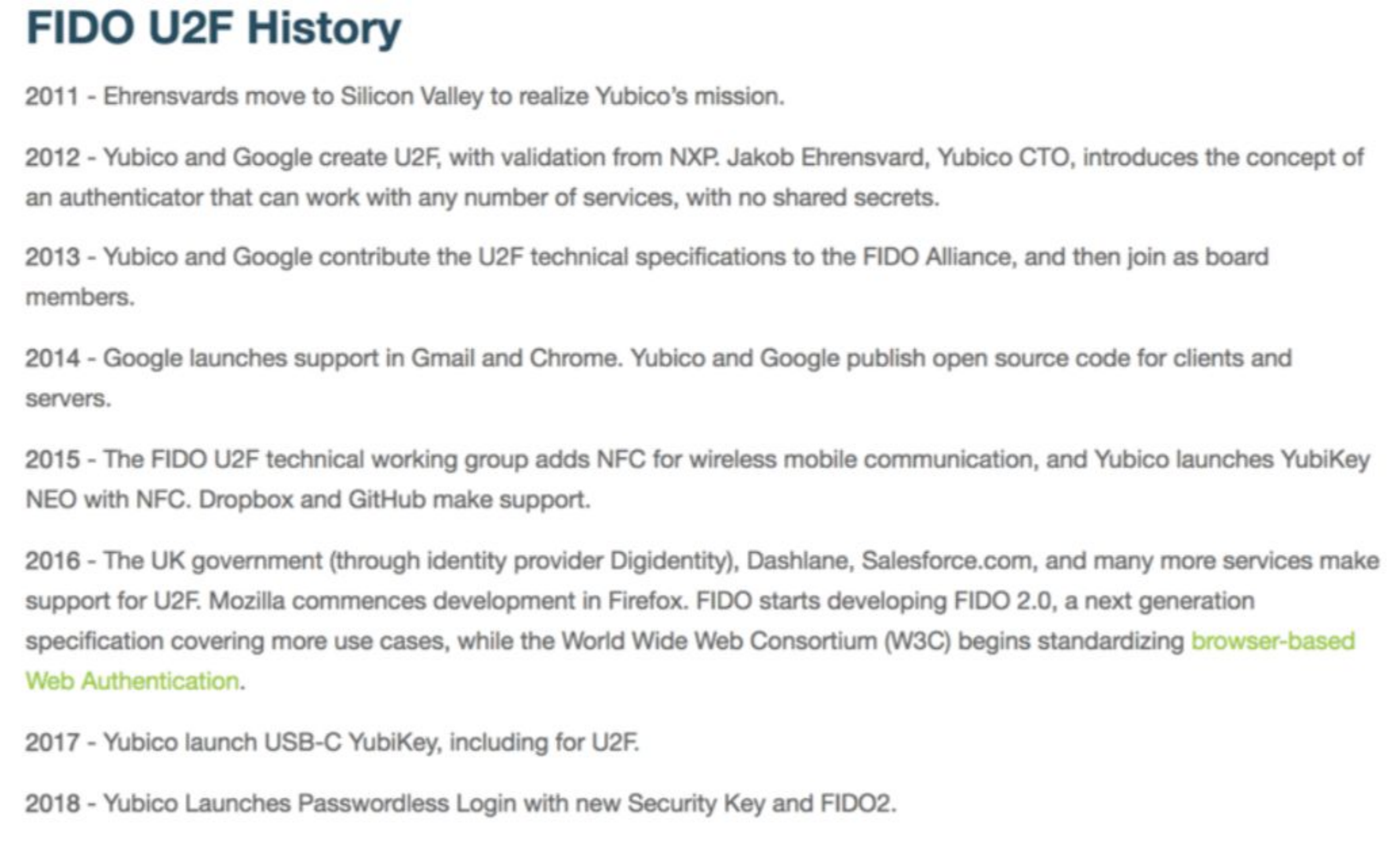
U2Fishing: Potential Security Threat Introduced by U2F Key Wrapping Mechanism

Wang Kang
Alibaba Group

Introduction: U2F

- U2F: Universal 2 Factor
- FIDO: Fast IDentity Online
- Manufacturer: Yubikey, Nitrokey, FeiTian
- Chrome native support; other browsers on the way
- Driver-free: USB-HID
    - Also over BTLE or NFC



FIDO U2F History

2011 - Ehrensvards move to Silicon Valley to realize Yubico's mission.

2012 - Yubico and Google create U2F, with validation from NXP. Jakob Ehrensvard, Yubico CTO, introduces the concept of an authenticator that can work with any number of services, with no shared secrets.

2013 - Yubico and Google contribute the U2F technical specifications to the FIDO Alliance, and then join as board members.

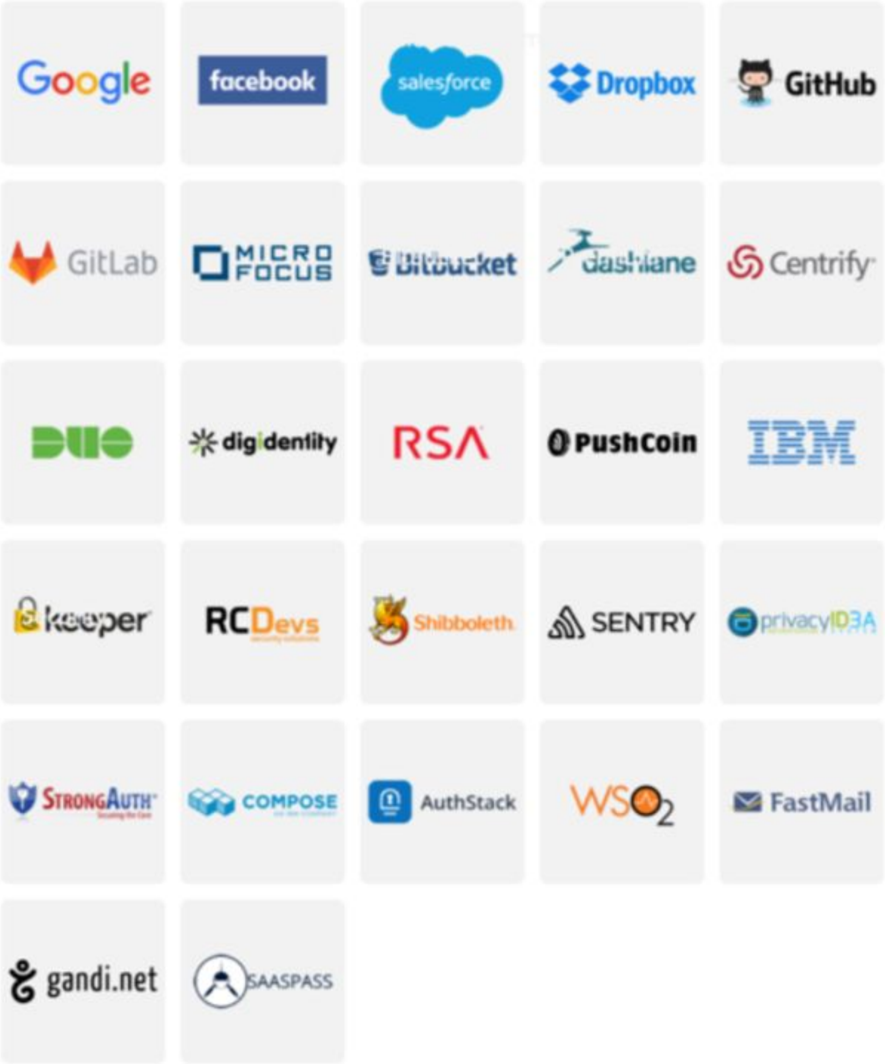
2014 - Google launches support in Gmail and Chrome. Yubico and Google publish open source code for clients and servers.

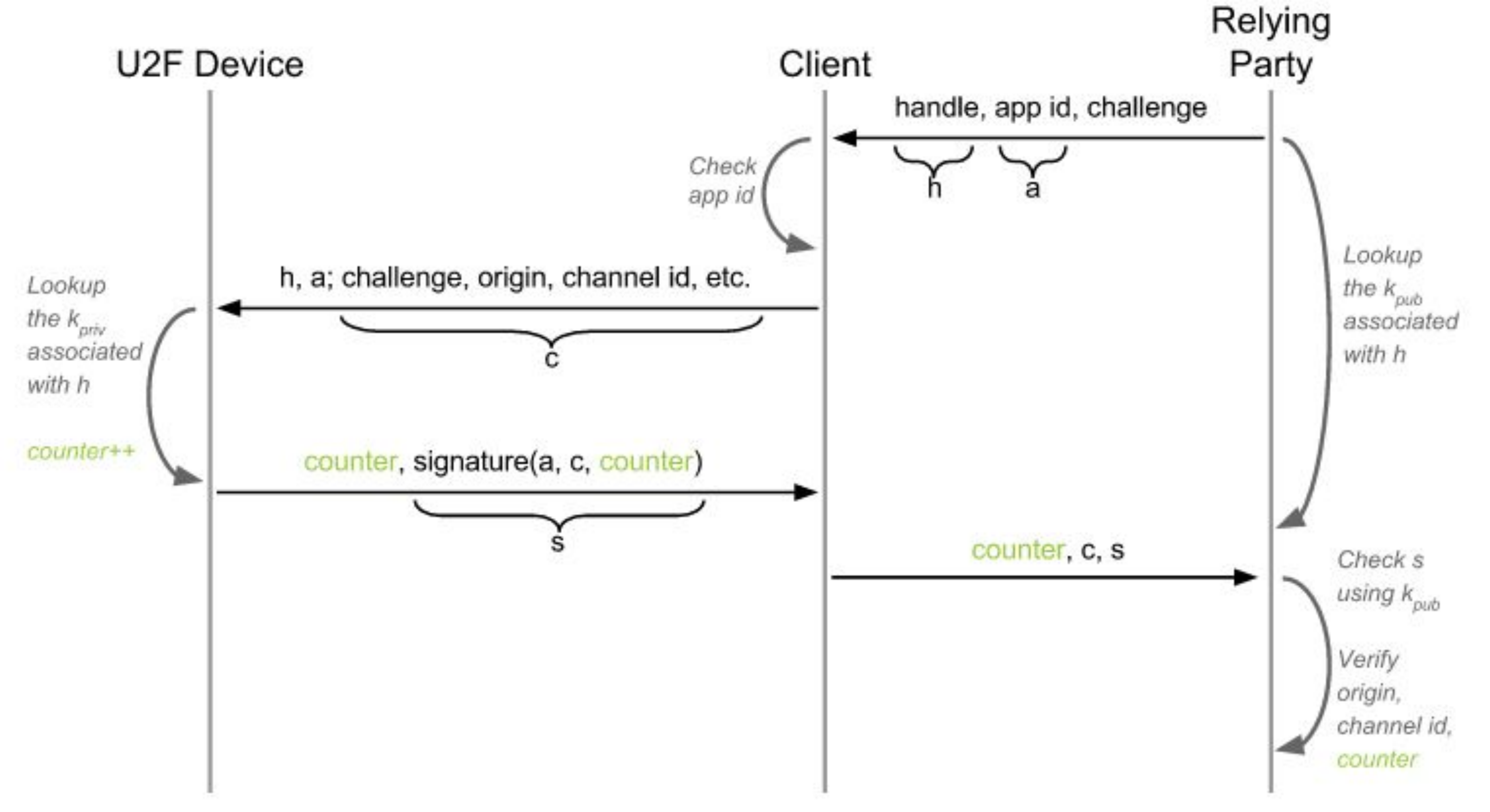
2015 - The FIDO U2F technical working group adds NFC for wireless mobile communication, and Yubico launches YubiKey NEO with NFC. Dropbox and GitHub make support.

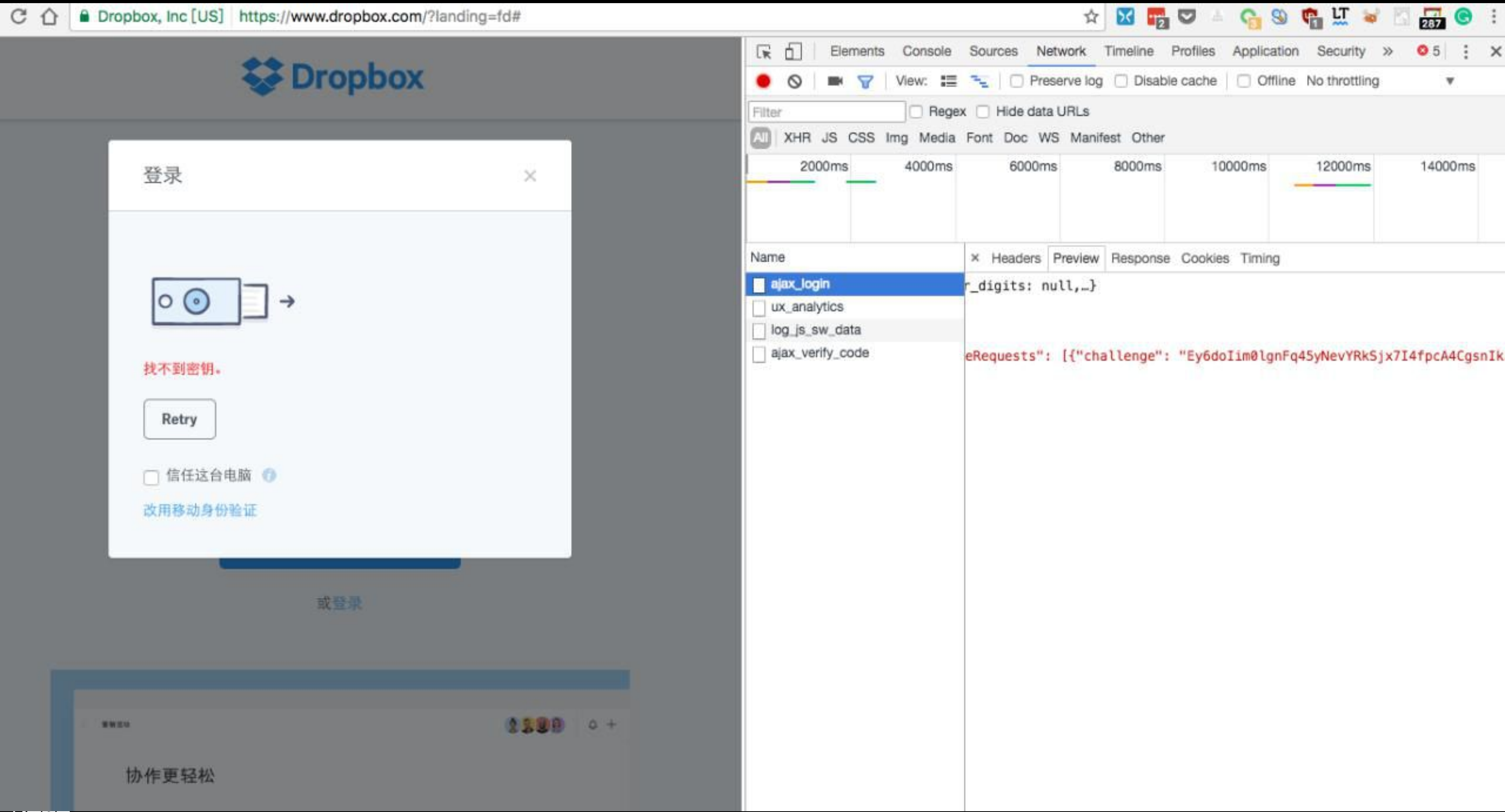
2016 - The UK government (through identity provider Digidentity), Dashlane, Salesforce.com, and many more services make support for U2F. Mozilla commences development in Firefox. FIDO starts developing FIDO 2.0, a next generation specification covering more use cases, while the World Wide Web Consortium (W3C) begins standardizing browser-based Web Authentication.

2017 - Yubico launch USB-C YubiKey, including for U2F.

2018 - Yubico Launches Passwordless Login with new Security Key and FIDO2.


Google
facebook
salesforce
Dropbox
GitHub
GitLab
MICRO FOCUS
Bitbucket
dashlane
Centrify
DUO
digidentity
RSA
PushCoin
IBM
keeper
RCDevs
Shibboleth
SENTRY
privacyIDEA
StrongAuth
COMPOSE
AuthStack
WSO2
FastMail
gandi.net
SAASPASS


U2F Device
Client
Relying Party
handle, app id, challenge
h
a
Check app id
Lookup the k_{pub} associated with h
h, a; challenge, origin, channel id, etc.
c
Lookup the k_{priv} associated with h
counter++
counter, signature(a, c, counter)
s
counter, c, s
Check s using k_{pub}
Verify origin, channel id, counter



登录

找不到密钥。

Retry

□ 信任这台电脑

改用移动身份验证

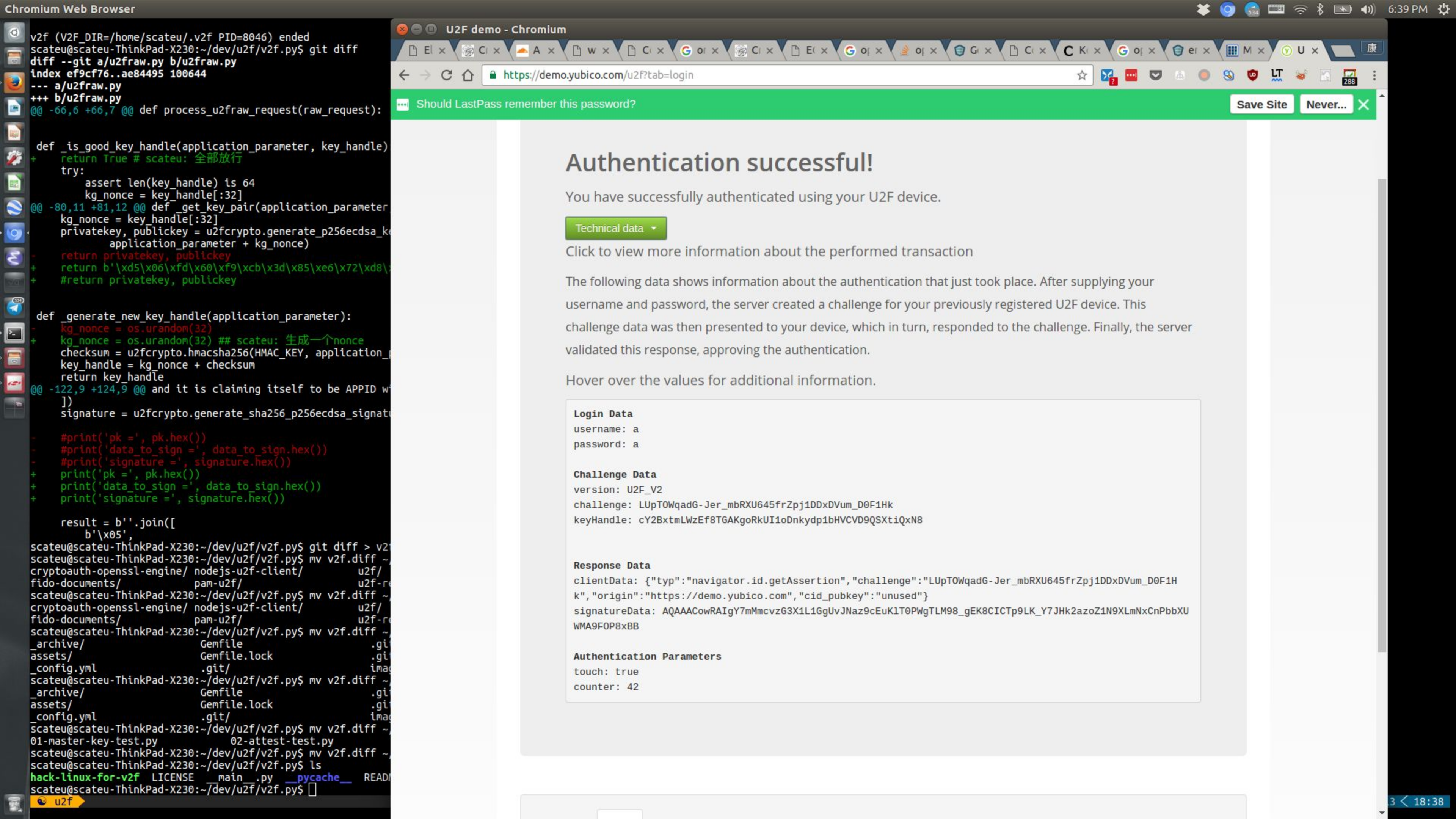
或登录

协作更轻松

```
ajax_login
ux_analytics
log_js_sw_data
ajax_verify_code
```

```
r_digits: null,…}

eRequests": [{"challenge": "Ey6doIim0lgnFq45yNevYRkSjx7I4fpcA4CgsnIk
```



```
v2f (V2F_DIR=/home/scateu/.v2f PID=8046) ended
scateu@scateu-ThinkPad-X230:~/dev/u2f/v2f.py$ git diff
diff --git a/u2fraw.py b/u2fraw.py
index ef9cf76..ae84495 100644
--- a/u2fraw.py
+++ b/u2fraw.py
@@ -66,6 +66,7 @@ def process_u2fraw_request(raw_request):


 def _is_good_key_handle(application_parameter, key_handle)
+    return True # scateu: 全部放行
     try:
         assert len(key_handle) is 64
         kg_nonce = key_handle[:32]
@@ -80,11 +81,12 @@ def _get_key_pair(application_parameter
     kg_nonce = key_handle[:32]
     privatekey, publickey = u2fcrypto.generate_p256ecdsa_ke
             application_parameter + kg_nonce)
-    return privatekey, publickey
+    return b'\xd5\x06\xfd\x60\xf9\xcb\x3d\x85\xe6\x72\xd8\
+    #return privatekey, publickey


 def _generate_new_key_handle(application_parameter):
-    kg_nonce = os.urandom(32)
+    kg_nonce = os.urandom(32) ## scateu: 生成一个nonce
     checksum = u2fcrypto.hmacsha256(HMAC_KEY, application_
     key_handle = kg_nonce + checksum
     return key_handle
@@ -122,9 +124,9 @@ and it is claiming itself to be APPID w
     ])
     signature = u2fcrypto.generate_sha256_p256ecdsa_signatu

-    #print('pk =', pk.hex())
-    #print('data_to_sign =', data_to_sign.hex())
-    #print('signature =', signature.hex())
+    print('pk =', pk.hex())
+    print('data_to_sign =', data_to_sign.hex())
+    print('signature =', signature.hex())

     result = b''.join([
         b'\x05',
scateu@scateu-ThinkPad-X230:~/dev/u2f/v2f.py$ git diff > v2
scateu@scateu-ThinkPad-X230:~/dev/u2f/v2f.py$ mv v2f.diff ~
cryptoauth-openssl-engine/ nodejs-u2f-client/         u2f/
fido-documents/            pam-u2f/                   u2f-r
scateu@scateu-ThinkPad-X230:~/dev/u2f/v2f.py$ mv v2f.diff ~
cryptoauth-openssl-engine/ nodejs-u2f-client/         u2f/
fido-documents/            pam-u2f/                   u2f-r
scateu@scateu-ThinkPad-X230:~/dev/u2f/v2f.py$ mv v2f.diff ~
_archive/                   Gemfile                    .gi
assets/                     Gemfile.lock               .gi
_config.yml                 .git/                      imag
scateu@scateu-ThinkPad-X230:~/dev/u2f/v2f.py$ mv v2f.diff ~
_archive/                   Gemfile                    .gi
assets/                     Gemfile.lock               .gi
_config.yml                 .git/                      imag
scateu@scateu-ThinkPad-X230:~/dev/u2f/v2f.py$ mv v2f.diff ~
01-master-key-test.py            02-attest-test.py
scateu@scateu-ThinkPad-X230:~/dev/u2f/v2f.py$ mv v2f.diff ~
scateu@scateu-ThinkPad-X230:~/dev/u2f/v2f.py$ ls
hack-linux-for-v2f  LICENSE  __main__.py  __pycache__  READ
scateu@scateu-ThinkPad-X230:~/dev/u2f/v2f.py$ 
 u2f
```

Authentication successful!

You have successfully authenticated using your U2F device.

Technical data

Click to view more information about the performed transaction

The following data shows information about the authentication that just took place. After supplying your username and password, the server created a challenge for your previously registered U2F device. This challenge data was then presented to your device, which in turn, responded to the challenge. Finally, the server validated this response, approving the authentication.

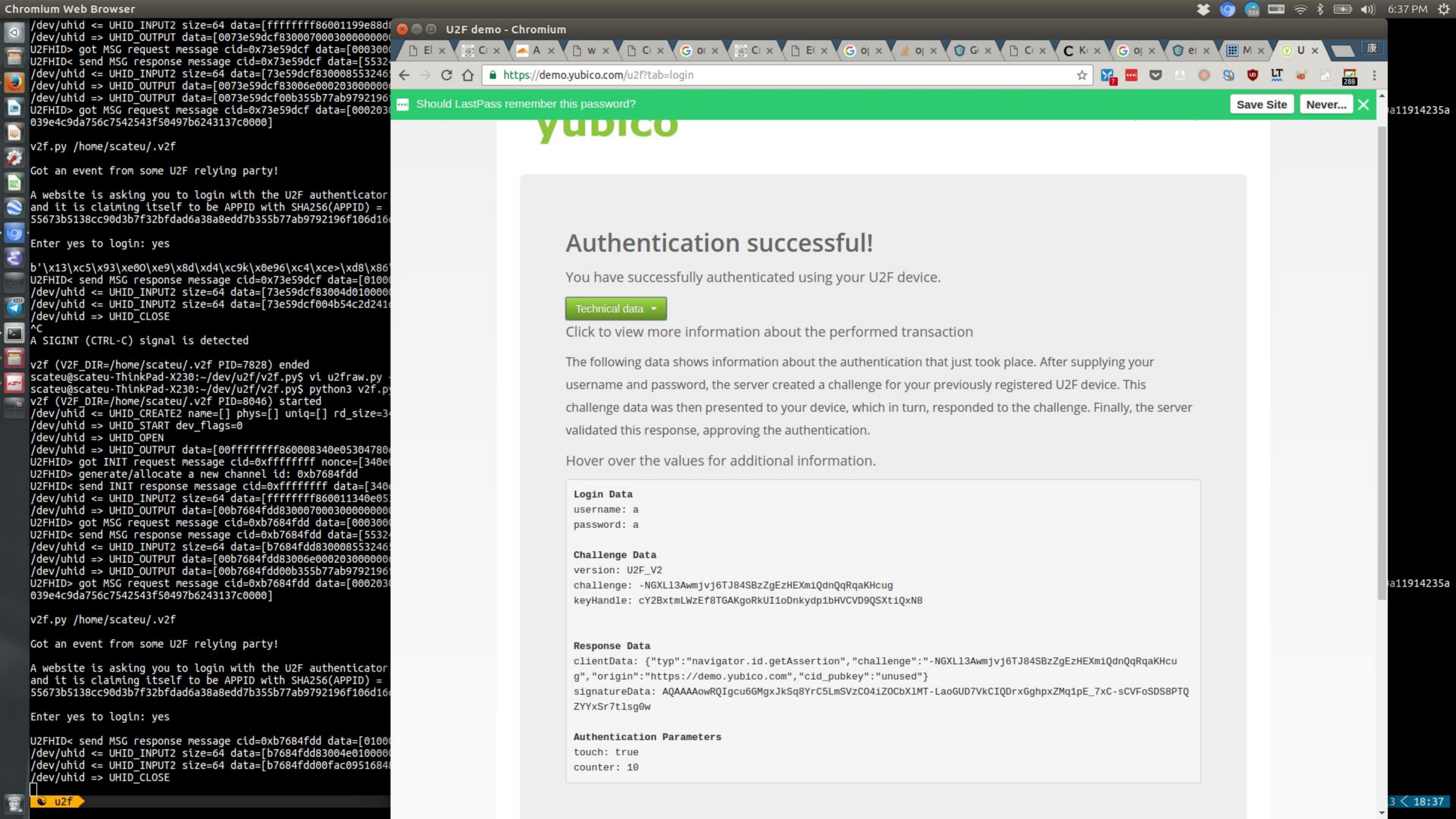
Hover over the values for additional information.

```
Login Data
username: a
password: a

Challenge Data
version: U2F_V2
challenge: LUpTOWqadG-Jer_mbRXU645frZpj1DDxDVum_D0F1Hk
keyHandle: cY2BxtmLWzEf8TGAKgoRkUI1oDnkydp1bHVCVD9QSXtiQxN8


Response Data
clientData: {"typ":"navigator.id.getAssertion","challenge":"LUpTOWqadG-Jer_mbRXU645frZpj1DDxDVum_D0F1H
k","origin":"https://demo.yubico.com","cid_pubkey":"unused"}
signatureData: AQAAACowRAIgY7mMmcvzG3X1L1GgUvJNaz9cEuKlT0PWgTLM98_gEK8CICTp9LK_Y7JHk2azoZ1N9XLmNxCnPbbXU
WMA9FOP8xBB

Authentication Parameters
touch: true
counter: 42
```



```
/dev/uhid <= UHID_INPUT2 size=64 data=[ffffffff86001199e88d
/dev/uhid => UHID_OUTPUT data=[0073e59dcf830007000300000000
U2FHID> got MSG request message cid=0x73e59dcf data=[000300
U2FHID< send MSG response message cid=0x73e59dcf data=[5532
/dev/uhid <= UHID_INPUT2 size=64 data=[73e59dcf83000855324
/dev/uhid => UHID_OUTPUT data=[0073e59dcf83006e000203000000
/dev/uhid => UHID_OUTPUT data=[0073e59dcf00b355b77ab979219
U2FHID> got MSG request message cid=0x73e59dcf data=[000203
039e4c9da756c7542543f50497b6243137c0000]

v2f.py /home/scateu/.v2f

Got an event from some U2F relying party!

A website is asking you to login with the U2F authenticator
and it is claiming itself to be APPID with SHA256(APPID) =
55673b5138cc90d3b7f32bfdad6a38a8edd7b355b77ab9792196f106d16

Enter yes to login: yes

b'\x13\xc5\x93\xe0O\xe9\x8d\xd4\xc9k\x0e96\xc4\xce>\xd8\x86
U2FHID< send MSG response message cid=0x73e59dcf data=[0100
/dev/uhid <= UHID_INPUT2 size=64 data=[73e59dcf83004d010000
/dev/uhid <= UHID_INPUT2 size=64 data=[73e59dcf004b54c2d241
/dev/uhid => UHID_CLOSE
^C
A SIGINT (CTRL-C) signal is detected

v2f (V2F_DIR=/home/scateu/.v2f PID=7828) ended
scateu@scateu-ThinkPad-X230:~/dev/u2f/v2f.py$ vi u2fraw.py
scateu@scateu-ThinkPad-X230:~/dev/u2f/v2f.py$ python3 v2f.py
v2f (V2F_DIR=/home/scateu/.v2f PID=8046) started
/dev/uhid <= UHID_CREATE2 name=[] phys=[] uniq=[] rd_size=3
/dev/uhid => UHID_START dev_flags=0
/dev/uhid => UHID_OPEN
/dev/uhid => UHID_OUTPUT data=[00ffffffff860008340e05304780
U2FHID> got INIT request message cid=0xffffffff nonce=[340e
U2FHID> generate/allocate a new channel id: 0xb7684fdd
U2FHID< send INIT response message cid=0xffffffff data=[340
/dev/uhid <= UHID_INPUT2 size=64 data=[ffffffff860011340e05
/dev/uhid => UHID_OUTPUT data=[00b7684fdd830007000300000000
U2FHID> got MSG request message cid=0xb7684fdd data=[000300
U2FHID< send MSG response message cid=0xb7684fdd data=[5532
/dev/uhid <= UHID_INPUT2 size=64 data=[b7684fdd83000855324
/dev/uhid => UHID_OUTPUT data=[00b7684fdd83006e000203000000
/dev/uhid => UHID_OUTPUT data=[00b7684fdd00b355b77ab979219
U2FHID> got MSG request message cid=0xb7684fdd data=[000203
039e4c9da756c7542543f50497b6243137c0000]

v2f.py /home/scateu/.v2f

Got an event from some U2F relying party!

A website is asking you to login with the U2F authenticator
and it is claiming itself to be APPID with SHA256(APPID) =
55673b5138cc90d3b7f32bfdad6a38a8edd7b355b77ab9792196f106d16

Enter yes to login: yes

U2FHID< send MSG response message cid=0xb7684fdd data=[0100
/dev/uhid <= UHID_INPUT2 size=64 data=[b7684fdd83004e010000
/dev/uhid <= UHID_INPUT2 size=64 data=[b7684fdd00fac0951684
/dev/uhid => UHID_CLOSE
```

Authentication successful!

You have successfully authenticated using your U2F device.

Click to view more information about the performed transaction

The following data shows information about the authentication that just took place. After supplying your username and password, the server created a challenge for your previously registered U2F device. This challenge data was then presented to your device, which in turn, responded to the challenge. Finally, the server validated this response, approving the authentication.

Hover over the values for additional information.

Login Data
username: a
password: a

Challenge Data
version: U2F_V2
challenge: -NGXLl3Awmjvj6TJ84SBzZgEzHEXmiQdnQqRqaKHcug
keyHandle: cY2BxtmLWzEf8TGAKgoRkUI1oDnkydp1bHVCVD9QSXtiQxN8

Response Data
clientData: {"typ":"navigator.id.getAssertion","challenge":"-NGXLl3Awmjvj6TJ84SBzZgEzHEXmiQdnQqRqaKHcug","origin":"https://demo.yubico.com","cid_pubkey":"unused"}
signatureData: AQAAAAowRQIgcu6GMgxJkSq8YrC5LmSVzCO4iZOCbXlMT-LaoGUD7VkCIQDrxGghpxZMq1pE_7xC-sCVFoSDS8PTQZYYxSr7tlsg0w

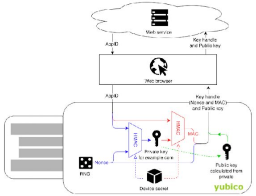
Authentication Parameters
touch: true
counter: 10

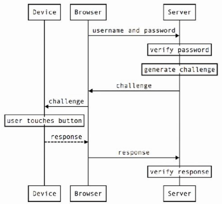
通用型密码学两步验证U盾 U2F 的克隆钓鱼攻击

U2F(Universal 2 Factor) 通用双因子标准由 Yubico 和 Google 发起的 FIDO (Fast IDentity Online) 联盟推出

U2F支持的平台

U2F标准旨在提供一个方便的免驱动、通用型的密码认证令牌
期望能在让用户在有U2F认证的情况下
即使只用短密码, 也能实现高强度的认证.
不依赖中心服务器, 完全基于公私钥/PKI体系.
免驱动，即插即用，Chrome浏览器原生支持





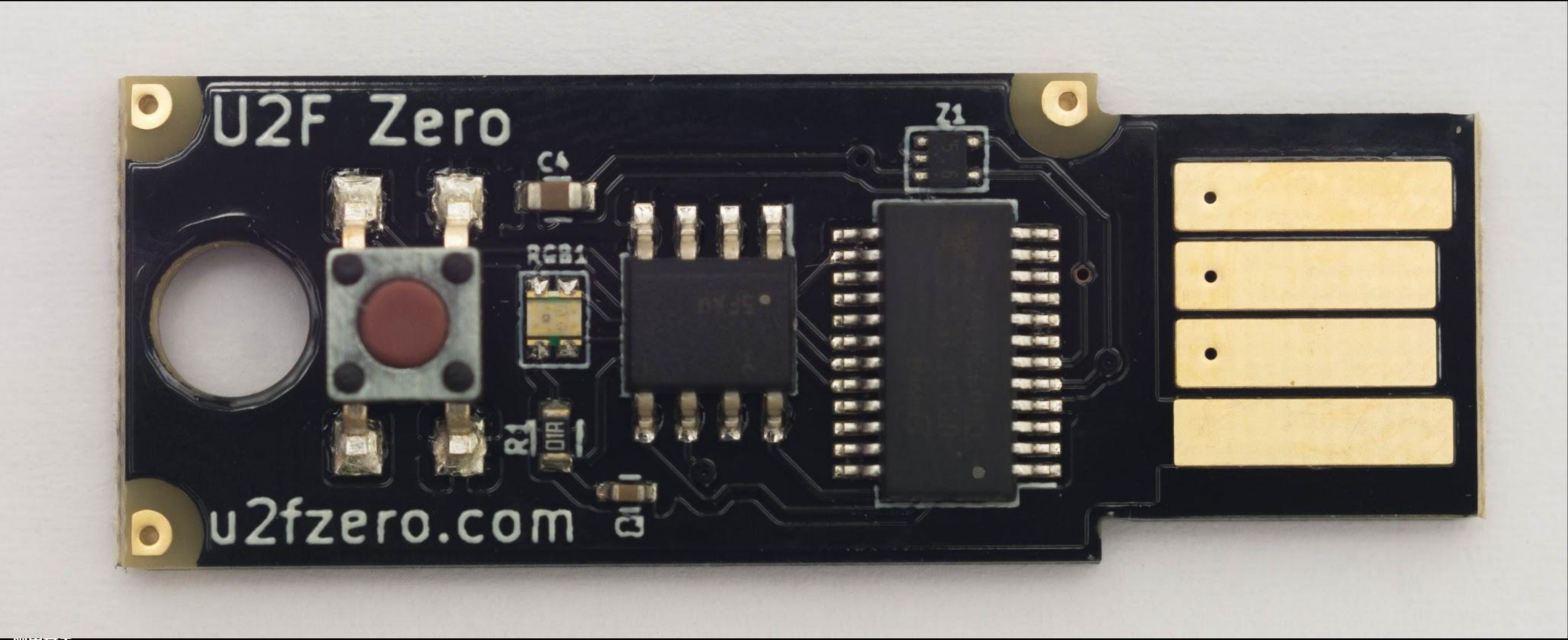
U2F标准中的关于Key Wrapping的机制的引入，造成了安全风险。即使在U盾使用了Secure Element的情况下，也可能被克隆，从而导致双因子认证被攻破。

攻击者在U2F令牌的初始化过程中，将主密钥提取，并绕过服务商的克隆计数器检测，即可攻破受害目标的两步验证措施。

阿里安全研究团队首先发现并给出了攻击示例，并提出了检测及缓解措施。并对市面上使用了U2F认证的服务提供商进行了检测，向受影响的厂商提供了漏洞检测报告。

这起安全事件进一步提醒业界对供应链全周期安全可信的关注。
阿里安全举办的“功守道”阿里软件供应链安全大赛今年正在进行,
“以武会友、攻守切磋”，以促进软件供应链安全技术的发展。

U2F Zero



Key Wrapping

- Secure Element:
    - Public / Private Key Pair
    - On-chip operation: generation, signing
        - Import Key
    - Limited Storage
- Solution:
    - Device Secret
    - Key Derivation

Key Wrapping Mechanism

https://fidoalliance.org/specs/fido-u2f-v1. 0-nfc-bt-amendment-20150514/fido-u2f-overview.html# allowing-for-inexpensive-u2f-devices,

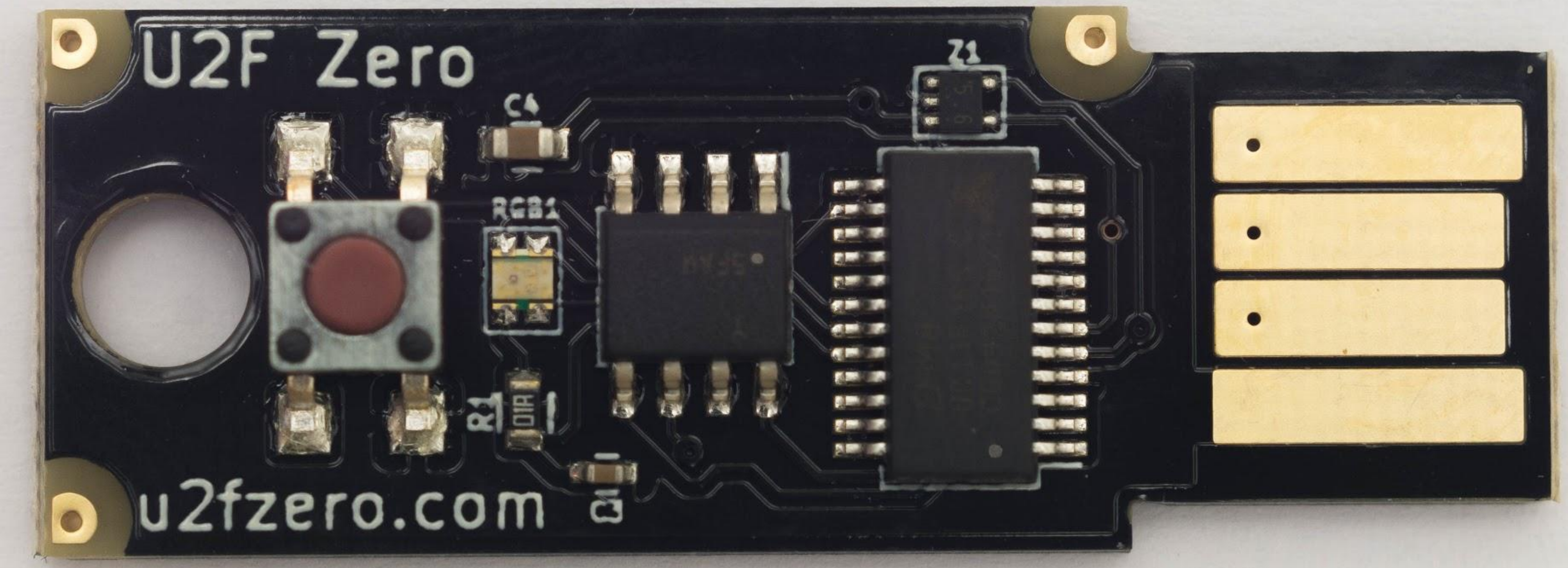
7. Allowing for Inexpensive U2F Devices
A key goal of this program is to enable extremely inexpensive yet secure devices. To enable new secure element chips to be as inexpensive as possible it is important to allow them to have minimal or no onboard memory.

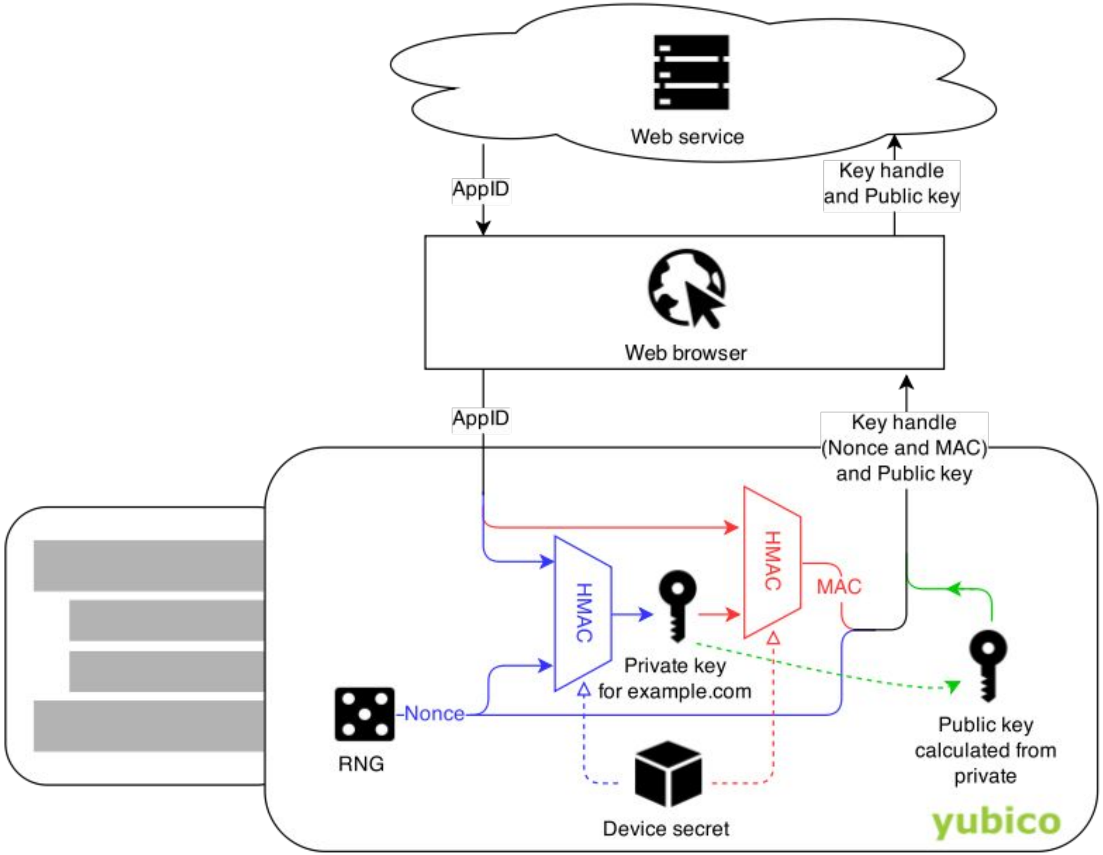
A U2F device allows for this. The Key Handle issued by the U2F device does not have to be an index to the private key stored on board the U2F device secure element chip. Instead, the Key Handle can ’ store’ (i.e., contain) the private key for the origin and the hash of the origin encrypted with a ’wrapping’ key known only to the U2F device secure element. When the Key Handle goes back to the secure element it ’unwraps’ it to ’retrieve’ the private key and the origin that it was generated for. As another alternative, the U2F device could store this ’wrapped’ information in a table in off-chip memory outside the secure element (which is presumably cheaper). This memory is still on board the U2F device. In this case, the Key Handle sent to the origin would be an index into this table in off-chip memory. As another possibility in the design spectrum, the Key Handle might only encode the origin and an index number, while the private key might still be kept on board -- this would, of course, imply the number of keys is limited by the amount of memory.

U2F Zero

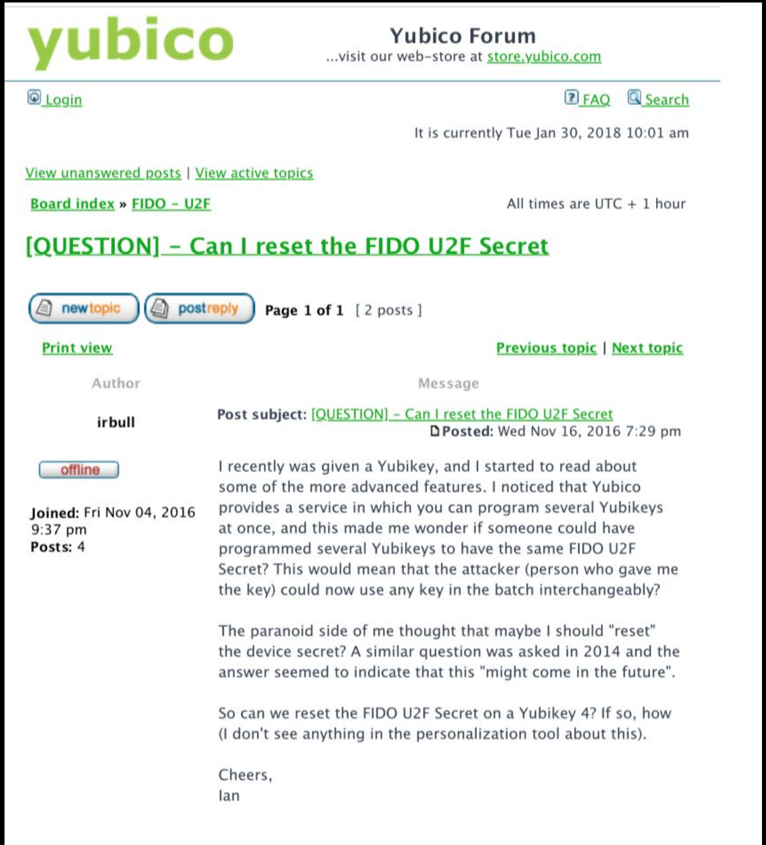
PrivateKey = HMAC(AppID+nonce,DeviceKey)

KeyHandle = nonce, HMAC(AppID,PrivateKey)




Web service
AppID
Key handle
and Public key
Web browser
AppID
Key handle
(Nonce and MAC)
and Public key
HMAC
HMAC
MAC
Private key
for example.com
Nonce
RNG
Device secret
Public key
calculated from
private
yubico

Should be worried?



Yubico Forum
...visit our web-store at store.yubico.com

It is currently Tue Jan 30, 2018 10:01 am

View unanswered posts | View active topics

Board index » FIDO - U2F

All times are UTC + 1 hour

[QUESTION] - Can I reset the FIDO U2F Secret

Page 1 of 1 [2 posts]

Print view

Previous topic | Next topic

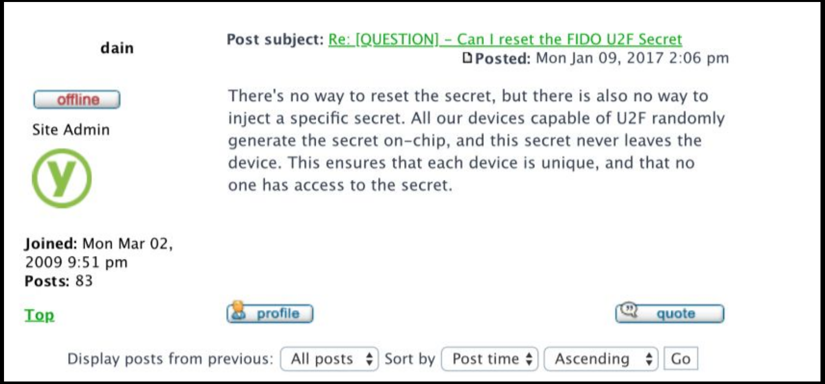
Author	Message
irbull offline **Joined:** Fri Nov 04, 2016 9:37 pm **Posts:** 4	**Post subject:** [QUESTION] - Can I reset the FIDO U2F Secret **Posted:** Wed Nov 16, 2016 7:29 pm

I recently was given a Yubikey, and I started to read about some of the more advanced features. I noticed that Yubico provides a service in which you can program several Yubikeys at once, and this made me wonder if someone could have programmed several Yubikeys to have the same FIDO U2F Secret? This would mean that the attacker (person who gave me the key) could now use any key in the batch interchangeably?

The paranoid side of me thought that maybe I should "reset" the device secret? A similar question was asked in 2014 and the answer seemed to indicate that this "might come in the future".

So can we reset the FIDO U2F Secret on a Yubikey 4? If so, how (I don't see anything in the personalization tool about this).

Cheers,
Ian



dain

Site Admin

Joined: Mon Mar 02, 2009 9:51 pm
Posts: 83

Post subject: Re: [QUESTION] - Can I reset the FIDO U2F Secret
Posted: Mon Jan 09, 2017 2:06 pm

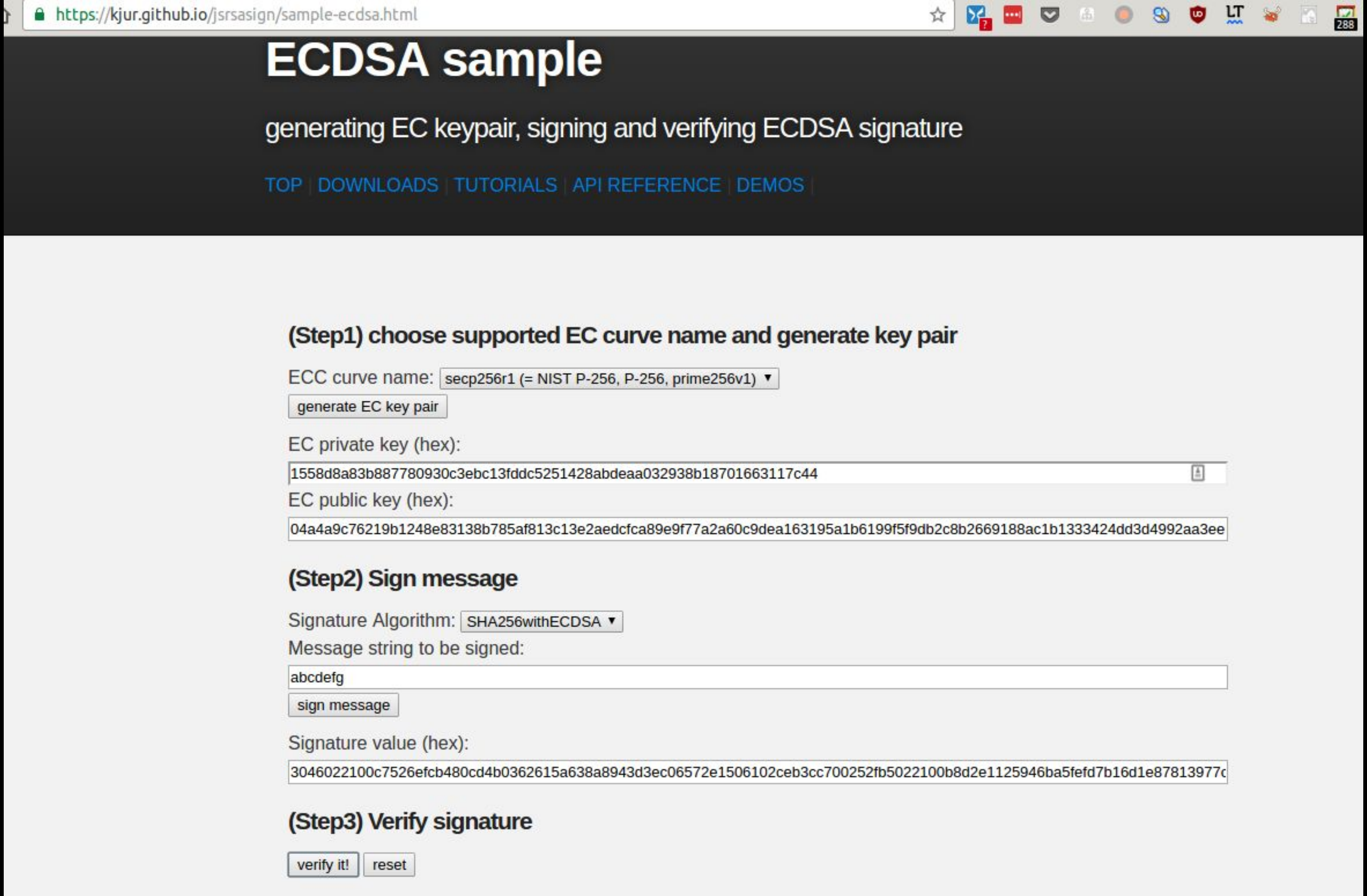
There's no way to reset the secret, but there is also no way to inject a specific secret. All our devices capable of U2F randomly generate the secret on-chip, and this secret never leaves the device. This ensures that each device is unique, and that no one has access to the secret.

Top

Display posts from previous: All posts Sort by Post time Ascending Go

Proposed Attack Scenario

1. The attacker extracts the master key during manufacturing process of U2F keys, which in my case is open- source U2F Zero.
2. Attacker clone this U2F key. (In this case, we integrated it with a software U2F implementation.)
3. Attacker gives this U2F key to a victim.
4. Assume the victim use this U2F key to register with Google.
5. Attacker gets to know the password from another source. (such as social engineering, or other ways of password phishing)
6. Login.



https://kjur.github.io/jsrsasign/sample-ecdsa.html

ECDSA sample

generating EC keypair, signing and verifying ECDSA signature

(Step1) choose supported EC curve name and generate key pair

ECC curve name: secp256r1 (= NIST P-256, P-256, prime256v1) ▼

EC private key (hex):

1558d8a83b887780930c3ebc13fddc5251428abdeaa032938b18701663117c44

EC public key (hex):

04a4a9c76219b1248e83138b785af813c13e2aedcfca89e9f77a2a60c9dea163195a1b6199f5f9db2c8b2669188ac1b1333424dd3d4992aa3ee

(Step2) Sign message

Signature Algorithm: SHA256withECDSA ▼

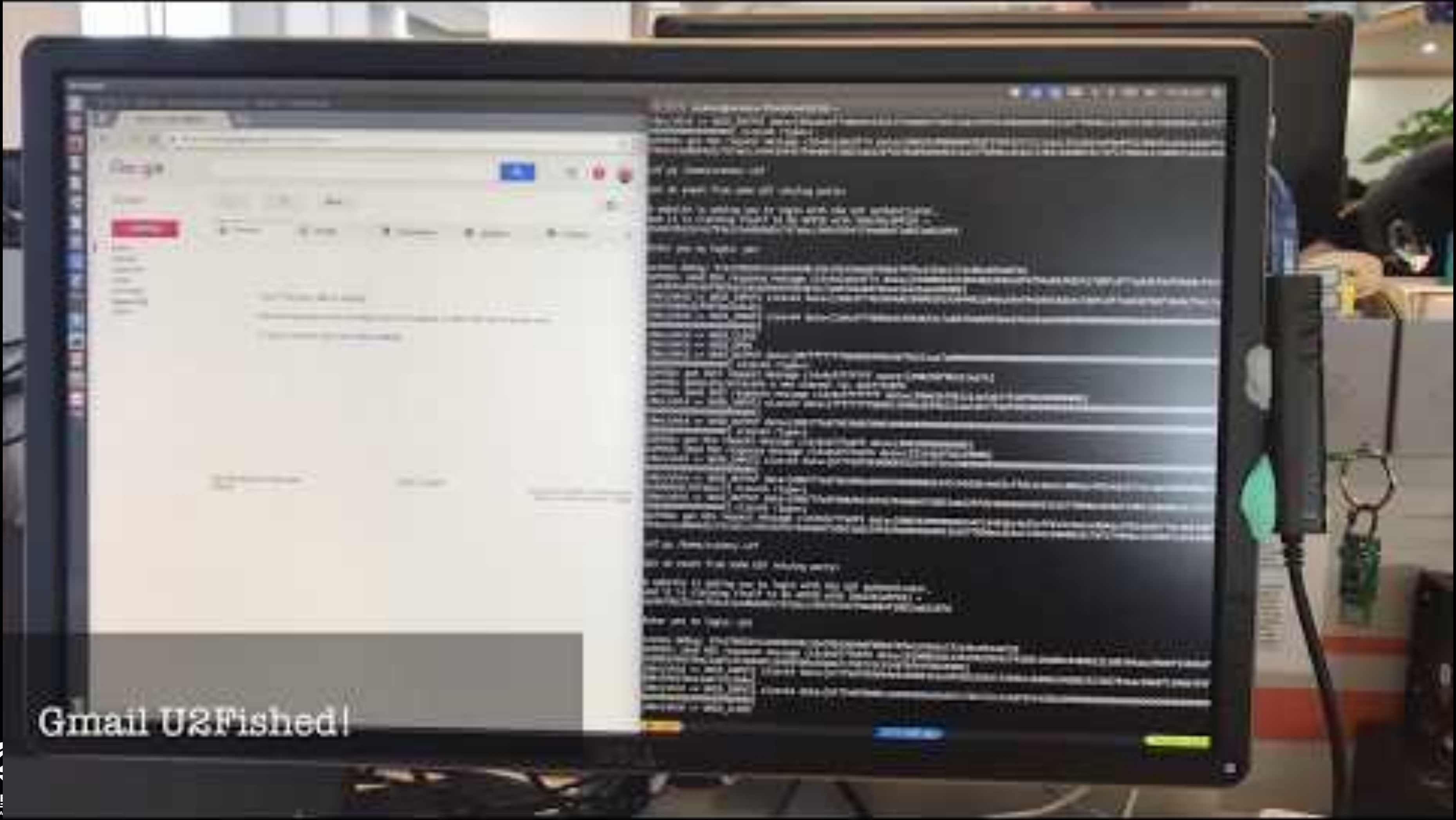
Message string to be signed:

abcdefg

Signature value (hex):

3046022100c7526efcb480cd4b0362615a638a8943d3ec06572e1506102ceb3cc700252fb5022100b8d2e1125946ba5fefd7b16d1e87813977c

(Step3) Verify signature



Gmail U2Fished!

Anti-clone Counter

- Inside Secure Element:
    - High-Endurance Monotonic Counters.
- Counter: 100
- Counter: 101
- Attacker:
    - Large: 900
        - Victim: Press 801 times
    - The best try: 102
        - Attacker: 1,2,3,.....,100,101,102
    - Multi services share a same counter
    - Counter overflow?

Key Findings

- Security Model
    - Ultimate Trust Root:
        - Traditional Dedicated USB Security Key
        - General Purpose USB Security Key
    - should be downgraded to “Manufacturer Trust Level”
    - At least, key regeneration function should be provided
- Anti-clone counter should be well implemented
    - Google/Facebook: users are not aware when cloned
    - Fastmail: didn’t check at all.
        - Reported, Confirmed

Mitigation

(Service Provider Side)

1. Trust Level Downgrade
2. Clone Detection should be well implemented:
   a. User aware
   b. Revoke

Conclusion

- Supply Chain Risk
- We give a real-world example of this kind of attack.
- We found that anti-clone mechanism is not well implemented in some websites.

Acknowledgement

- shankerwangmiao aronxu cqtest nsblink gaoyichuan dotkrnl bigeagle hexchain harrychen leoyaboo dangfan ppwwyyxx xavieryao mianmoe
- TUNA: Tsinghua University tuNa Association

Future Work

- FIDO2
- A phishing website trying to extract master secret, reversing HMAC function.

Wang Kang

3@14159265358979323846264338327950 28.com

scateu@gmail.com

wangkang.wk@alibaba-inc.com