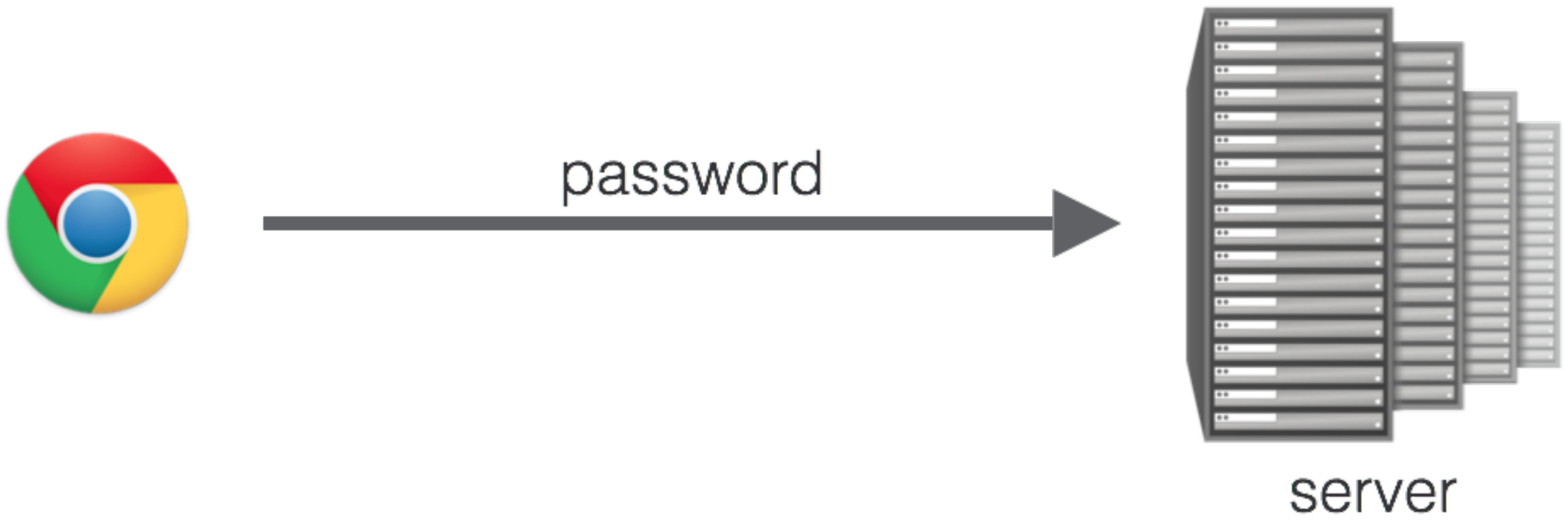
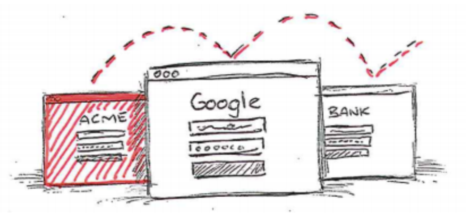
fido alliance | simpler stronger authentication

U2F - Universal 2nd Factor

Alexei Czeskis
(Google)


password
server

password == bearer token



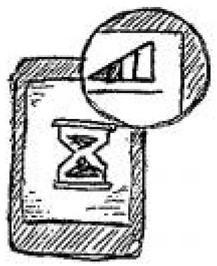
REUSED

PHISHED

INTERCEPTED

KEYLOGGED

Today's solution: One time codes: SMS or Device



SMS USABILITY

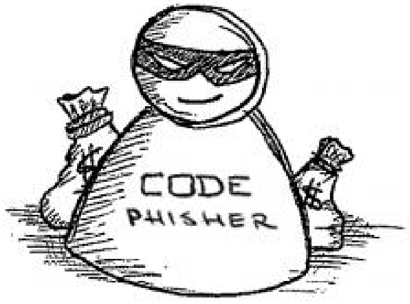
Coverage Issues - Delay - User Cost

DEVICE USABILITY

One Per Site - Expensive - Fragile

USER EXPERIENCE

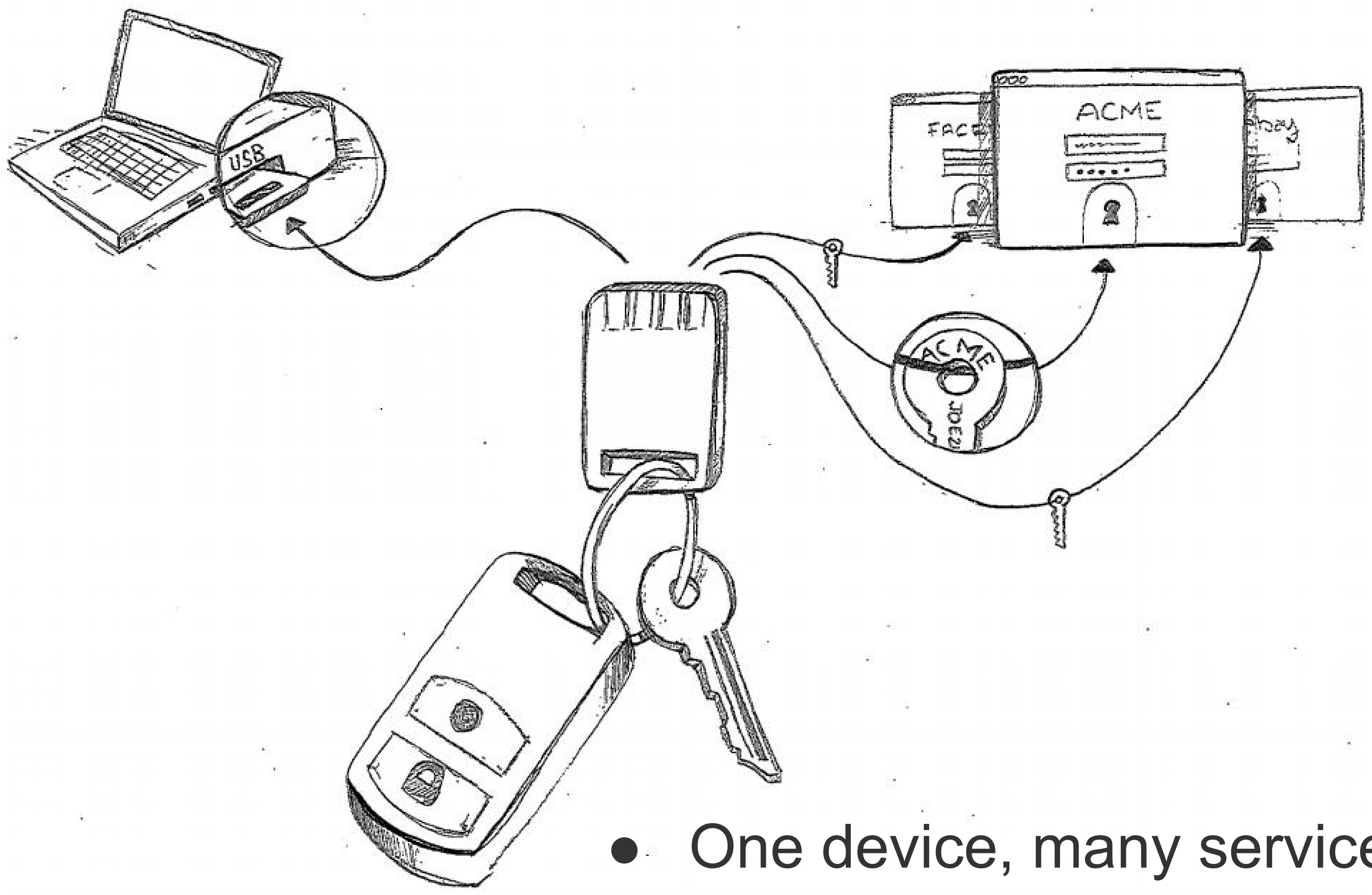
Users find it hard



PHISHABLE

Easy to steal

The U2F solution: How it works



- One device, many services
- Easy: Insert and press button
- Safe: Un-phishable Security

U2F Protocol

Core idea: Standard public key cryptography:

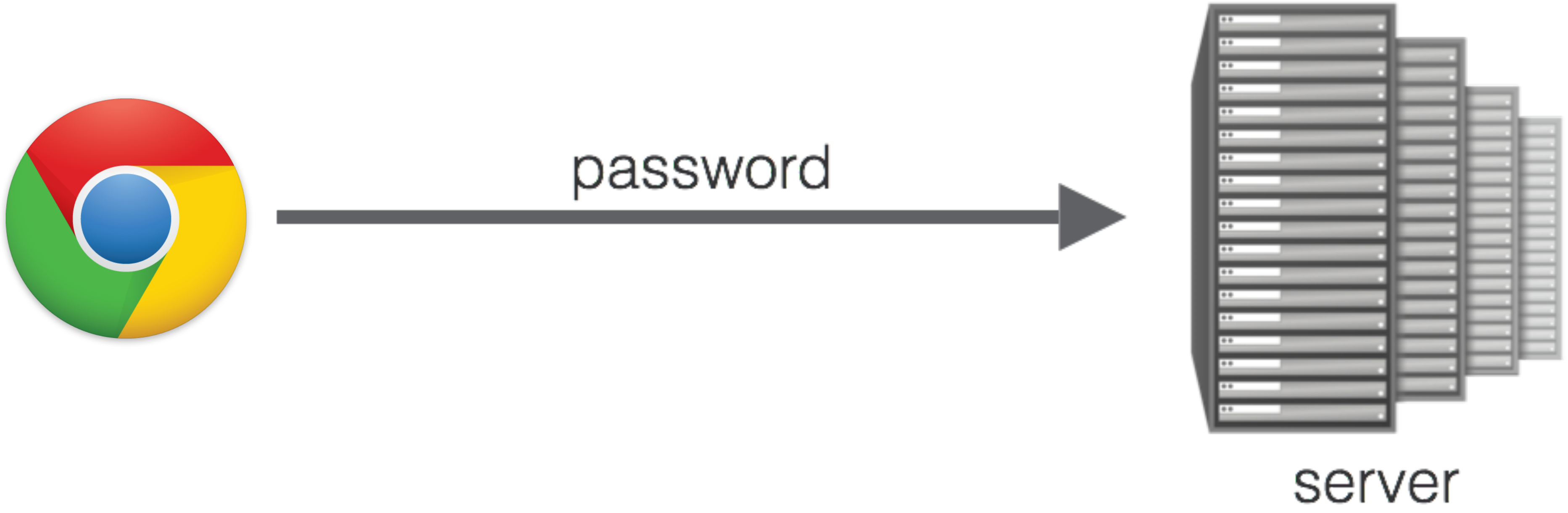
- User's device mints new key pair, gives public key to server
- Server asks user's device to sign data to verify the user.
- **One device, many services, "bring your own device" enabled**

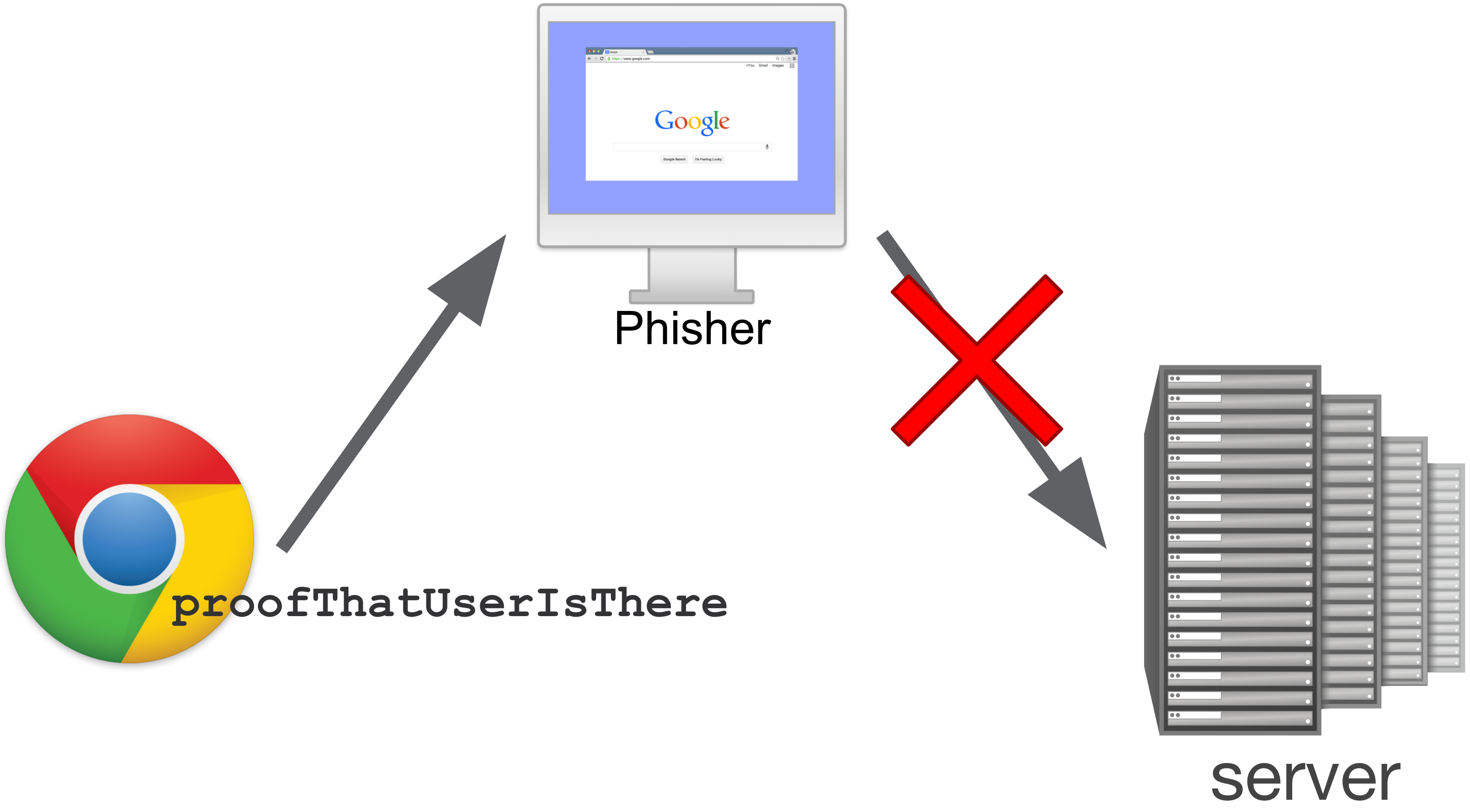
Lots of refinement for this to be consumer facing:

- **Privacy**: Site Specific Keys, No unique ID per device
- **Security:** No phishing, man-in-the-middles
- **Trust:** Verify who made the device
- **Pragmatics**: Affordable today, ride hardware cost curve down
- **Speed for user:** Fast crypto in device (Elliptic Curve)

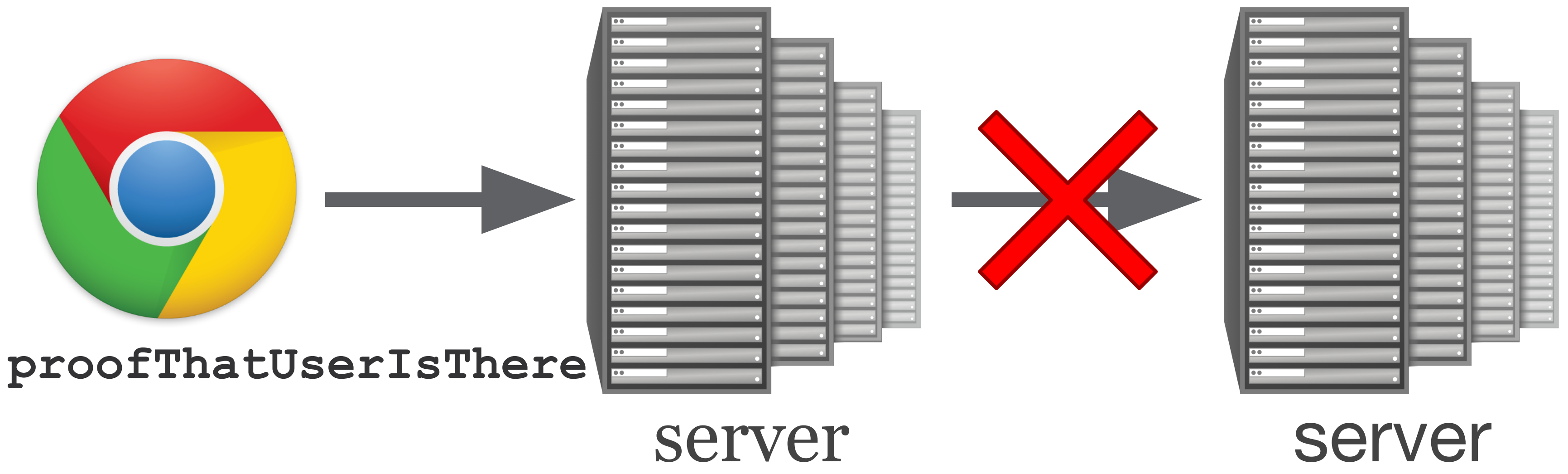
Think "Smartcard re-designed for modern consumer web"

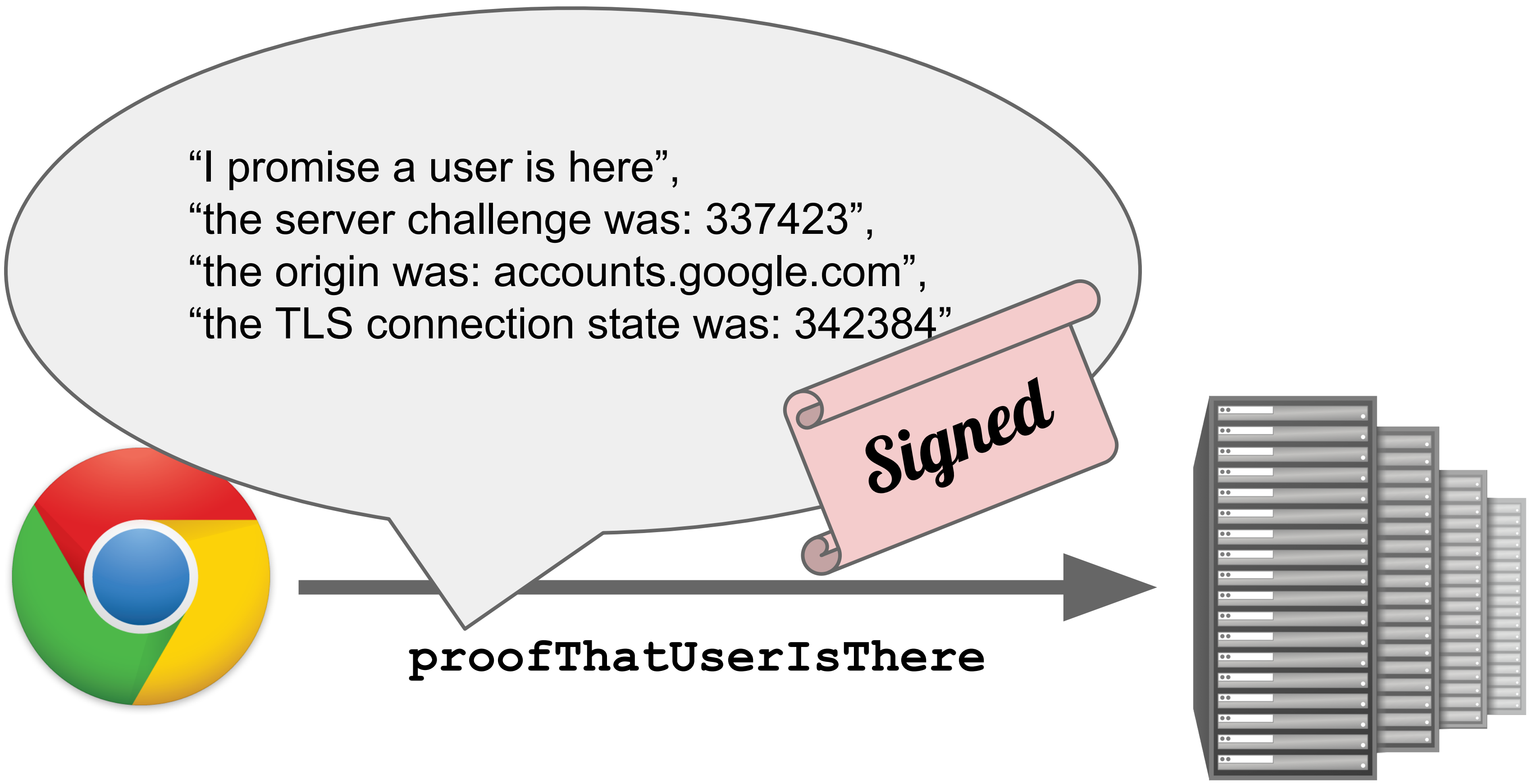
DEMO

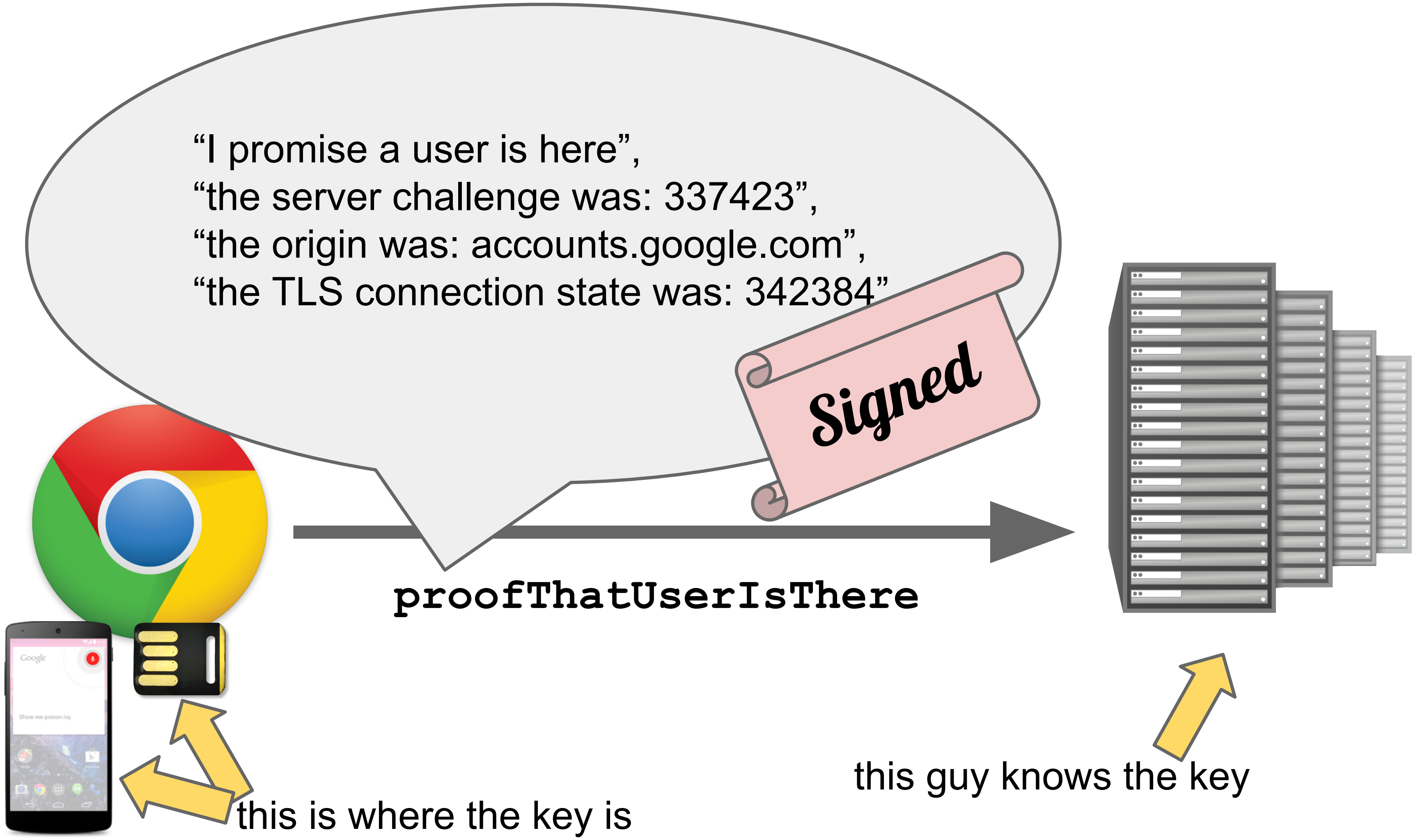

password
server

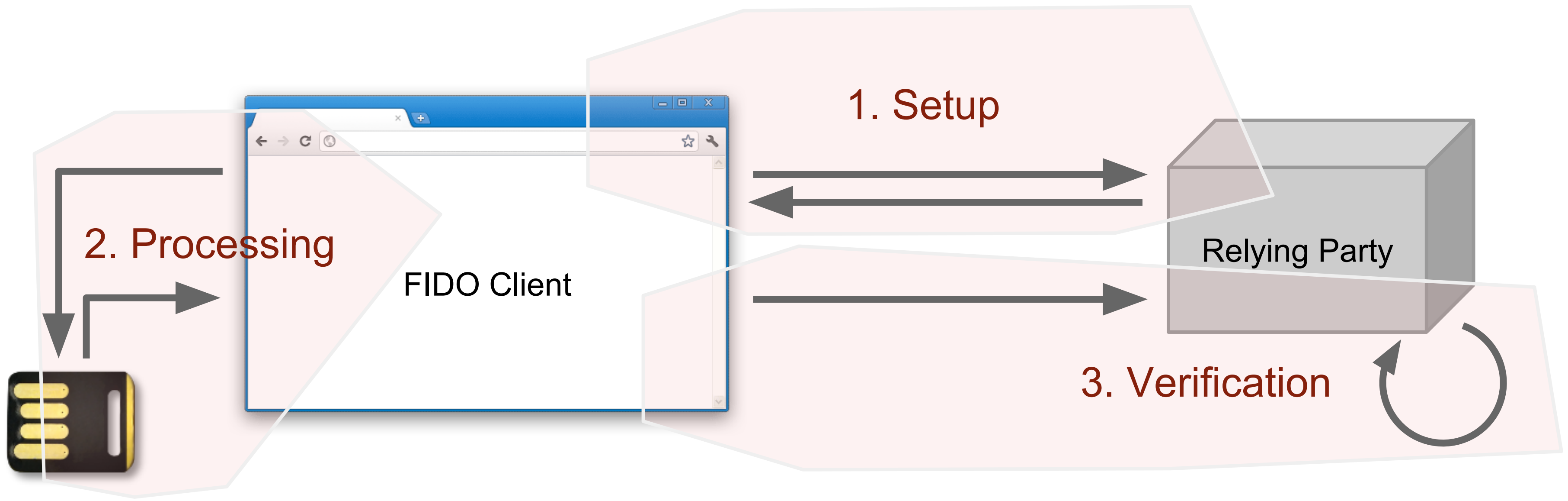


Phisher
proofThatUserIsThere
server

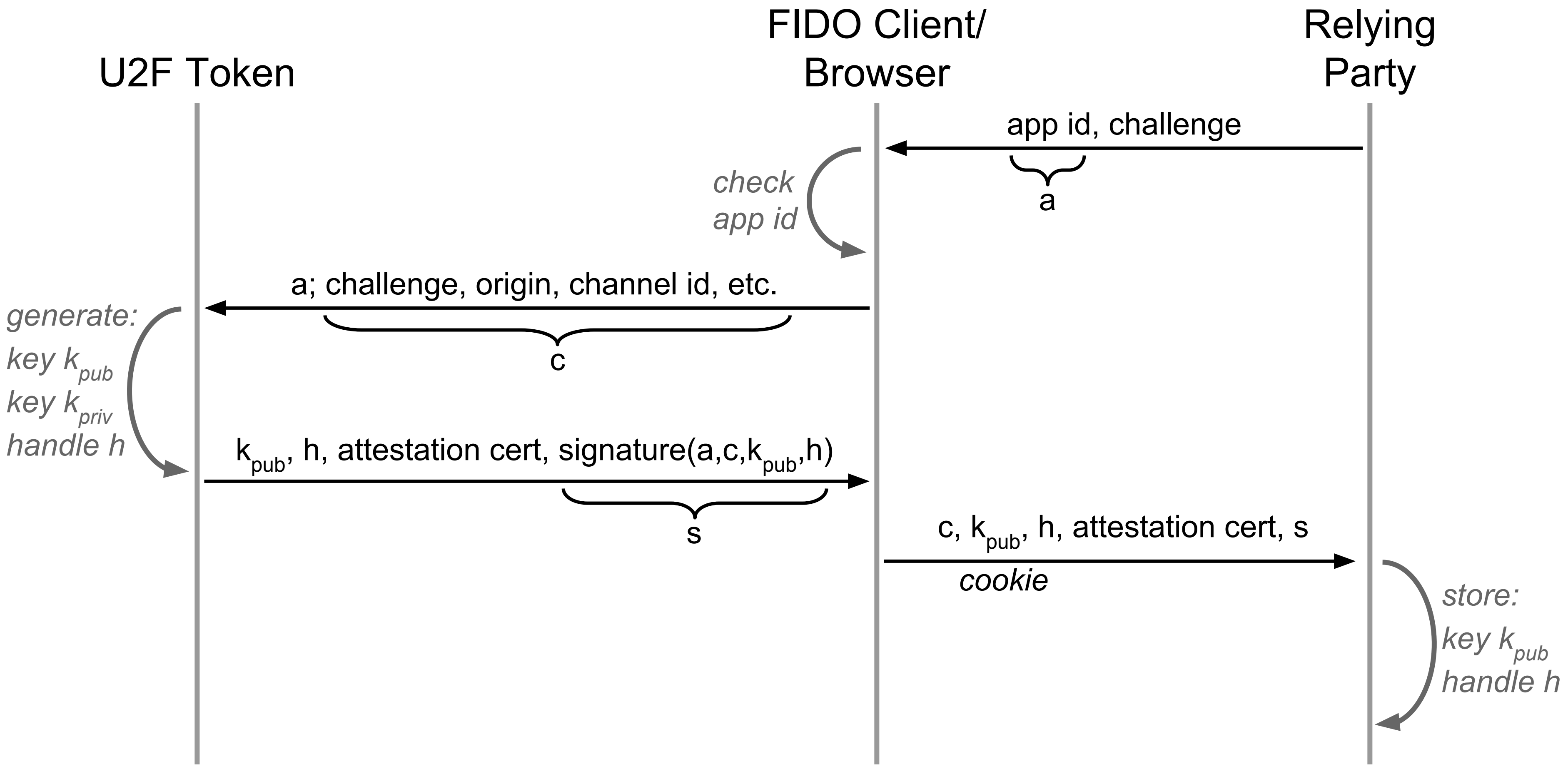

proofThatUserIsThere
server
server


“I promise a user is here”,
“the server challenge was: 337423”,
“the origin was: accounts.google.com”,
“the TLS connection state was: 342384”
Signed
proofThatUserIsThere

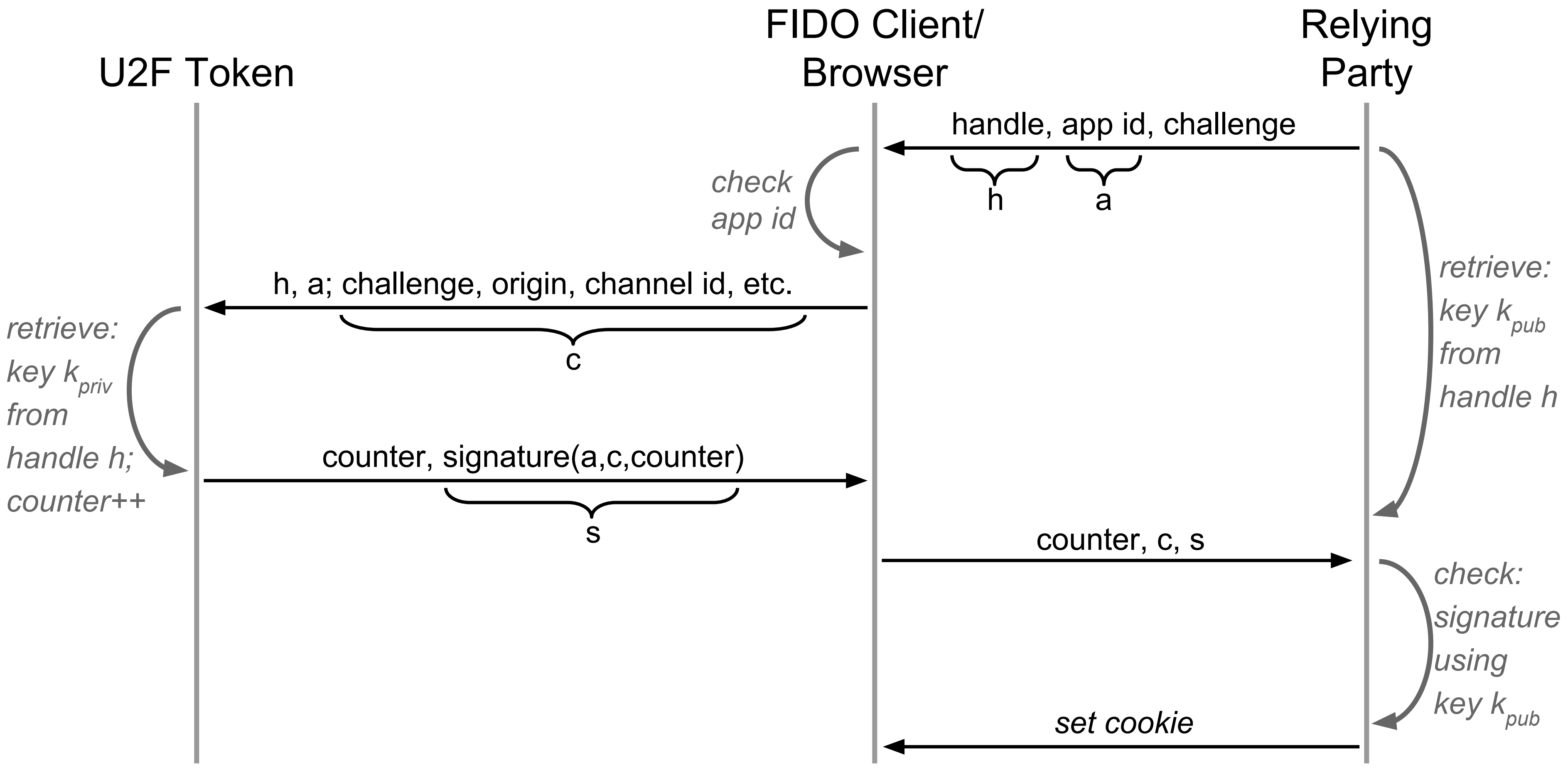

"I promise a user is here",
"the server challenge was: 337423",
"the origin was: accounts.google.com",
"the TLS connection state was: 342384"
Signed
proofThatUserIsThere
this guy knows the key
this is where the key is


1. Setup
2. Processing
FIDO Client
Relying Party
3. Verification

Registration



Authentication


U2F Token
FIDO Client/ Browser
Relying Party
handle, app id, challenge
h
a
check app id
h, a; challenge, origin, channel id, etc.
c
retrieve: key k_{priv} from handle h; counter++
counter, signature(a,c,counter)
s
retrieve: key k_{pub} from handle h
counter, c, s
check: signature using key k_{pub}
set cookie

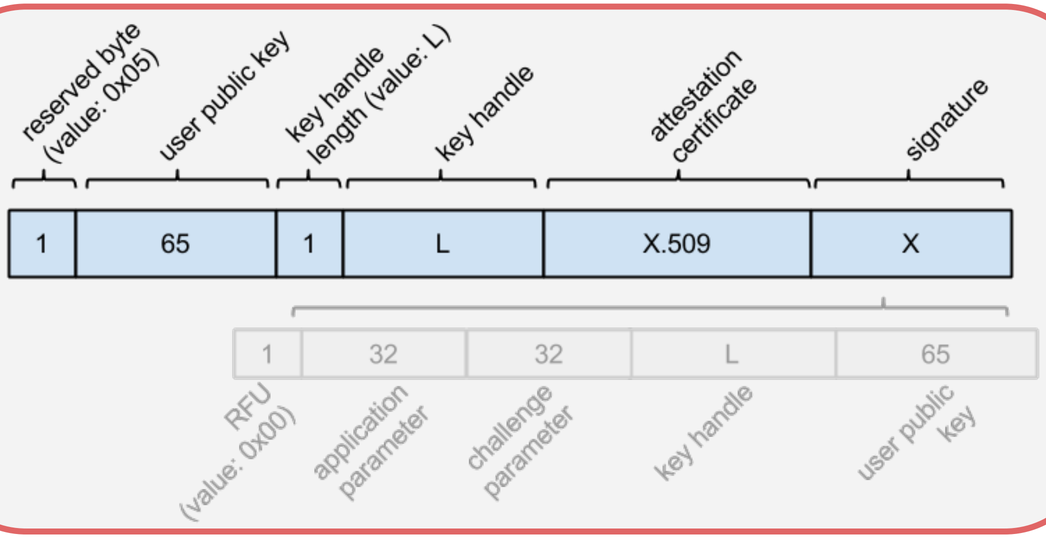
User Presence API

```
u2f.register({
    ‘challenge’: ‘KSDJsdASAS-AIS_AsS’,
    ‘app_id’: ‘https://www.google.com/facets.json’
   }, callback);

callback = function(response) {
  sendToServer(
    response[‘clientData’],
    response[‘tokenData’]);
};
```

Use

```
{
  "typ":"register",
  "challenge":"KSDJsdASAS-AIS_AsS",
  "cid_pubkey": {
    "kty":"EC",
    "crv":"P-256",
    "x":"HzQwlfXX7Q4S5MtCRMzPO9tOyWjBqRl4tJ8",
    "y":"XVguGFLIZx1fXg375hi4-7-BxhMljw42Ht4"
  },
  "origin":"https://accounts.google.com"
}
```

```
u2f

                                        s.json'

callback = fu        (res

  sendToServer(
    response['client
    response['tokenData

};
```



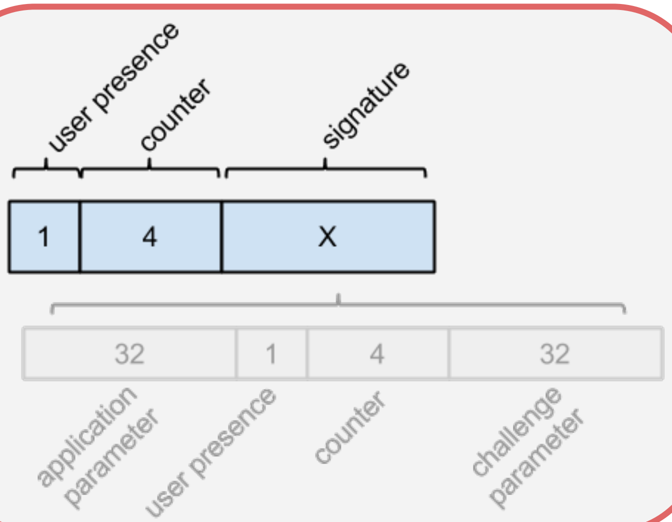
User Presence API

```
u2f.sign({
    ‘challenge’: ‘KSDJsdASAS-AIS_AsS’,
    ‘app_id’: ‘https://www.google.com/facets.json’,
    ‘key_handle’: ‘JkjhdsfkjSDFKJ_ld-sadsAJDKLSAD’
   }, callback);

callback = function(response) {
  sendToServer(
    response[‘clientData’],
    response[‘tokenData’]);
};
```

User Presence API

```
u2f.sign({
    'challenge': 'K
    'app_id': 'http
    'key_handle': '
  }, callback);

callback = function(r          e) {
  sendToServer(
    response['clientData'],
    response['tokenData'  ]);
};
```

```
{
  "typ":"authenticate",
  "challenge":"KSDJsdASAS-AIS_AsS",
  "cid_pubkey": {
    "kty":"EC",
    "crv":"P-256",
    "x":"HzQwlfXX7Q4S5MtCRMzPO9tOyWjBqRl4tJ8",
    "y":"XVguGFLIZx1fXg375hi4-7-BxhMljw42Ht4"
  },
  "origin":"https://accounts.google.com"
}
```



What if…
...I want to accept U2F logins?

- Browser: Call JS APIs
  - native APIs since Chrome 41+
- Server: Implement registration flow
  - decide how to handle attestation certificates
  - verify registration response
  - store public key, key handle with user account
- Server: Implement login flow
  - check username/password, look up key handle
  - verify authentication response (origin, signature, counter, …)
- Check your account recovery flow

What if…
...I want to offer a USB U2F token?

- Implement ECDSA P-256
- Implement counter
- Decide on key handle strategy
  - must recover private key, app id
- Implement USB framing spec
- No responses without user presence!
  - (with one exception)
  - check that app id matches

What if…
...I want to offer a NFC/BLE/... token?

- Come join FIDO!

What if…
...I have a different token form factor?

- Come join FIDO!

Next Steps

- Other platforms: browsers on Android, etc.
- Other platforms: native apps on Android, etc.
- Other message framing: BLE, NFC, UICC, etc.

Next Steps

Read the Specs: https://fidoalliance.org/specifications/download/ (U2F)

Get the Reference Source: https://github.com/google/u2f-ref-code/

Play with the Demo server: https://u2fdemo.appspot.com/

Get a Device: http://smile.amazon.com/s/ref=sr_kk_1?rh=k:u2f (search for *u2f*)

Thanks!