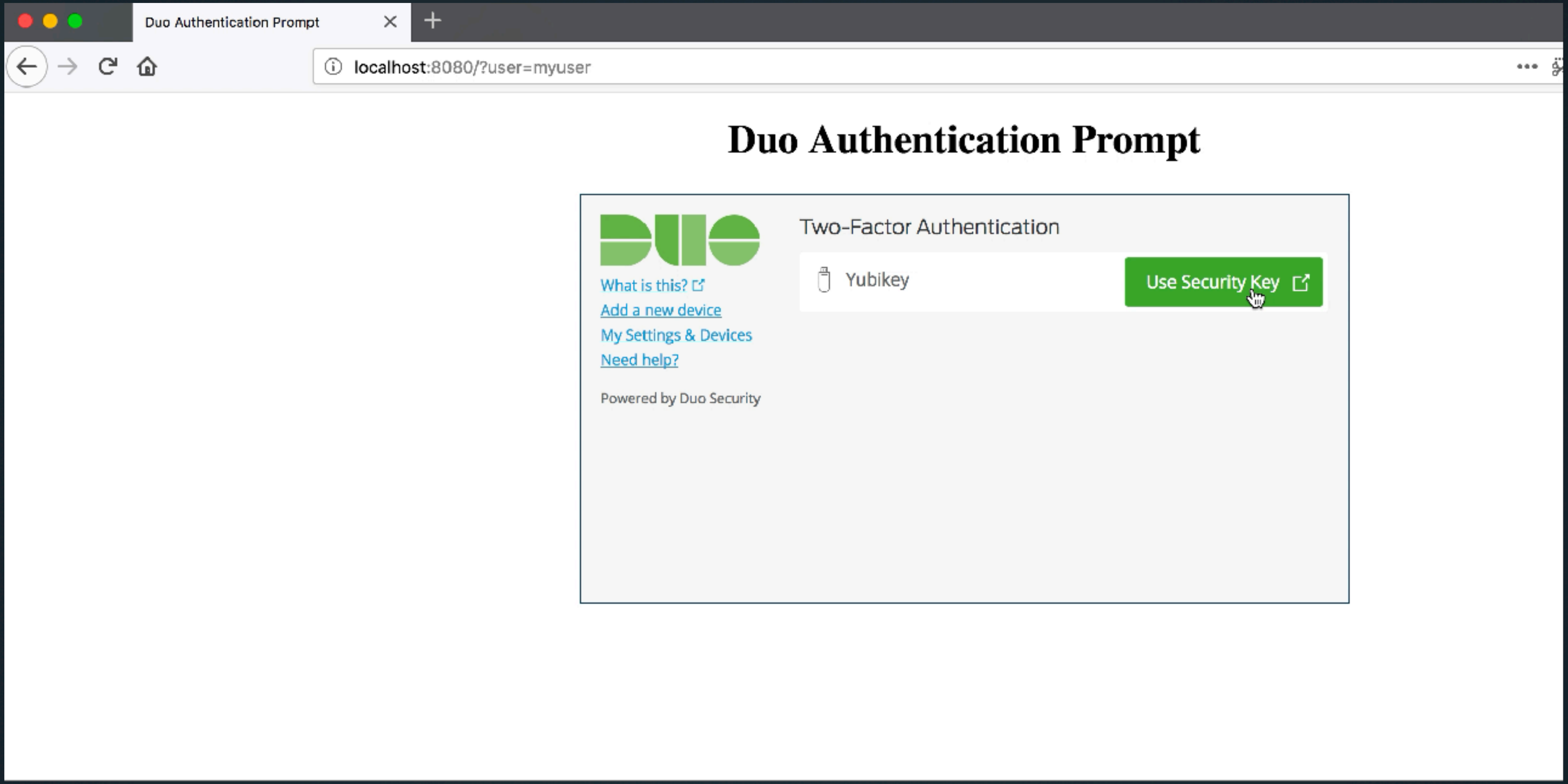


sameOriginWithAncestors

This argument is a Boolean value which is `true` if and only if the caller's environment settings object is same-origin with its ancestors.

1. If *sameOriginWithAncestors* is `false`, return a "`NotAllowedError`" `DOMException`.

> Note: This "sameOriginWithAncestors" restriction aims to address the concern raised in the Origin Confusion section of [CREDENTIAL-MANAGEMENT-1], while allowing Relying Party script access to Web Authentication functionality, e.g., when running in a secure context framed document that is same-origin with its ancestors. However, in the future, this specification (in conjunction with [CREDENTIAL-MANAGEMENT-1]) may provide Relying Parties with more fine-grained control--e.g., ranging from allowing only top-level access to Web Authentication functionality, to allowing cross-origin embedded cases--by leveraging [Feature-Policy] once the latter specification becomes stably implemented in user agents.

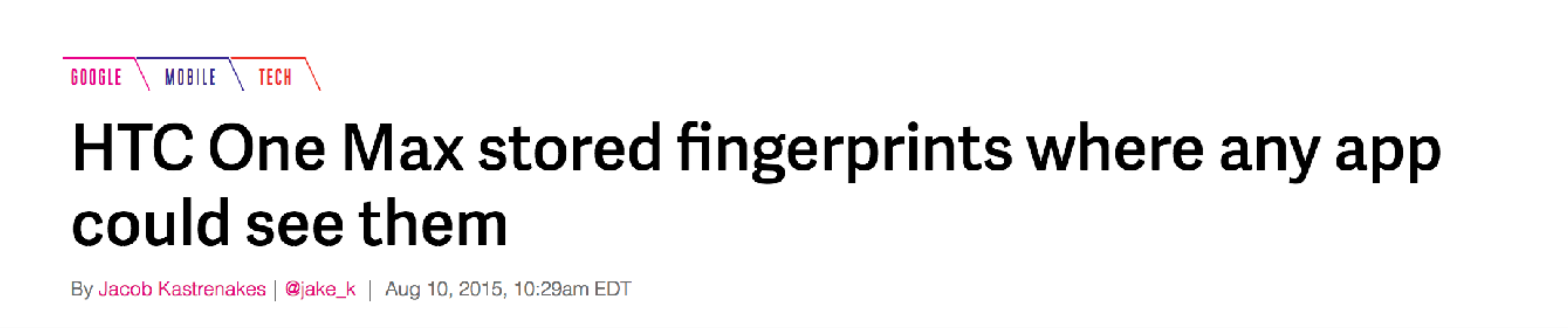


Duo Authentication Prompt

localhost:8080/?user=myuser

Duo Authentication Prompt

Two-Factor Authentication

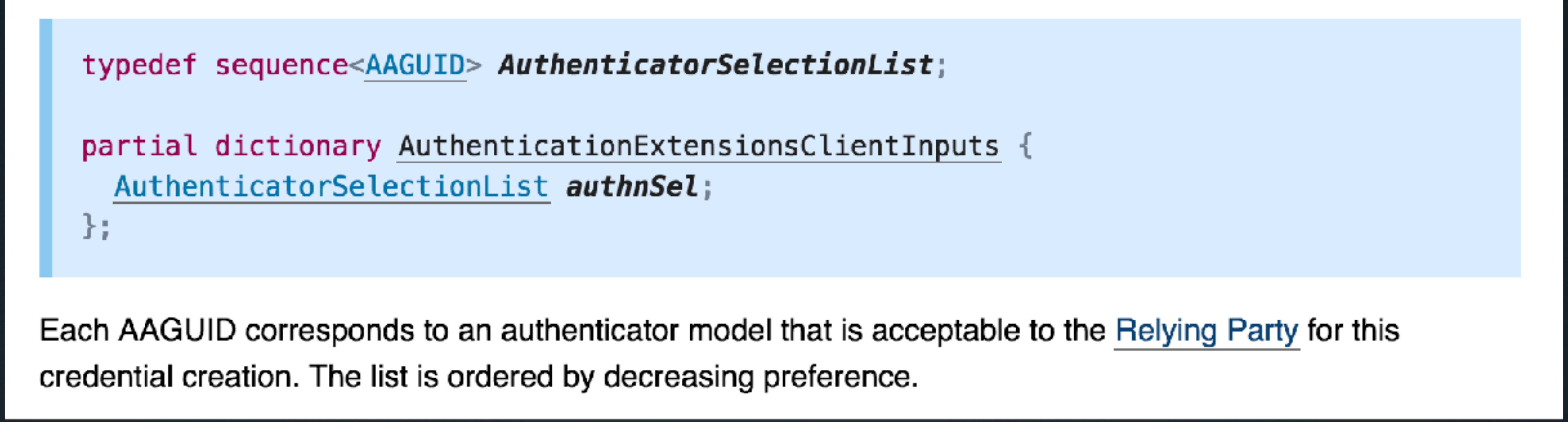


GOOGLE MOBILE TECH

HTC One Max stored fingerprints where any app could see them

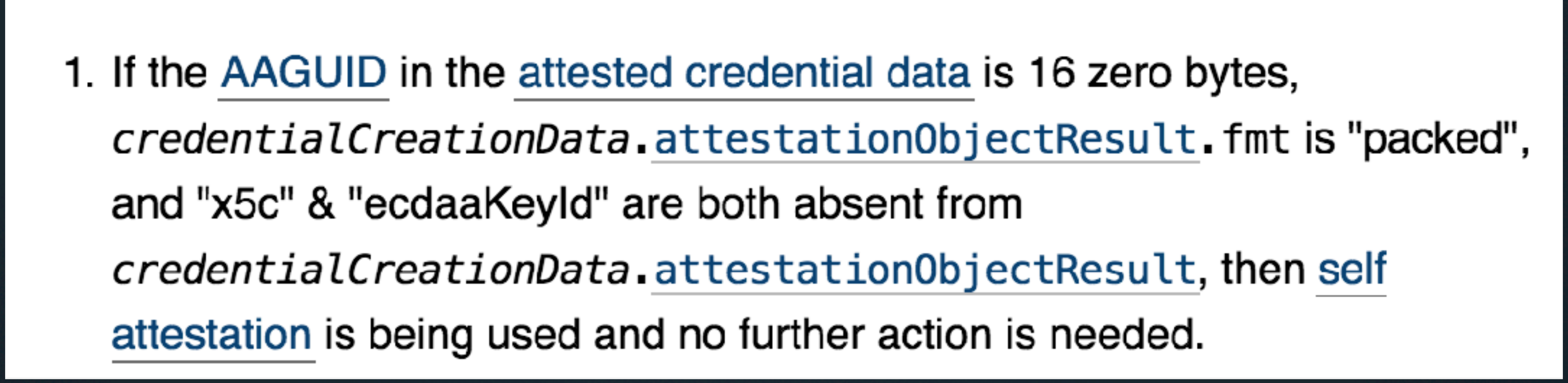
By Jacob Kastrenakes | @jake_k | Aug 10, 2015, 10:29am EDT

Whitelisting Authenticators



```
typedef sequence<AAGUID> AuthenticatorSelectionList;

partial dictionary AuthenticationExtensionsClientInputs {
  AuthenticatorSelectionList authnSel;
};
```

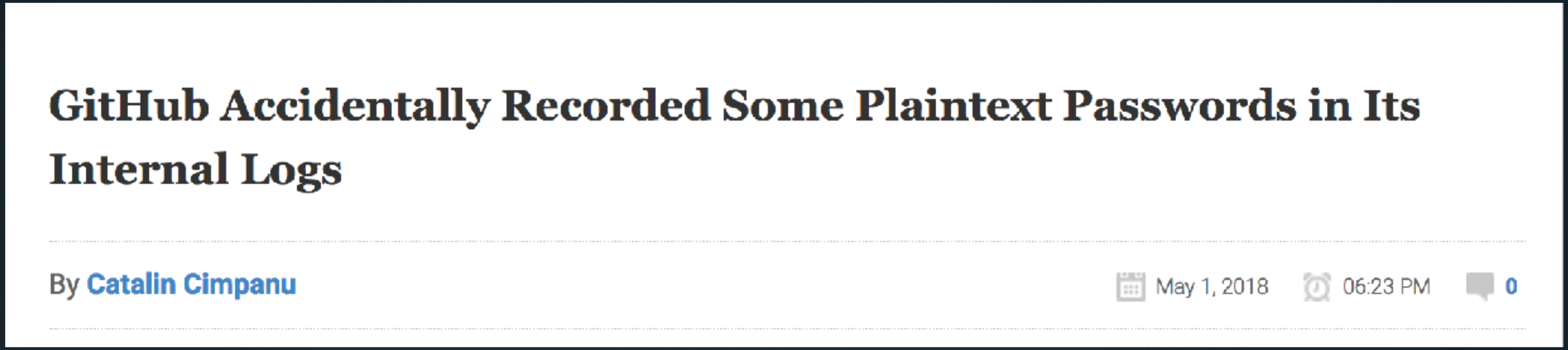
Each AAGUID corresponds to an authenticator model that is acceptable to the Relying Party for this credential creation. The list is ordered by decreasing preference.



1. If the AAGUID in the attested credential data is 16 zero bytes, *credentialCreationData*.`attestationObjectResult`.`fmt` is "packed", and "x5c" & "ecdaaKeyId" are both absent from *credentialCreationData*.`attestationObjectResult`, then self attestation is being used and no further action is needed.

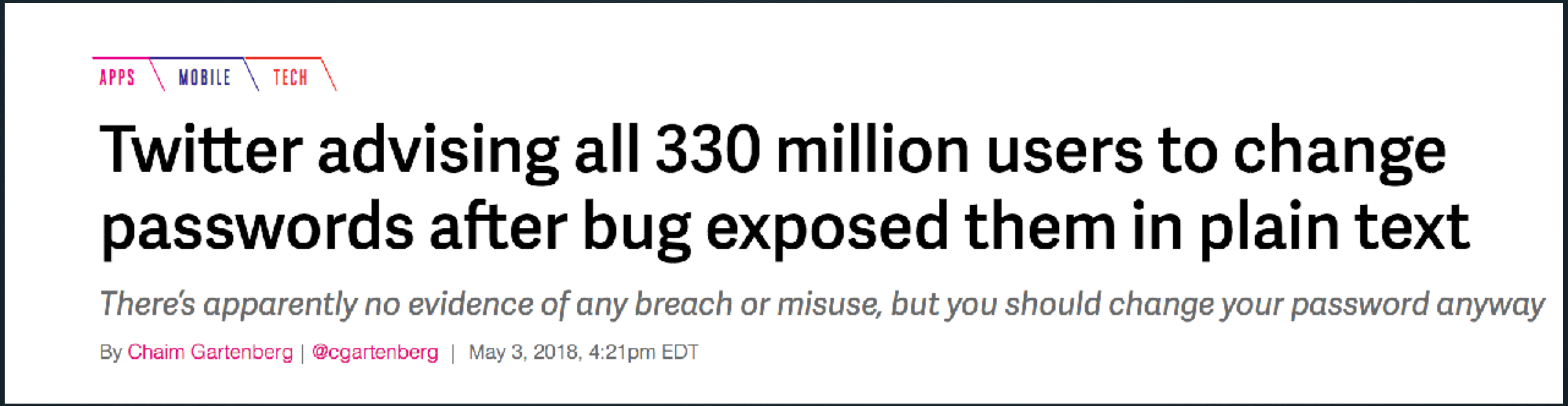
Rolling out to users

Log Everything



GitHub Accidentally Recorded Some Plaintext Passwords in Its Internal Logs

By Catalin Cimpanu May 1, 2018 06:23 PM 0



APPS MOBILE TECH

Twitter advising all 330 million users to change passwords after bug exposed them in plain text

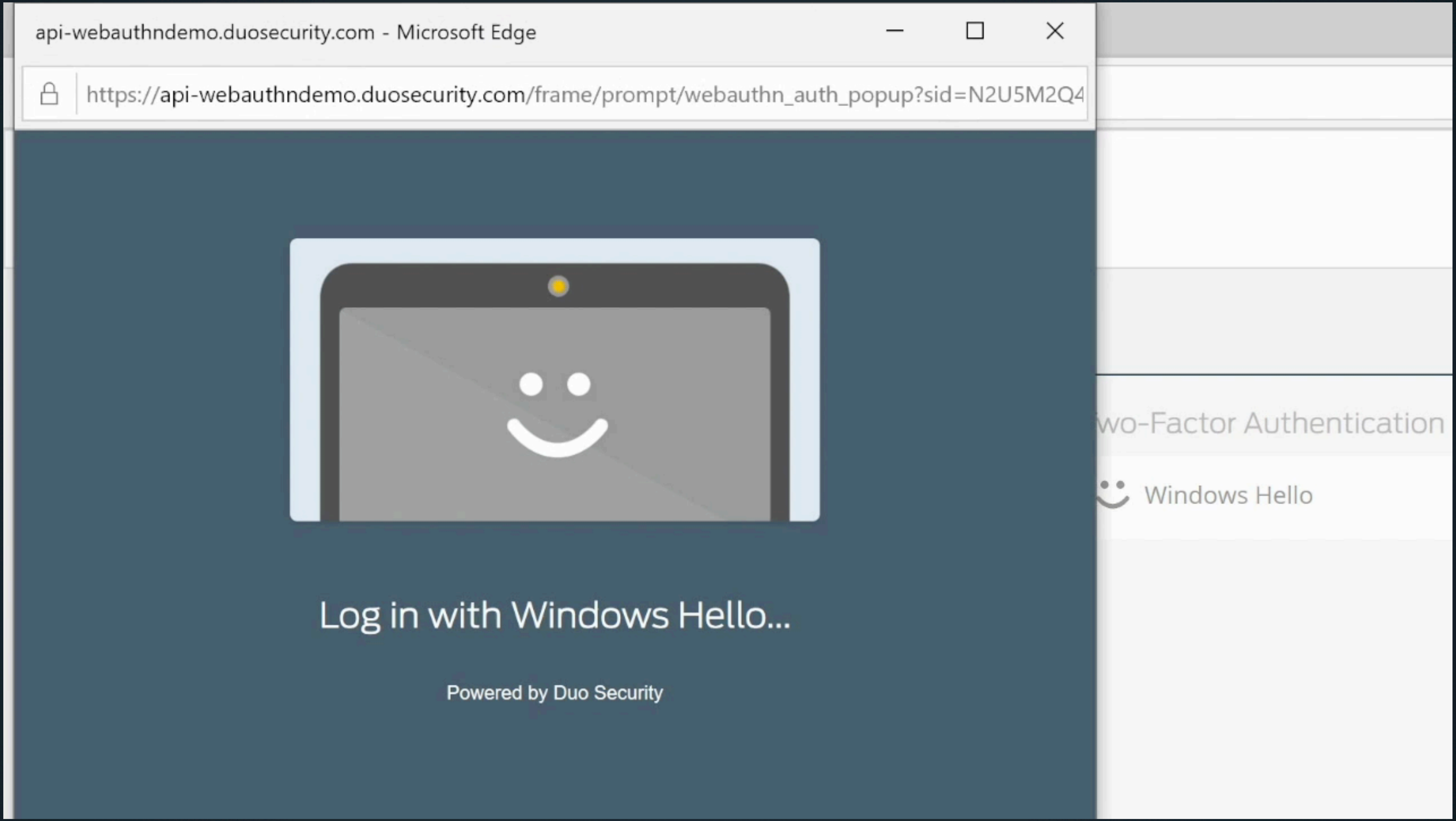
There's apparently no evidence of any breach or misuse, but you should change your password anyway

By Chaim Gartenberg | @cgartenberg | May 3, 2018, 4:21pm EDT

Incrementally add support for:

Browsers
Attestation types
Signature algorithms
Cross-platform vs platform authenticators

Looking Ahead



api-webauthndemo.duosecurity.com - Microsoft Edge

https://api-webauthndemo.duosecurity.com/frame/prompt/webauthn_auth_popup?sid=N2U5M2Q4

Log in with Windows Hello...

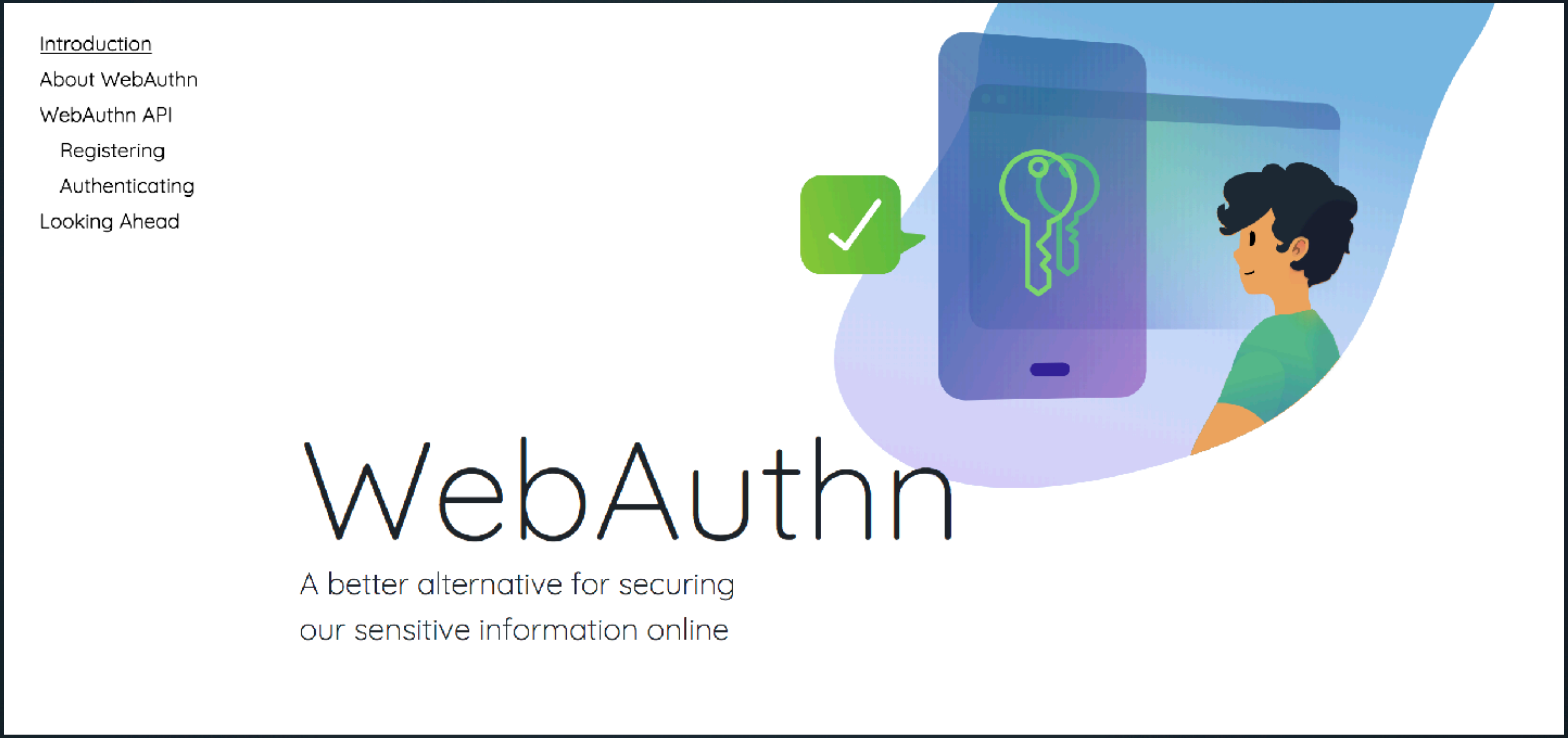
Powered by Duo Security

wo-Factor Authentication

Windows Hello

@subyraman

https://webauthn.guide
coming soon!



Introduction

About WebAuthn

WebAuthn API

- Registering
- Authenticating

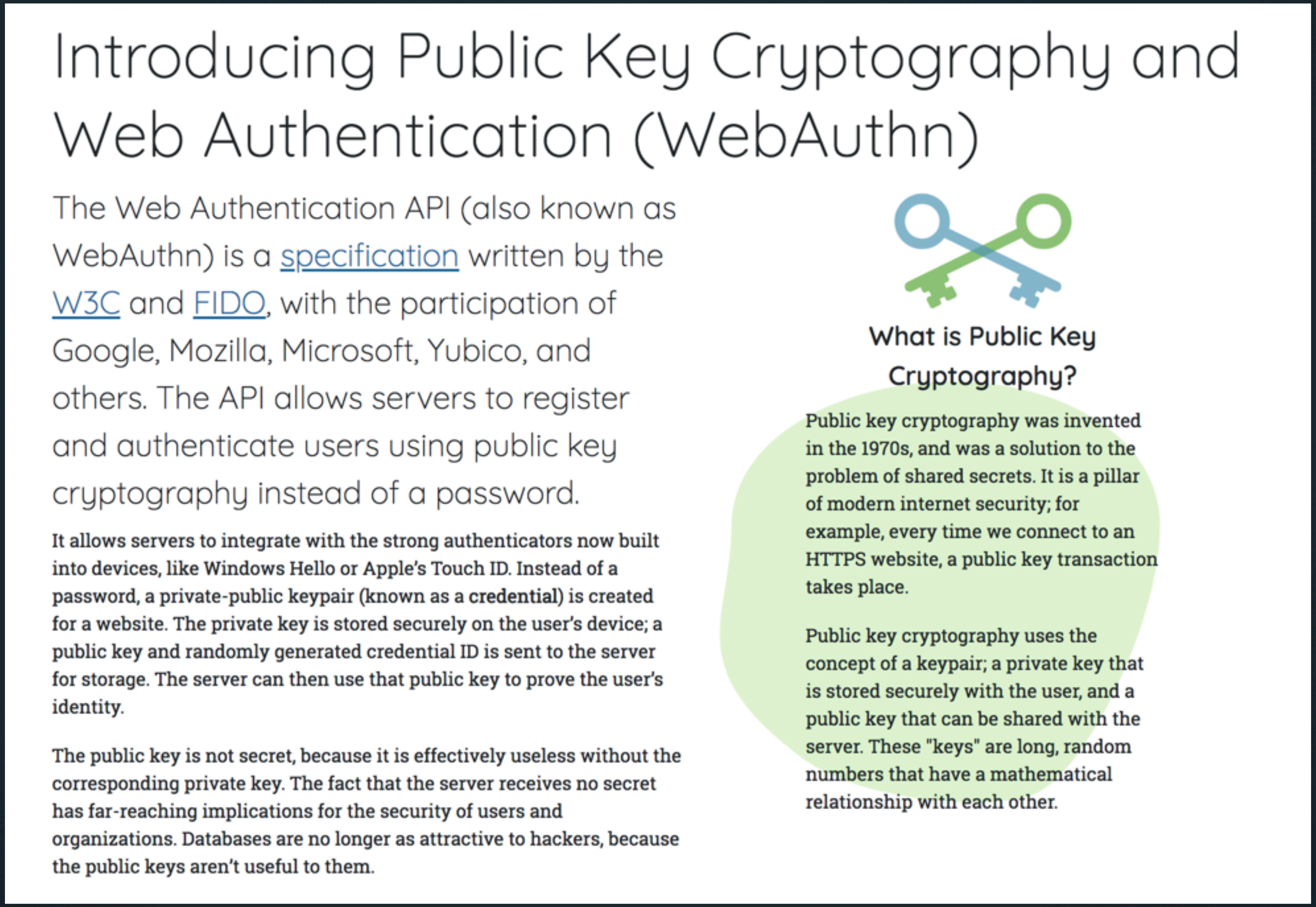
Looking Ahead

WebAuthn

A better alternative for securing
our sensitive information online

@subyraman

https://webauthn.guide
coming soon!



Introducing Public Key Cryptography and Web Authentication (WebAuthn)

The Web Authentication API (also known as WebAuthn) is a specification written by the W3C and FIDO, with the participation of Google, Mozilla, Microsoft, Yubico, and others. The API allows servers to register and authenticate users using public key cryptography instead of a password.

It allows servers to integrate with the strong authenticators now built into devices, like Windows Hello or Apple's Touch ID. Instead of a password, a private-public keypair (known as a **credential**) is created for a website. The private key is stored securely on the user's device; a public key and randomly generated credential ID is sent to the server for storage. The server can then use that public key to prove the user's identity.

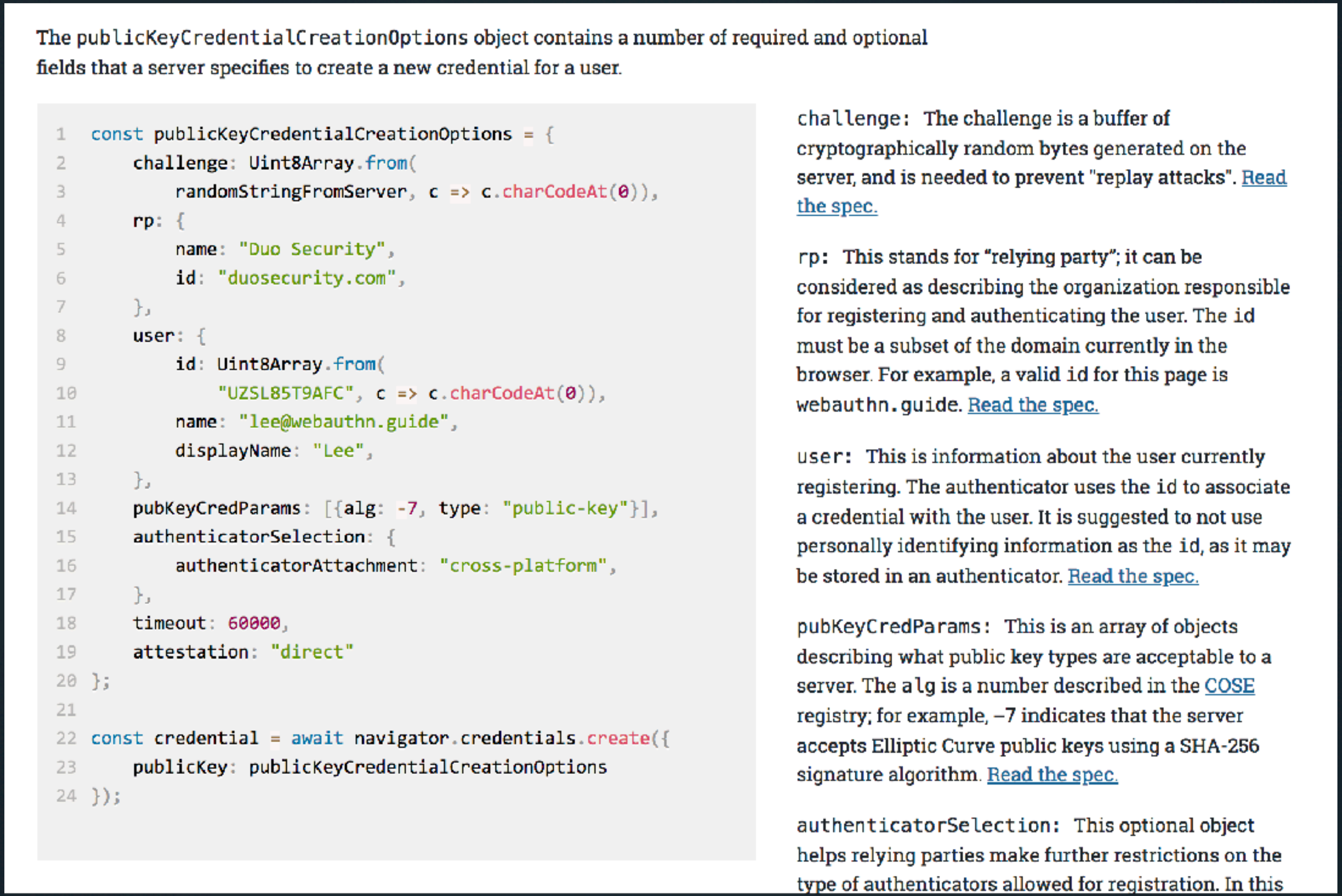
The public key is not secret, because it is effectively useless without the corresponding private key. The fact that the server receives no secret has far-reaching implications for the security of users and organizations. Databases are no longer as attractive to hackers, because the public keys aren't useful to them.

What is Public Key Cryptography?

Public key cryptography was invented in the 1970s, and was a solution to the problem of shared secrets. It is a pillar of modern internet security; for example, every time we connect to an HTTPS website, a public key transaction takes place.

Public key cryptography uses the concept of a keypair; a private key that is stored securely with the user, and a public key that can be shared with the server. These "keys" are long, random numbers that have a mathematical relationship with each other.

https://webauthn.guide
coming soon!



The publicKeyCredentialCreationOptions object contains a number of required and optional fields that a server specifies to create a new credential for a user.

```
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
});
```

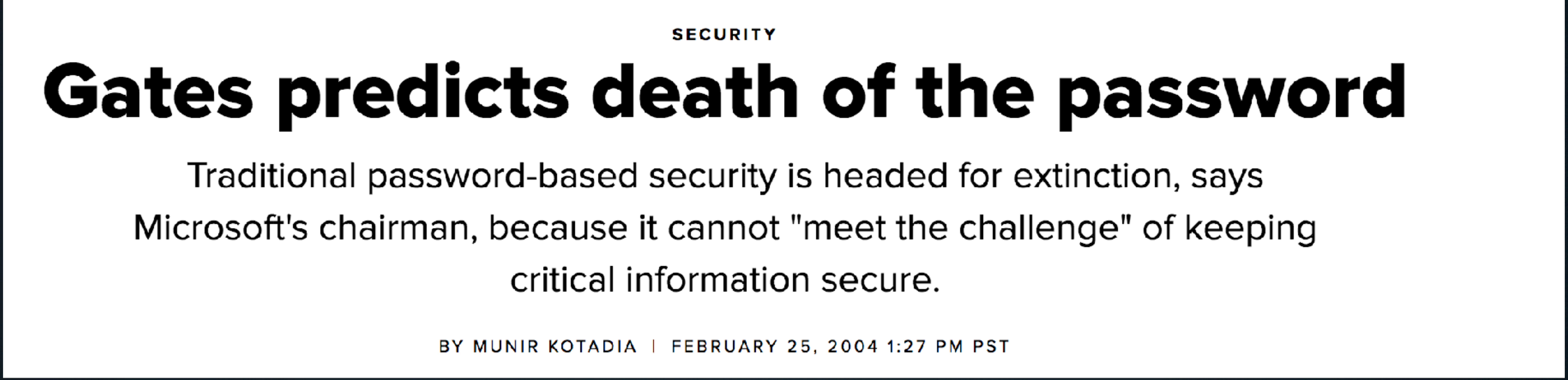
`challenge`: The challenge is a buffer of cryptographically random bytes generated on the server, and is needed to prevent "replay attacks". Read the spec.

`rp`: This stands for "relying party"; it can be considered as describing the organization responsible for registering and authenticating the user. The `id` must be a subset of the domain currently in the browser. For example, a valid `id` for this page is `webauthn.guide`. Read the spec.

`user`: This is information about the user currently registering. The authenticator uses the `id` to associate a credential with the user. It is suggested to not use personally identifying information as the `id`, as it may be stored in an authenticator. Read the spec.

`pubKeyCredParams`: This is an array of objects describing what public key types are acceptable to a server. The `alg` is a number described in the COSE registry; for example, -7 indicates that the server accepts Elliptic Curve public keys using a SHA-256 signature algorithm. Read the spec.

`authenticatorSelection`: This optional object helps relying parties make further restrictions on the type of authenticators allowed for registration. In this

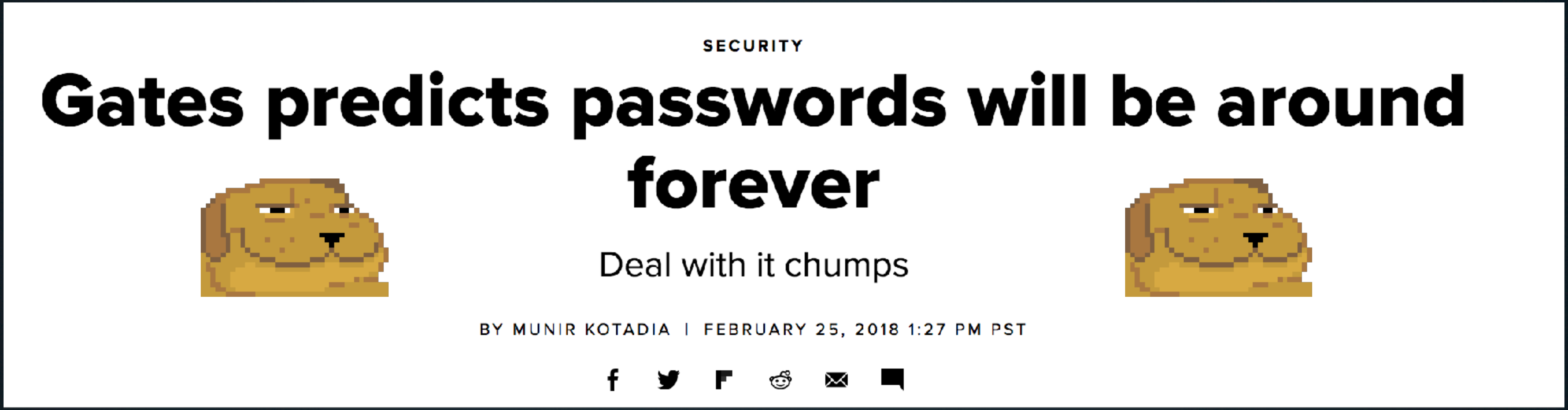


SECURITY

Gates predicts death of the password

Traditional password-based security is headed for extinction, says Microsoft's chairman, because it cannot "meet the challenge" of keeping critical information secure.

BY MUNIR KOTADIA | FEBRUARY 25, 2004 1:27 PM PST

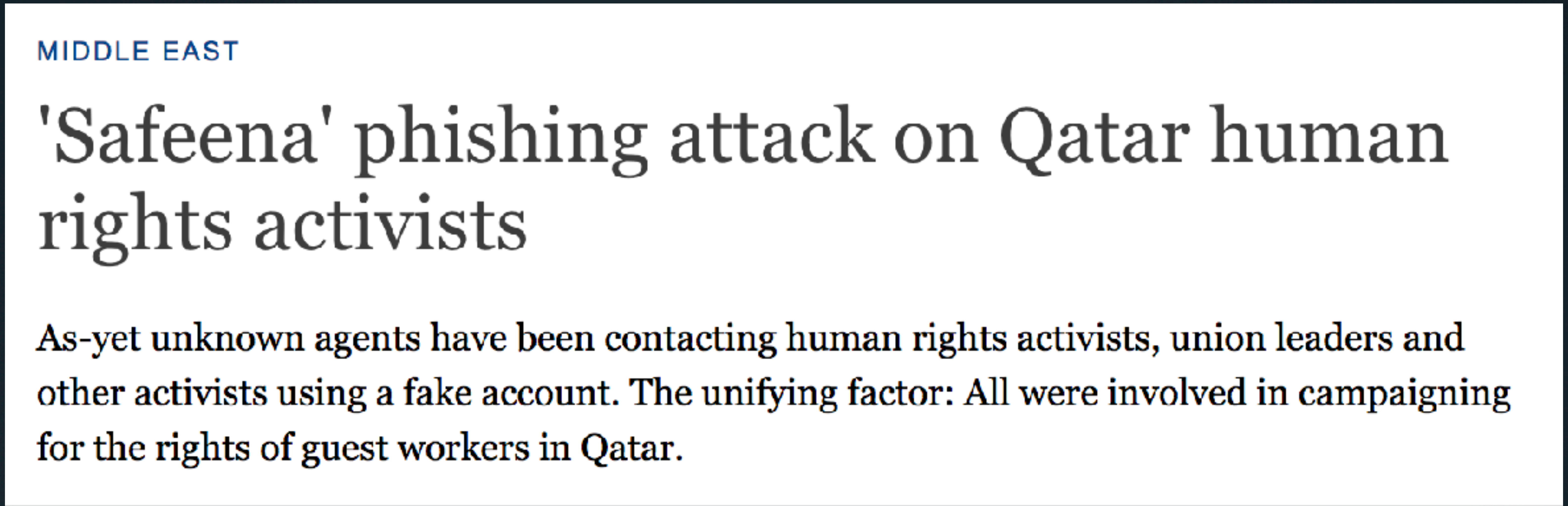


SECURITY

Gates predicts passwords will be around forever

Deal with it chumps

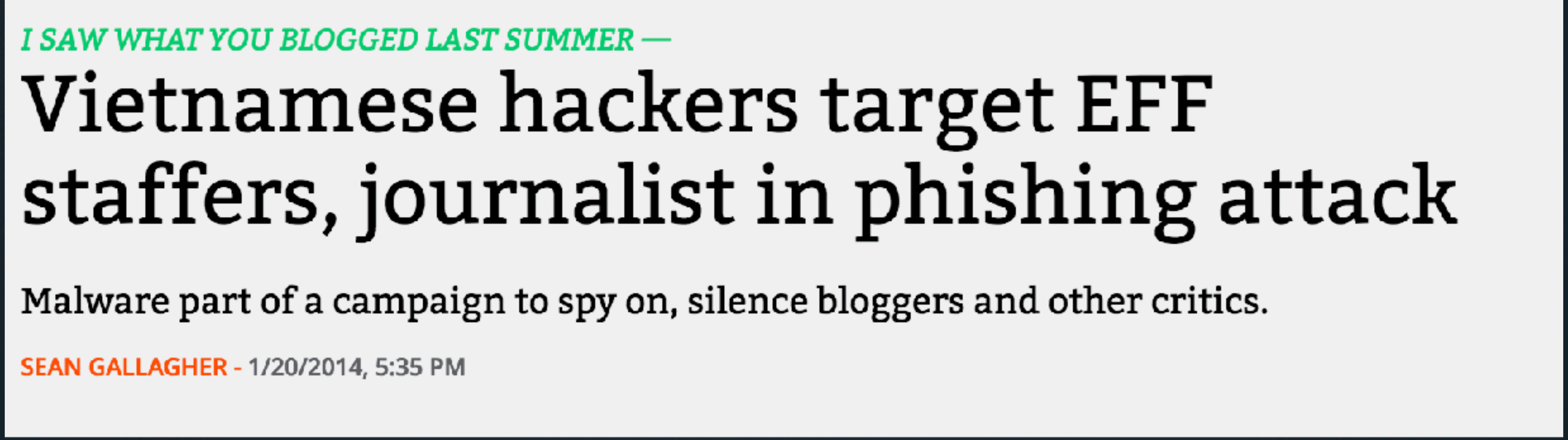
BY MUNIR KOTADIA | FEBRUARY 25, 2018 1:27 PM PST



MIDDLE EAST

'Safeena' phishing attack on Qatar human rights activists

As-yet unknown agents have been contacting human rights activists, union leaders and other activists using a fake account. The unifying factor: All were involved in campaigning for the rights of guest workers in Qatar.



I SAW WHAT YOU BLOGGED LAST SUMMER —

Vietnamese hackers target EFF staffers, journalist in phishing attack

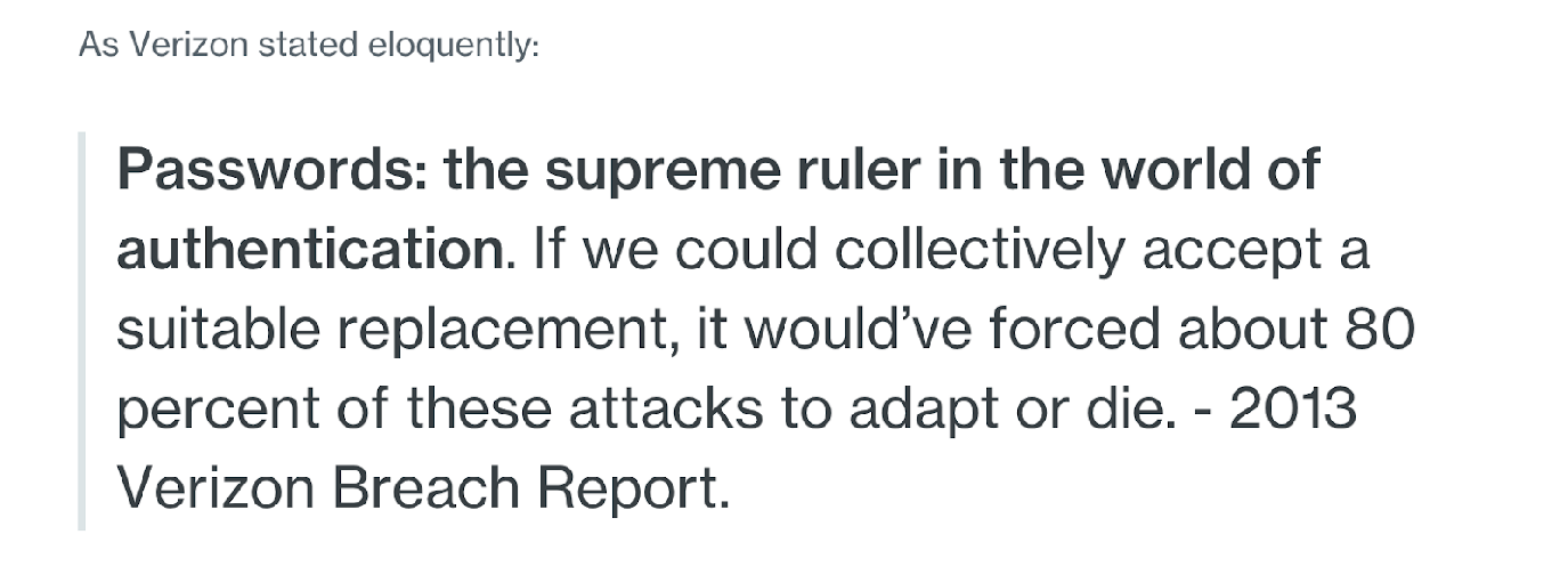
Malware part of a campaign to spy on, silence bloggers and other critics.

SEAN GALLAGHER - 1/20/2014, 5:35 PM

Industry News / Mar 12, 2014

Passwords Aren't Enough: 76% of Breaches Exploit Stolen Credentials

by Thu Pham

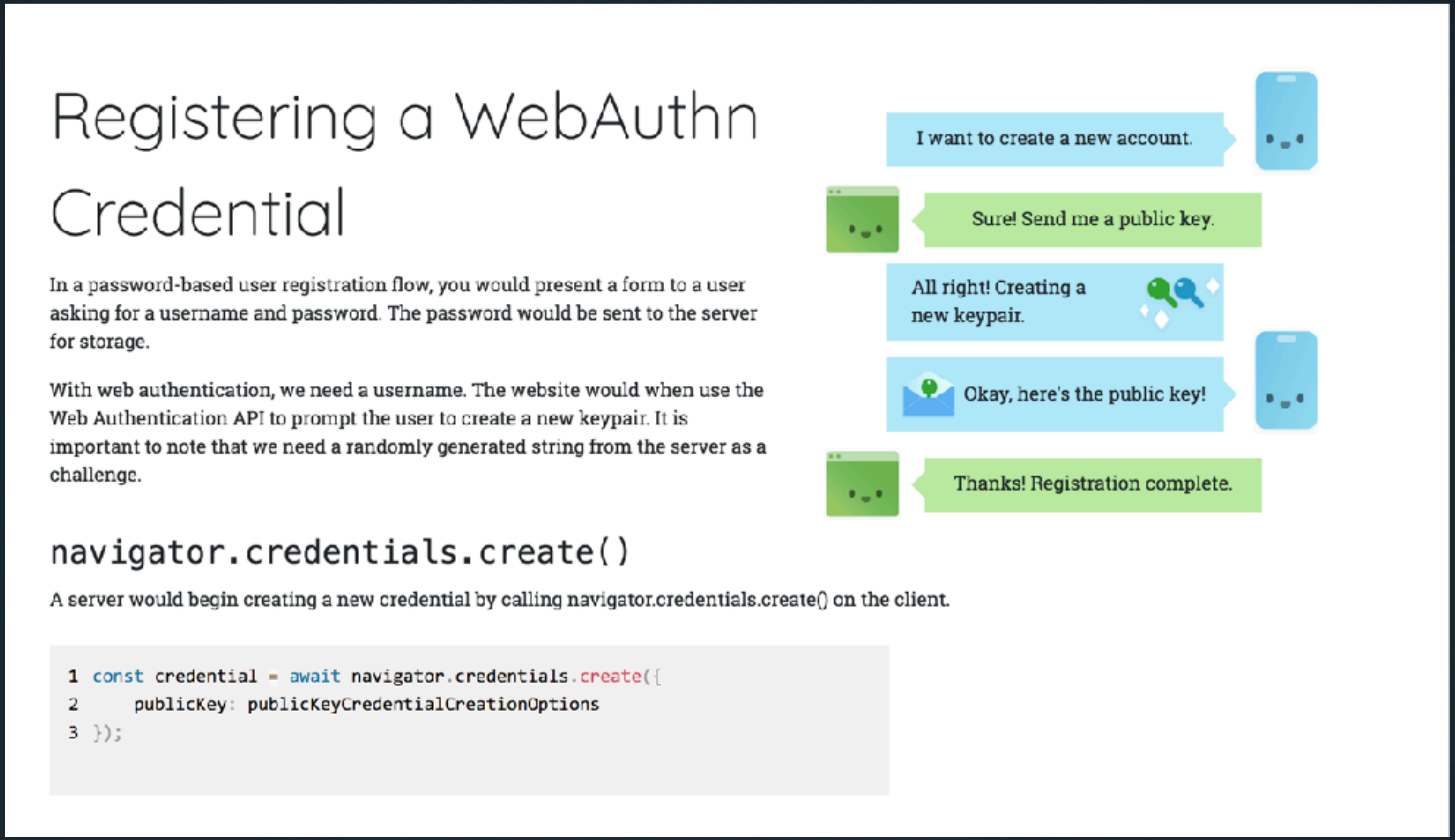


As Verizon stated eloquently:

> **Passwords: the supreme ruler in the world of authentication**. If we could collectively accept a suitable replacement, it would've forced about 80 percent of these attacks to adapt or die. - 2013 Verizon Breach Report.

https://duo.com/blog/passwords-arent-enough-76-of-breaches-exploit-stolen-credentials

https://webauthn.guide



Registering a WebAuthn Credential

In a password-based user registration flow, you would present a form to a user asking for a username and password. The password would be sent to the server for storage.

With web authentication, we need a username. The website would when use the Web Authentication API to prompt the user to create a new keypair. It is important to note that we need a randomly generated string from the server as a challenge.

I want to create a new account.

Sure! Send me a public key.

All right! Creating a new keypair.

Okay, here's the public key!

Thanks! Registration complete.

navigator.credentials.create()

A server would begin creating a new credential by calling navigator.credentials.create() on the client.

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

Suby Raman

@subyraman