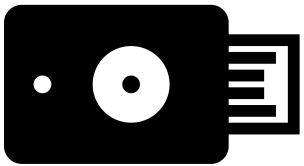
Making U2F Resistant to Implementation Bugs and Supply-Chain Tampering

Emma Dauterman, Henry Corrigan-Gibbs, David Mazières, Dan Boneh, Dominic Rizzo

Stanford and Google

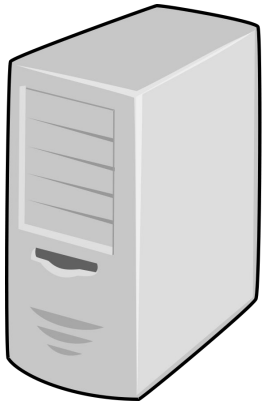
To appear at IEEE Security & Privacy 2019

U2F defends against phishing and browser compromise



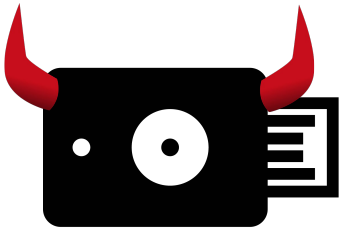
Token
(authenticator)

Host



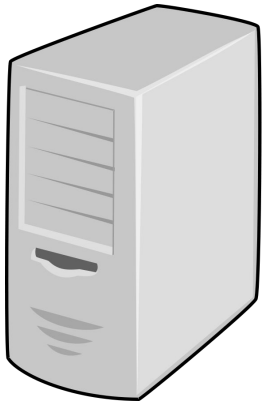
Relying Party

… but what about vulnerabilities in the token itself?



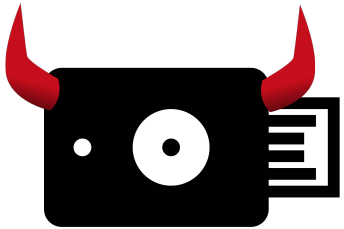
Token
(authenticator)

Host



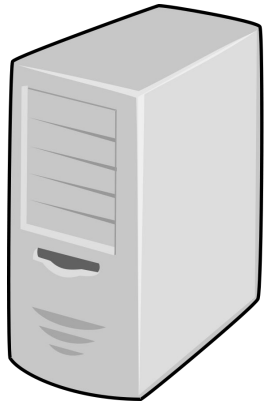
Relying Party

… but what about vulnerabilities in the token itself?



Token
(authenticator)

Host



Relying Party

1. Implementation bugs
2. Supply-chain tampering

U2F Shortcoming #1: Implementation bugs

ars TECHNICA SUBSCRIBE SIGN IN

COMPLETELY BROKEN —

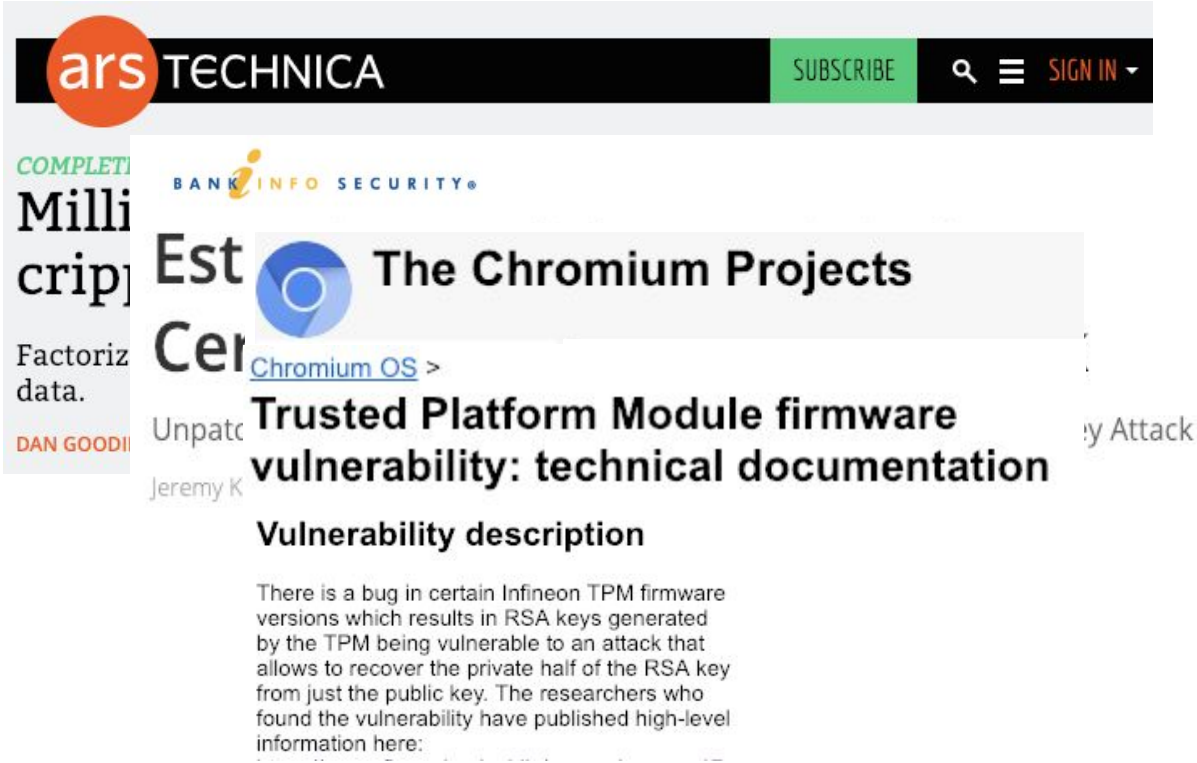
Millions of high-security crypto keys crippled by newly discovered flaw

Factorization weakness lets attackers impersonate key holders and decrypt their data.

DAN GOODIN - 10/16/2017, 4:00 AM

U2F Shortcoming #1: Implementation bugs

U2F Shortcoming #1: Implementation bugs



U2F Shortcoming #2: Supply-chain tampering

ars TECHNICA SUBSCRIBE SIGN IN

AT LEAST THEY PICK UP THE EXTRA SHIPPING —

Photos of an NSA "upgrade" factory show Cisco router getting implant

Servers, routers get "beacons" implanted at secret locations by NSA's TAO team.

SEAN GALLAGHER - 5/14/2014, 12:30 PM

(TS//SI//NF) Left: Intercepted packages are opened carefully; Right: A "load station" implants a beacon

MOTHERBOARD

CHINA | By Joseph Cox | Aug 31 2018, 5:05am

Experts Call for Transparency Around Google's Chinese-Made Security Keys

Google's Titan Security Keys, used to lock down accounts, are produced in China. Several experts want more answers on that supply chain process, for fears of tampering or security issues.

Our proposed enhancement of U2F

Goals:

- Augment U2F to protect against **faulty tokens**
- **Backwards-compatible** with U2F relying parties
- **Practical** on commodity hardware tokens

Our proposed enhancement of U2F

Goals:

- Augment U2F to protect against **faulty tokens**
- **Backwards-compatible** with U2F relying parties
- **Practical** on commodity hardware tokens

Design principles:

- Both host and token **contribute randomness** to the protocol.
- **Host can verify** all deterministic token operations.

U2F protocol steps

1. Registration (associating a token with an account)
2. Authentication (logging into an account)

Proposed protocol steps

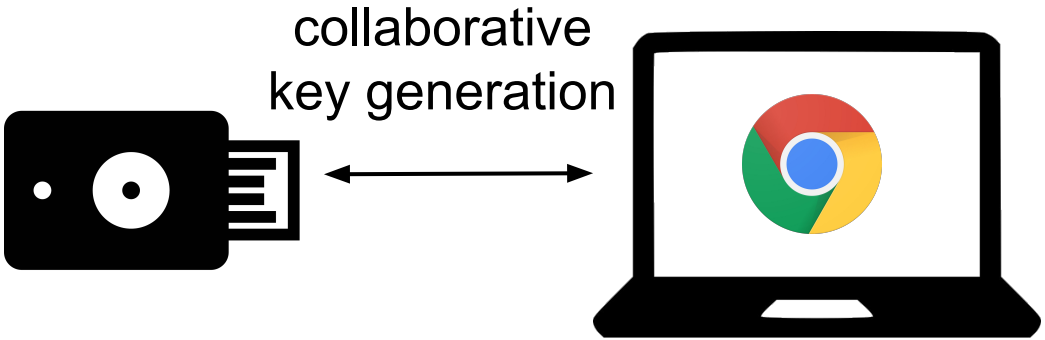
0. Initialization (after purchasing a token)
1. Registration (associating a token with an account)
2. Authentication (logging into an account)

Proposed protocol steps

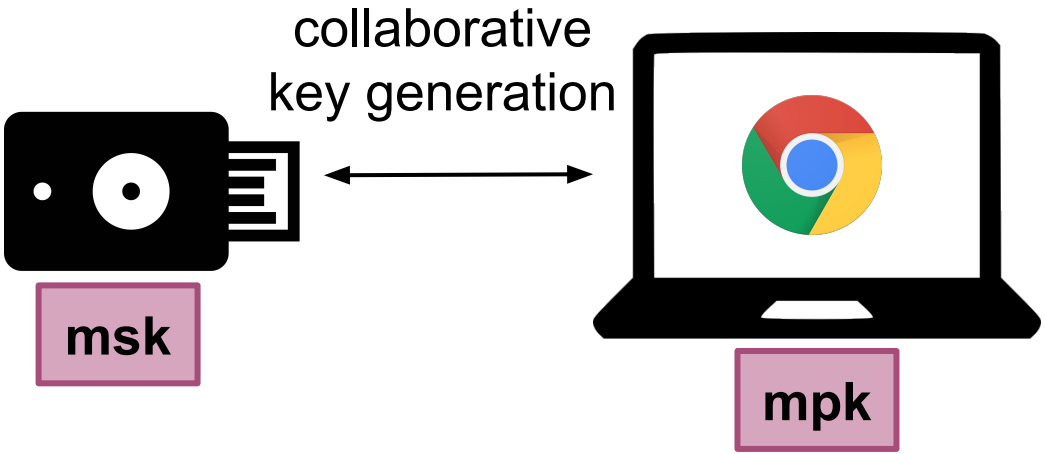
0. **Initialization (after purchasing a token)**
1. Registration (associating a token with an account)
2. Authentication (logging into an account)

Principle: Both host and token contribute randomness to the protocol.

Step #0: Initialization



Step #0: Initialization



Initialization properties

The token cannot bias mpk.

msk

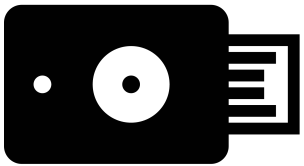
mpk

[CMBF13]

Initialization properties

The token cannot bias mpk.

The host learns nothing about msk.



msk

mpk

[CMBF13]

Initialization protocol using collaborative key generation

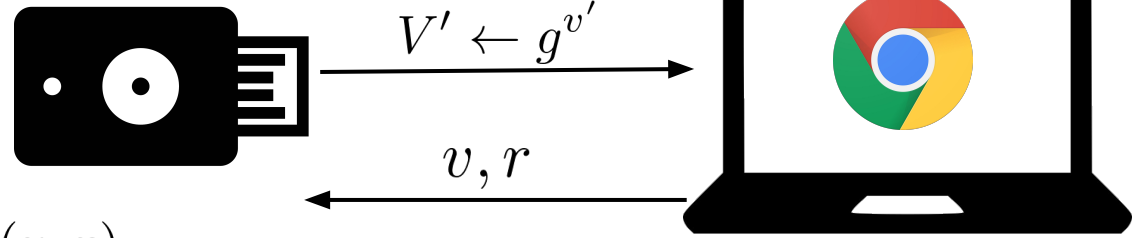
$\mathbb{G} = \langle g \rangle$ is a group of prime order q.
ECDSA keypairs have the form $(x, g^x) \in \mathbb{Z}_q \times \mathbb{G}$.

$r \xleftarrow{R} \mathbb{Z}_q$
$v \xleftarrow{R} \mathbb{Z}_q$

$c \leftarrow H(v, r)$

$v' \xleftarrow{R} \mathbb{Z}_q$

$V' \leftarrow g^{v'}$



v, r

$\text{mpk} \leftarrow V' \cdot g^v$
$= g^{v+v'}$

check $c = H(v, r)$

$\text{msk} \leftarrow v + v'$

Initialization protocol using collaborative key generation

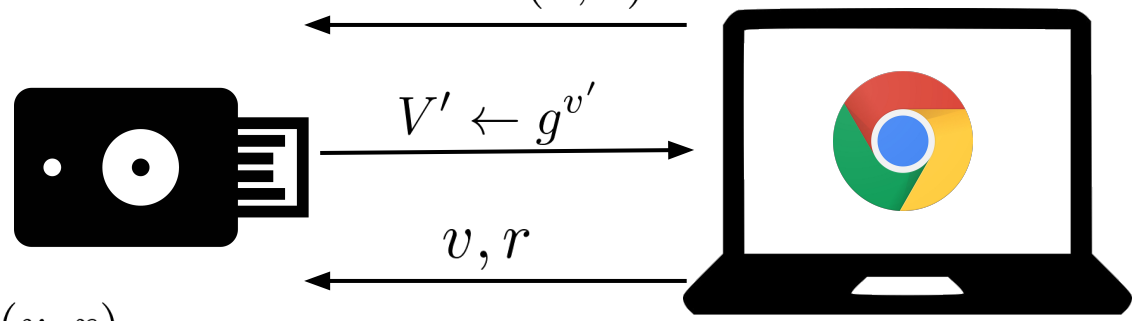
$\mathbb{G} = \langle g \rangle$ is a group of prime order q.
ECDSA keypairs have the form $(x, g^x) \in \mathbb{Z}_q \times \mathbb{G}$.

$r \xleftarrow{R} \mathbb{Z}_q$
$v \xleftarrow{R} \mathbb{Z}_q$

$c \leftarrow H(v, r)$

$v' \xleftarrow{R} \mathbb{Z}_q$

$V' \leftarrow g^{v'}$



v, r

$\text{mpk} \leftarrow V' \cdot g^v$
$= g^{v+v'}$

check $c = H(v, r)$

$\text{msk} \leftarrow v + v'$

✓ The token cannot bias mpk.

Initialization protocol using collaborative key generation

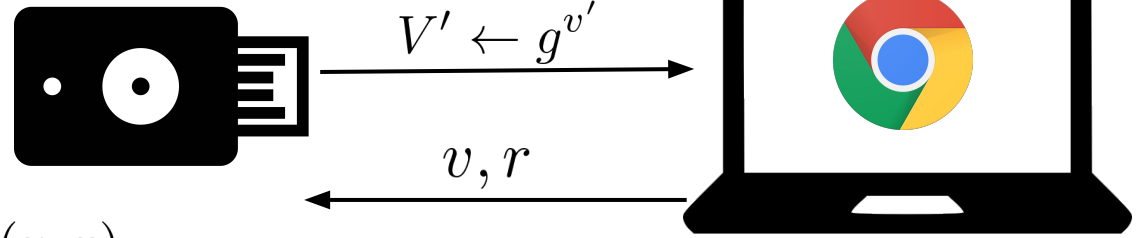
$\mathbb{G} = \langle g \rangle$ is a group of prime order q.
ECDSA keypairs have the form $(x, g^x) \in \mathbb{Z}_q \times \mathbb{G}$.

$r \xleftarrow{R} \mathbb{Z}_q$
$v \xleftarrow{R} \mathbb{Z}_q$

$c \leftarrow H(v, r)$

$v' \xleftarrow{R} \mathbb{Z}_q$

$V' \leftarrow g^{v'}$



v, r

$\text{mpk} \leftarrow V' \cdot g^v$
$= g^{v+v'}$

check $c = H(v, r)$

$\text{msk} \leftarrow v + v'$

✓ The host learns nothing about msk.

Proposed protocol steps

✓0. Initialization (after purchasing a token)

1. Registration (associating a token with an account)
2. Authentication (logging into an account)

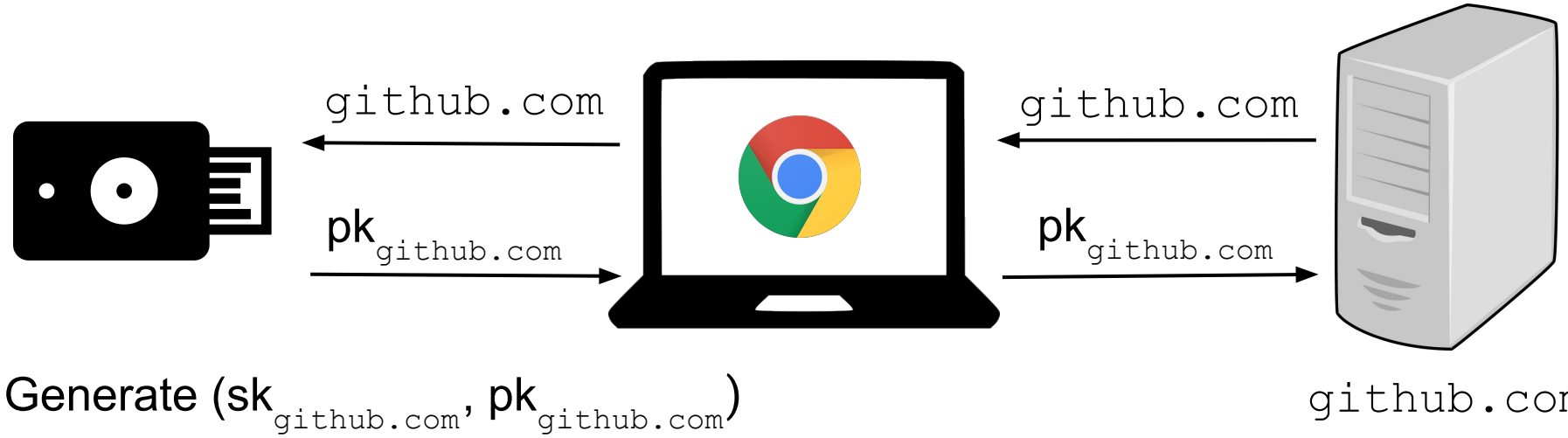
Proposed protocol steps

✓0. Initialization (after purchasing a token)

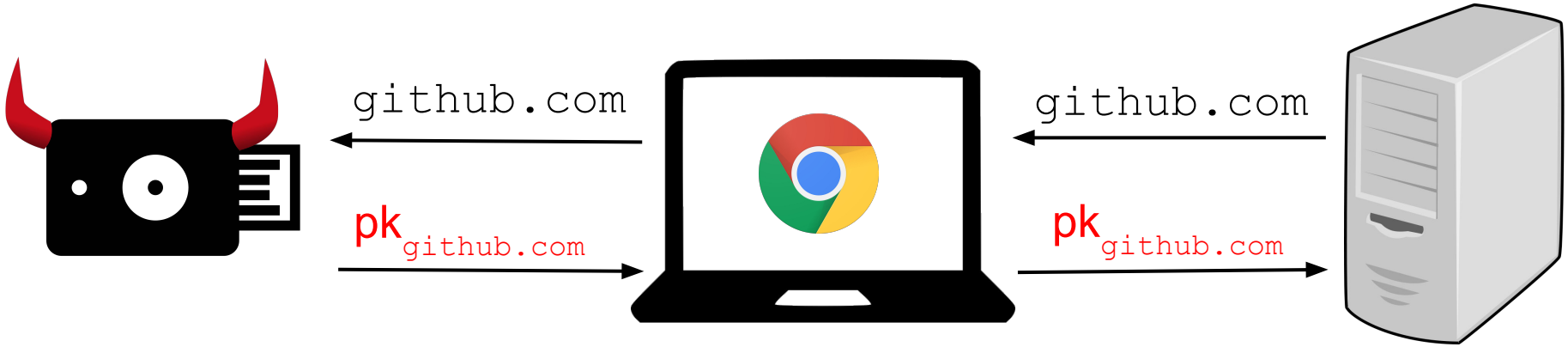
1. **Registration (associating a token with an account)**
2. Authentication (logging into an account)

Principle: All deterministic token operations can be verified by the host.

U2F Registration

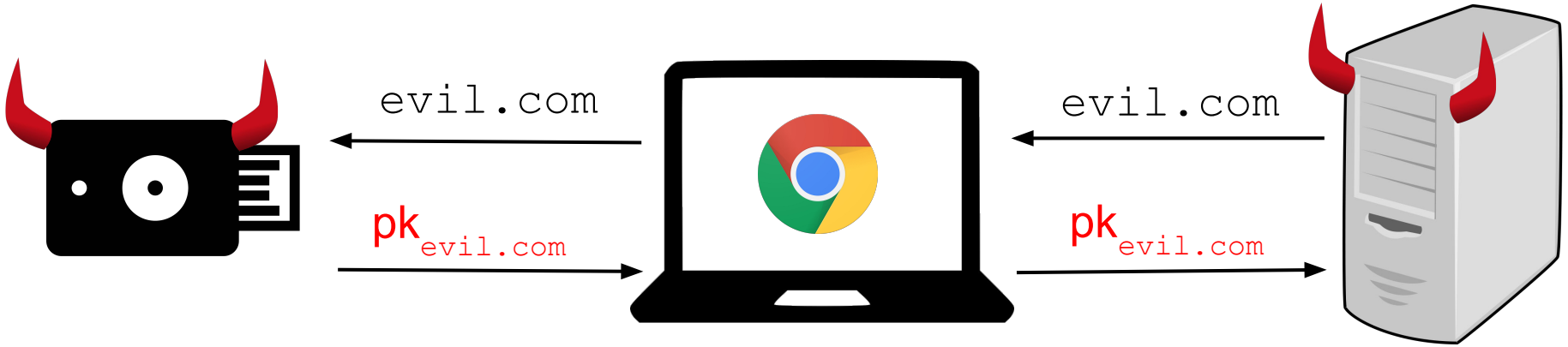


Implementation bug at registration

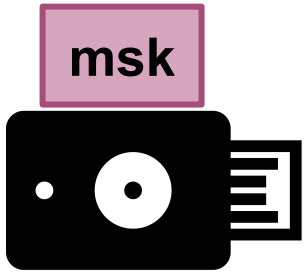


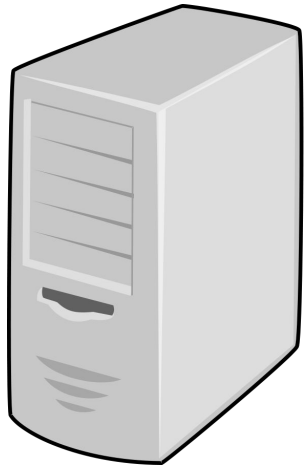
Generate ($sk_{github.com}$, $pk_{github.com}$)
using weak randomness

Supply-chain tampering at registration



Proposed Registration





Derive per-account keypairs in a **deterministic** and **verifiable** way from the master keypair.

Registration properties

Registration properties

1. **Unique:** The token can produce the unique keypair for `github.com`.

Registration properties

1. **Unique:** The token can produce the unique keypair for `github.com`.
2. **Verifiable:** The token can prove to the host that $pk_{\texttt{github.com}}$ is really the unique public key for `github.com`.

Registration properties

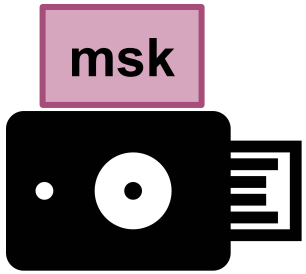
1. **Unique:** The token can produce the unique keypair for `github.com`.
2. **Verifiable:** The token can prove to the host that $pk_{\texttt{github.com}}$ is really the unique public key for `github.com`.
3. **Unlinkability:** `github.com` cannot distinguish $pk_{\texttt{github.com}}$ from a random ECDSA public key.

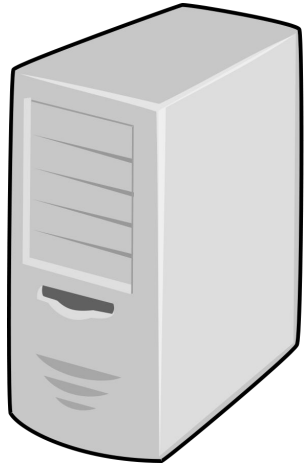
Registration properties

1. **Unique:** The token can produce the unique keypair for `github.com`.
2. **Verifiable:** The token can prove to the host that $pk_{\texttt{github.com}}$ is really the unique public key for `github.com`.
3. **Unlinkability:** `github.com` cannot distinguish $pk_{\texttt{github.com}}$ from a random ECDSA public key.
4. **Unforgeable:** The host cannot forge a signature under $pk_{\texttt{github.com}}$.

Registration construction

$\mathbb{G} = \langle g \rangle$ is a group of prime order q.



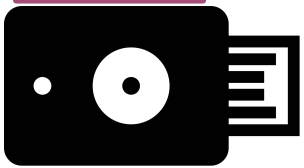


github.com

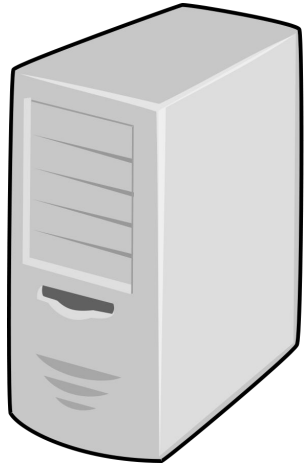
Registration construction

$\mathbb{G} = \langle g \rangle$ is a group of prime order q.

$x \in \mathbb{Z}_q$



$X = g^x \in \mathbb{G}$

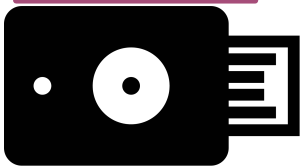


`github.com`

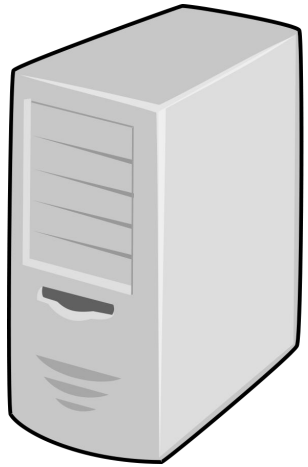
Registration construction

$\mathbb{G} = \langle g \rangle$ is a group of prime order q.

$x \in \mathbb{Z}_q$
$k = H(X)$

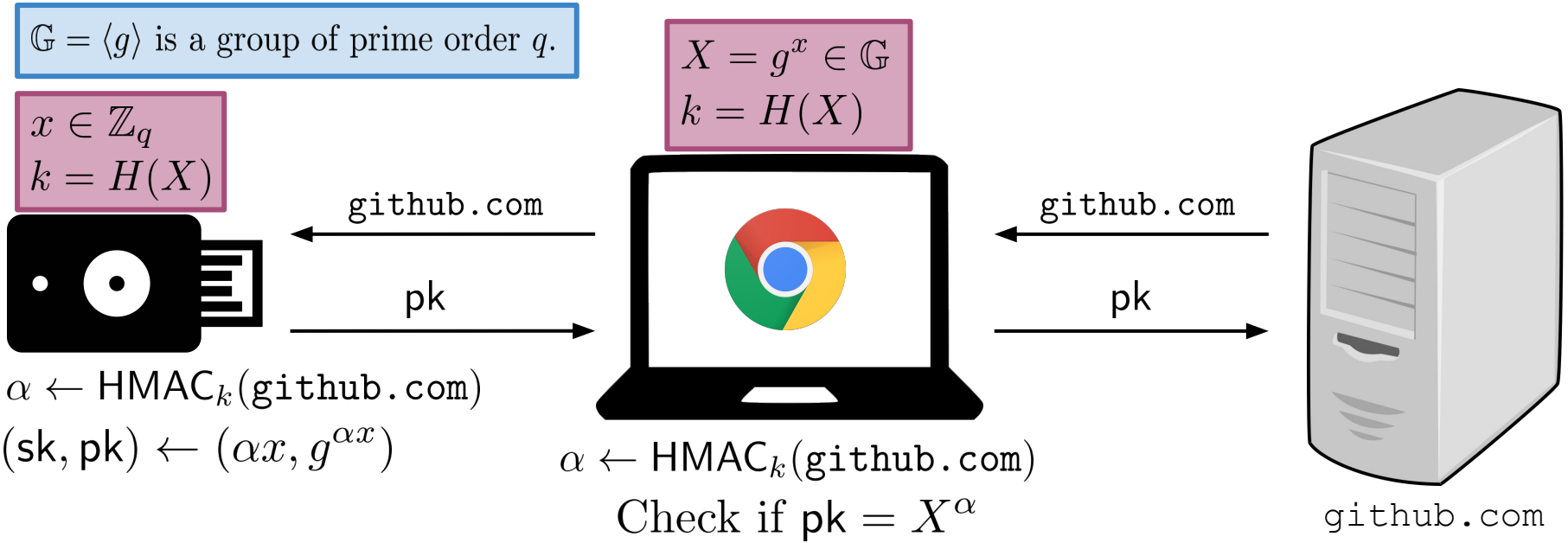


$X = g^x \in \mathbb{G}$
$k = H(X)$

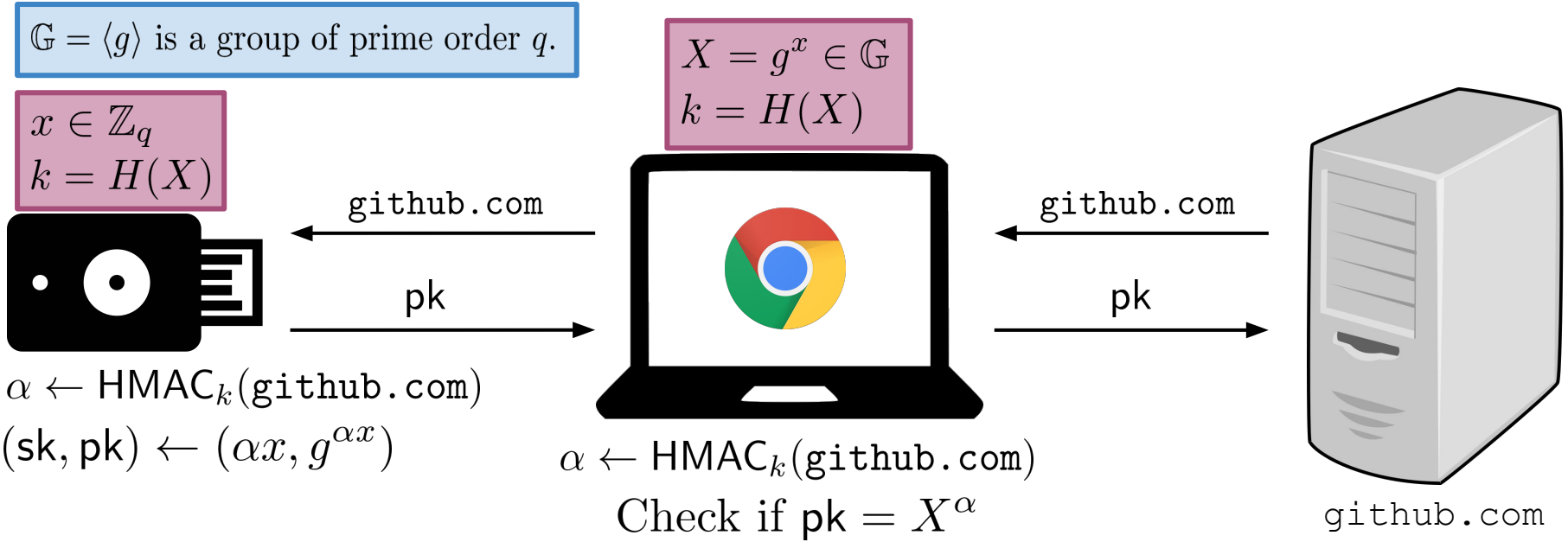


`github.com`

Registration construction



Registration construction



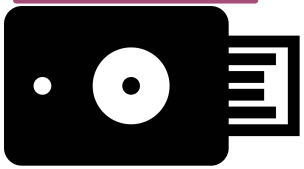
✓ **Unique:** The token can produce the unique keypair for `github.com`.

Registration construction

$\mathbb{G} = \langle g \rangle$ is a group of prime order q.

$x \in \mathbb{Z}_q$
$k = H(X)$

$X = g^x \in \mathbb{G}$
$k = H(X)$

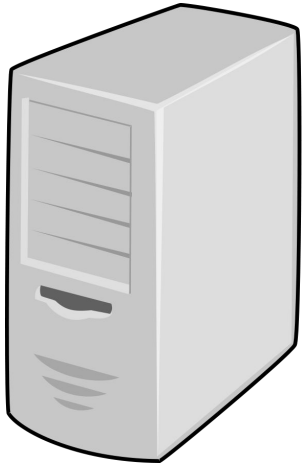


github.com

pk

github.com

pk



$\alpha \leftarrow \mathsf{HMAC}_k(\texttt{github.com})$
$(\mathsf{sk}, \mathsf{pk}) \leftarrow (\alpha x, g^{\alpha x})$

$\alpha \leftarrow \mathsf{HMAC}_k(\texttt{github.com})$
Check if $\mathsf{pk} = X^\alpha$

github.com

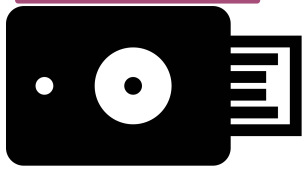
✓ **Verifiable:** The token can prove to the host that $\mathsf{pk}_{\texttt{github.com}}$ is really the unique public key for github.com.

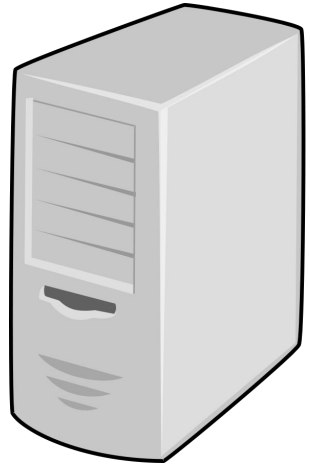
Registration construction

$\mathbb{G} = \langle g \rangle$ is a group of prime order q.

$x \in \mathbb{Z}_q$
$k = H(X)$

$X = g^x \in \mathbb{G}$
$k = H(X)$





github.com

pk

github.com

pk

$\alpha \leftarrow \mathsf{HMAC}_k(\texttt{github.com})$
$(\mathsf{sk}, \mathsf{pk}) \leftarrow (\alpha x, g^{\alpha x})$

$\alpha \leftarrow \mathsf{HMAC}_k(\texttt{github.com})$
Check if $\mathsf{pk} = X^{\alpha}$

github.com

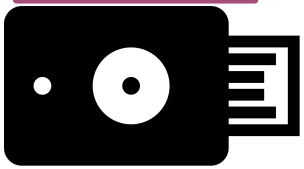
✓**Unlinkability:** github.com cannot distinguish $\mathsf{pk}_{\texttt{github.com}}$ from a random ECDSA public key.

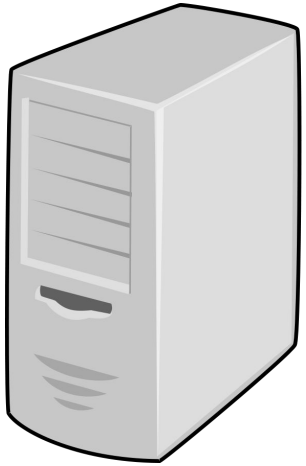
Registration construction

$\mathbb{G} = \langle g \rangle$ is a group of prime order q.

$x \in \mathbb{Z}_q$
$k = H(X)$

$X = g^x \in \mathbb{G}$
$k = H(X)$





github.com

pk

github.com

pk

$\alpha \leftarrow \mathsf{HMAC}_k(\texttt{github.com})$
$(\mathsf{sk}, \mathsf{pk}) \leftarrow (\alpha x, g^{\alpha x})$

$\alpha \leftarrow \mathsf{HMAC}_k(\texttt{github.com})$
Check if $\mathsf{pk} = X^\alpha$

github.com

✓**Unforgeable:** The host cannot forge a signature under $\mathsf{pk}_{\texttt{github.com}}$.

Proposed protocol steps

✓0. Initialization (after purchasing a token)

✓1. Registration (associating a token with an account)

2. Authentication (logging into an account)

Proposed protocol steps

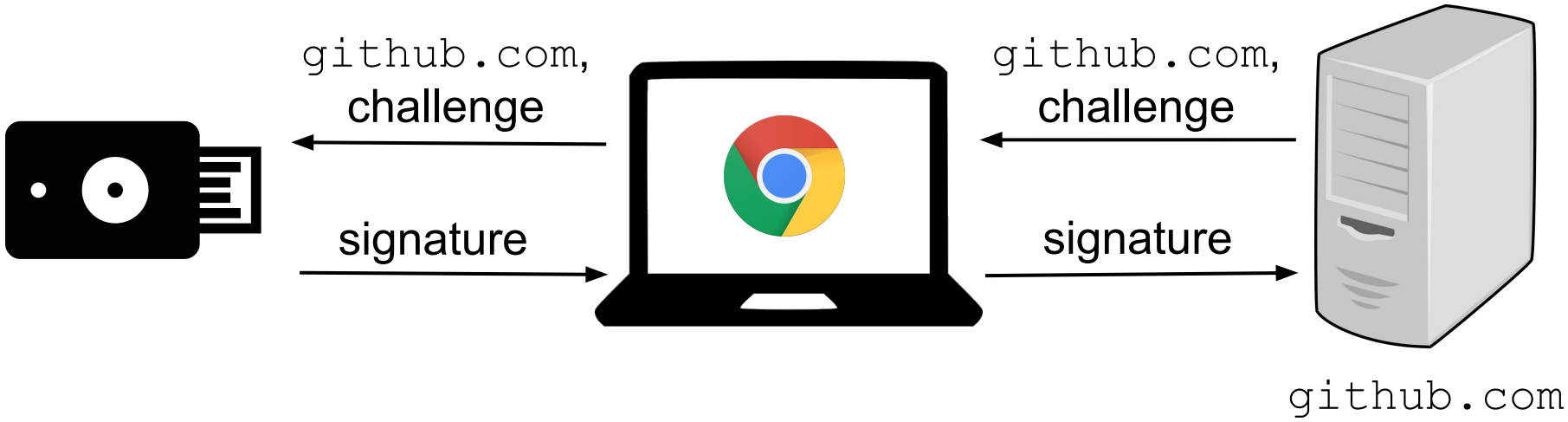
✓0. Initialization (after purchasing a token)

✓1. Registration (associating a token with an account)

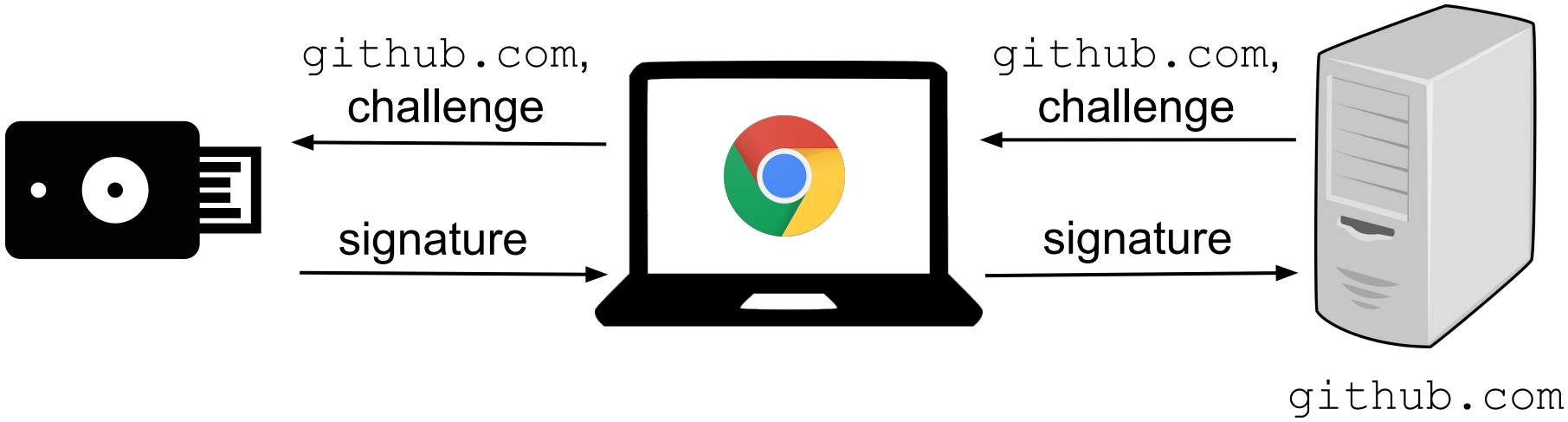
2. Authentication (logging into an account)

Principle: Both host and token contribute randomness to the protocol.

U2F Authentication



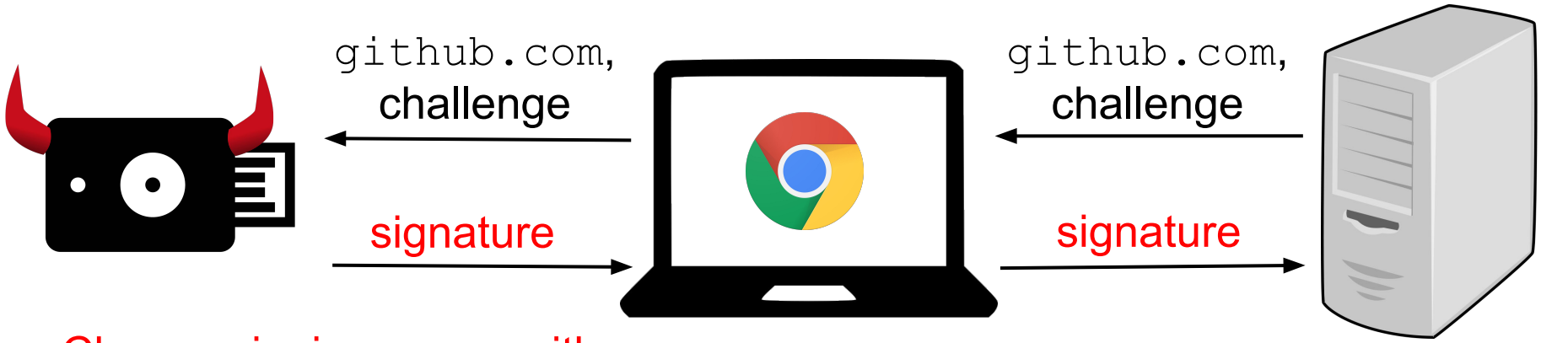
U2F Authentication



ECDSA signatures are randomized:

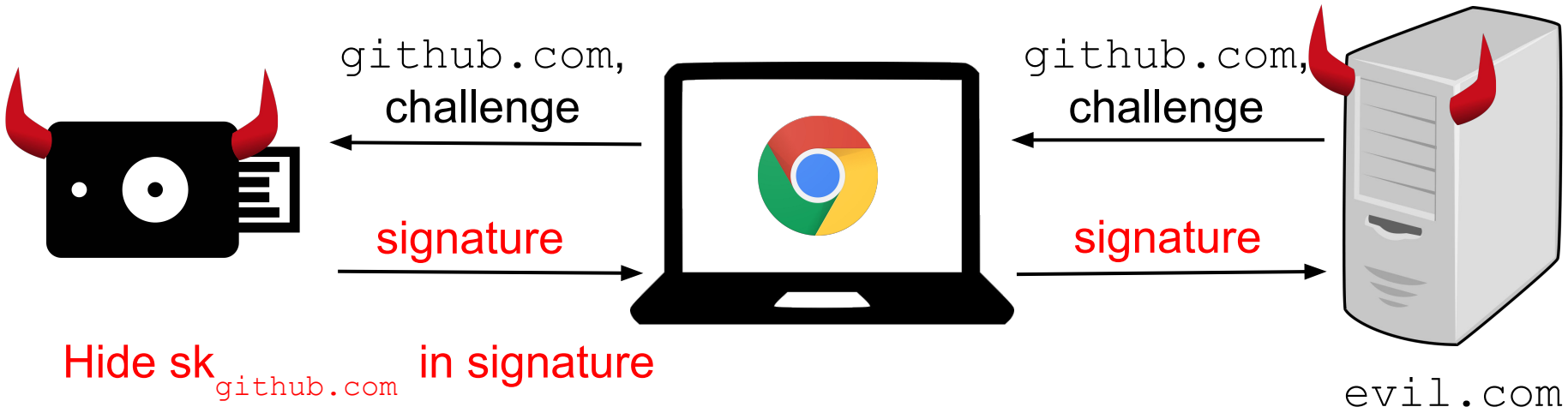
1. Signing nonce
2. Malleability (2 valid signatures)

Implementation bug at authentication



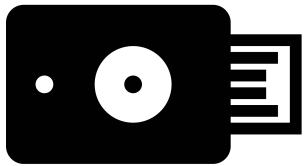
Choose signing nonce with weak randomness

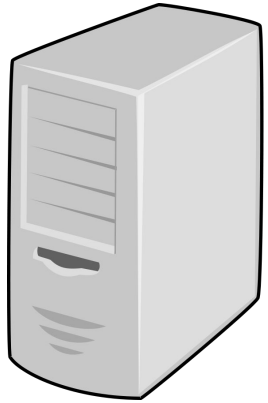
Supply-chain tampering at authentication



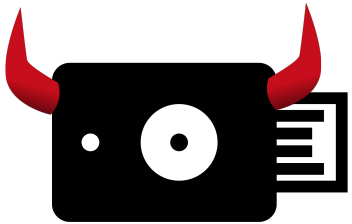
Subliminal channels: [Sim84], [Des88]
Unique signatures: [BLS04]

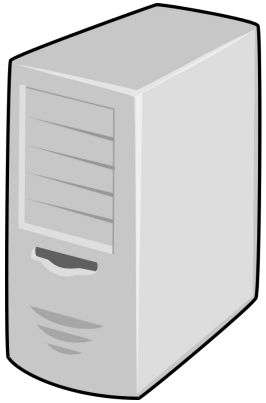
Authentication properties





Authentication properties

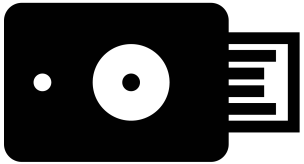


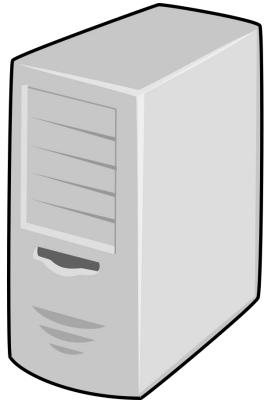


Exfiltration resistance: Token cannot exfiltrate any bits of information in signature.

[AMV15], [MS15], [DMS16]

Authentication properties



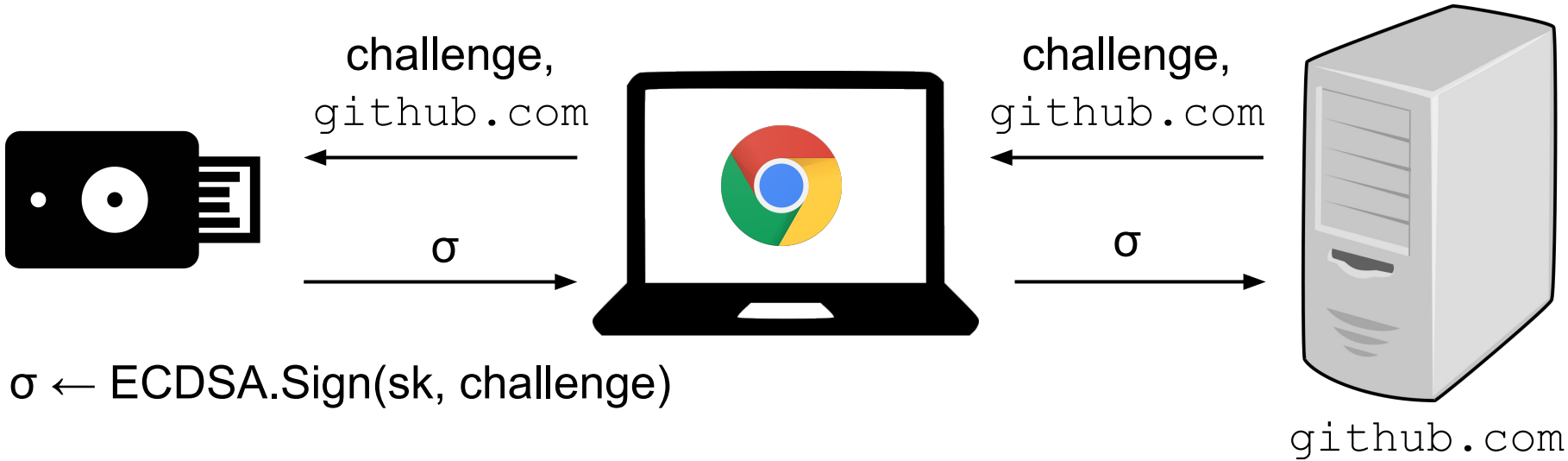


Exfiltration resistance: Token cannot exfiltrate any bits of information in signature.

Zero knowledge: Host “learns nothing” except a valid signature.

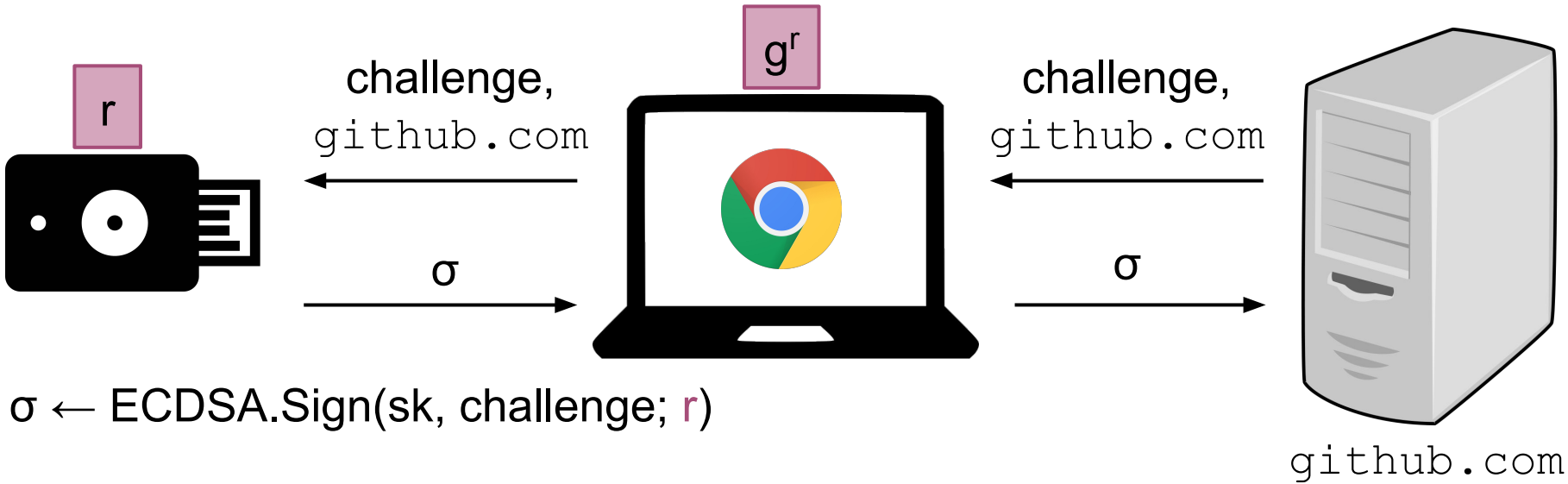
[AMV15], [MS15], [DMS16]

Authentication construction idea



σ ← ECDSA.Sign(sk, challenge)

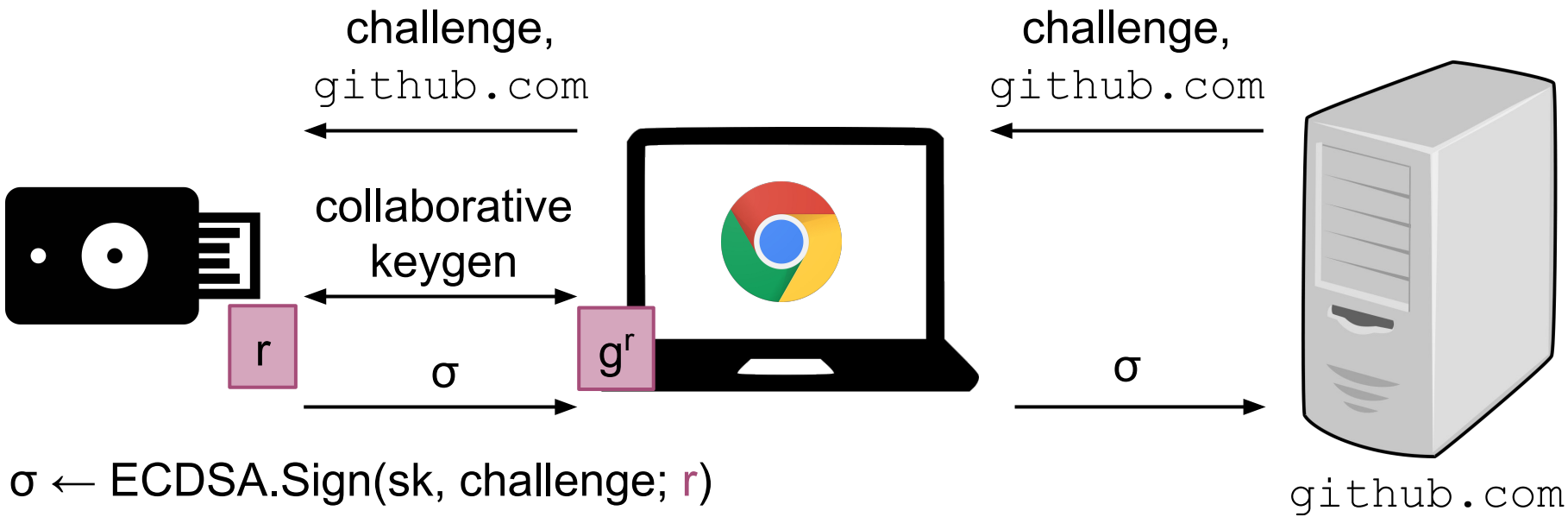
Authentication construction idea



σ ← ECDSA.Sign(sk, challenge; r)

Use g^r to verify that r is used in σ.

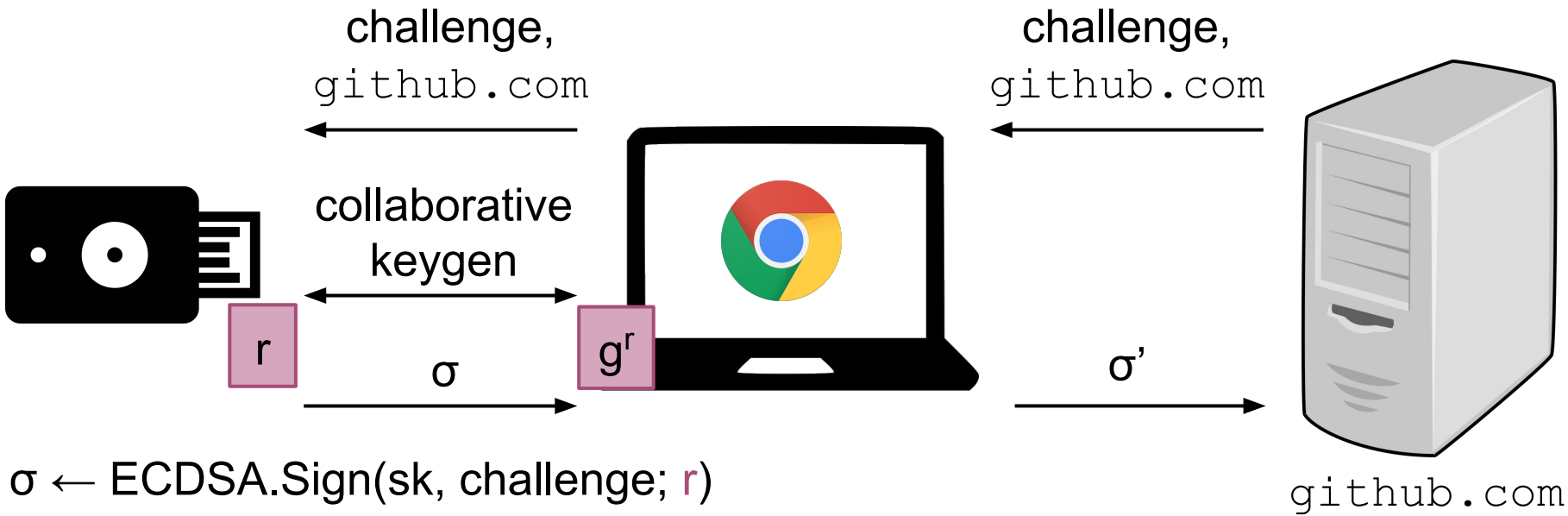
Authentication construction idea



σ ← ECDSA.Sign(sk, challenge; r)

Use g^r to verify that r is used in σ.

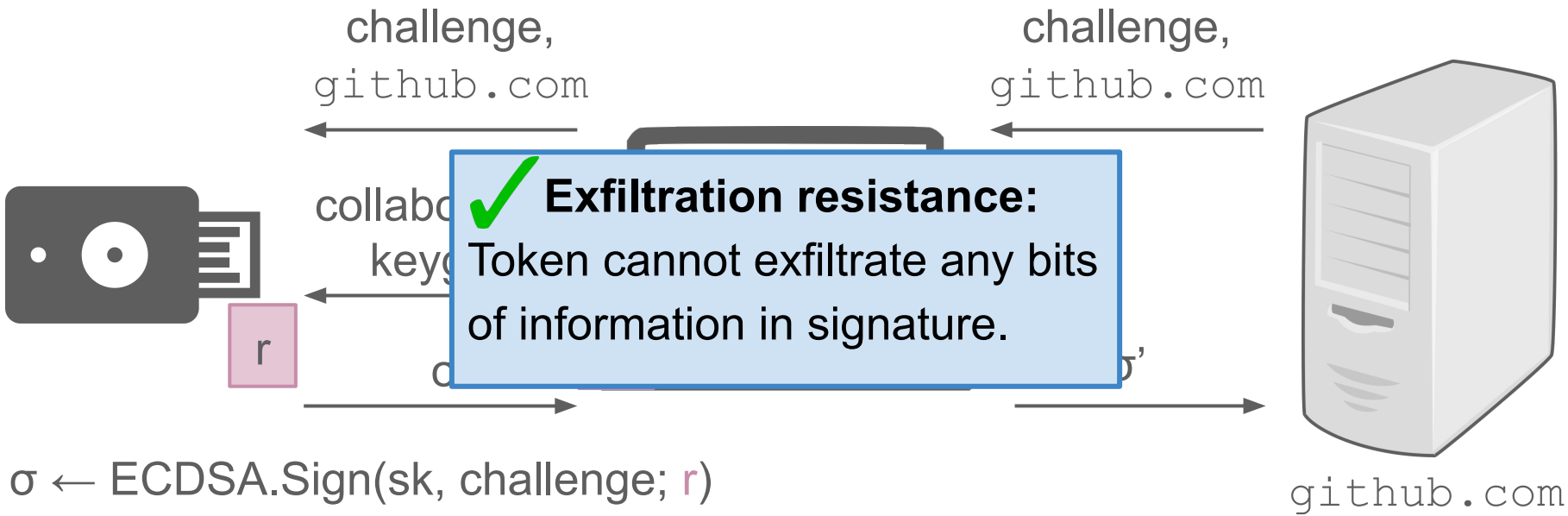
Authentication construction idea



σ ← ECDSA.Sign(sk, challenge; r)

Use g^r to verify that r is used in σ.
Randomly choose σ' from two
valid signatures.

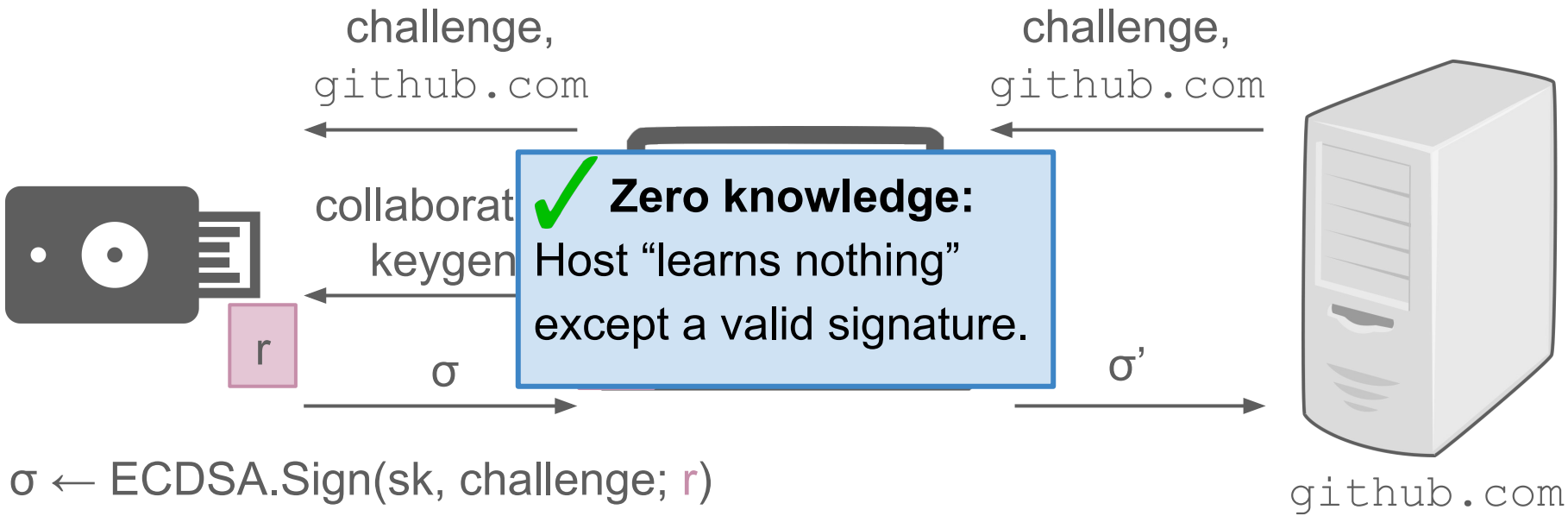
Authentication construction idea



σ ← ECDSA.Sign(sk, challenge; r)

Use g^r to verify that r is used in σ.
Randomly choose σ' from two valid signatures.

Authentication construction idea



σ ← ECDSA.Sign(sk, challenge; r)

Use g^r to verify that r is used in σ.
Randomly choose σ' from two
valid signatures.

Proposed protocol steps

✓0. Initialization (after purchasing a token)

✓1. Registration (associating a token with an account)

✓2. Authentication (logging into an account)

“Out-of-protocol” covert channels

Our protocol defends against “in-protocol” covert channels.

What about “out-of-protocol” covert channels?

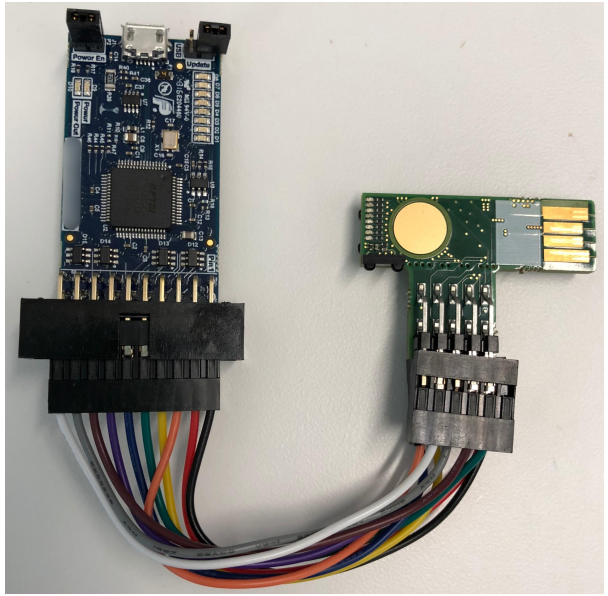
1. Timing: can be prevented by host
2. Failure: discard bad tokens
3. Malware
4. Other

[Hu92], [CBS04]

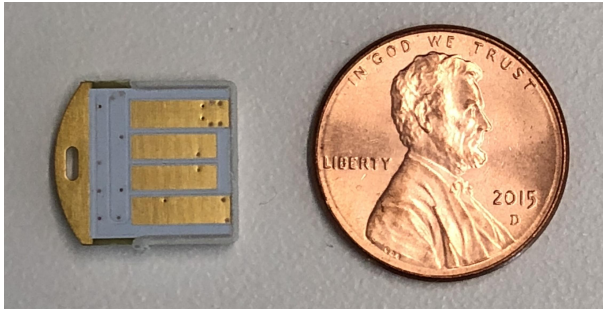
Other contributions (see paper)

- Flash-optimized data structure for storing U2F authentication counters
  - Provides stronger unlinkability than many existing U2F tokens
  - “Tear-resistant” and respects constraints of token flash
- Cryptographic optimizations tailored to token hardware
  - Offload hash-to-point to the host
  - Cache Verifiable Random Function outputs at the host

Implementation



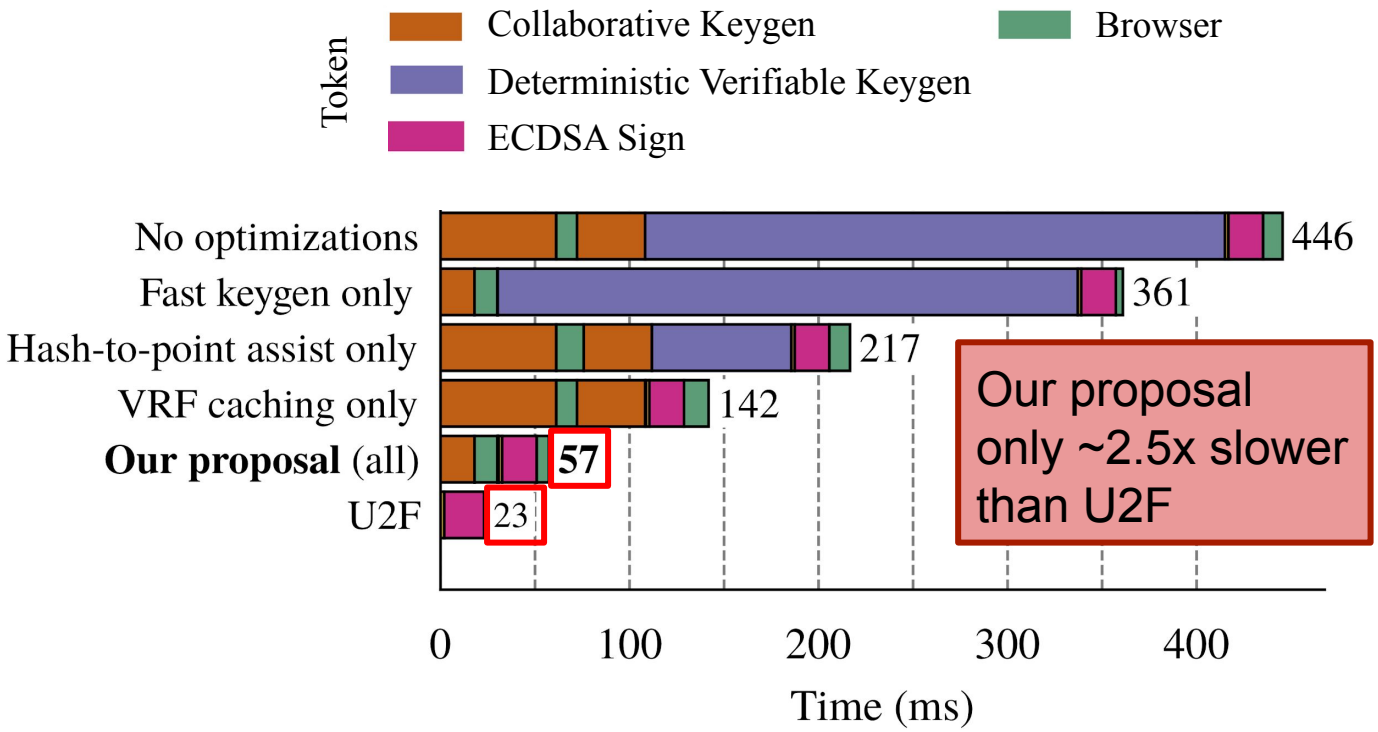
Google development board running our protocol.



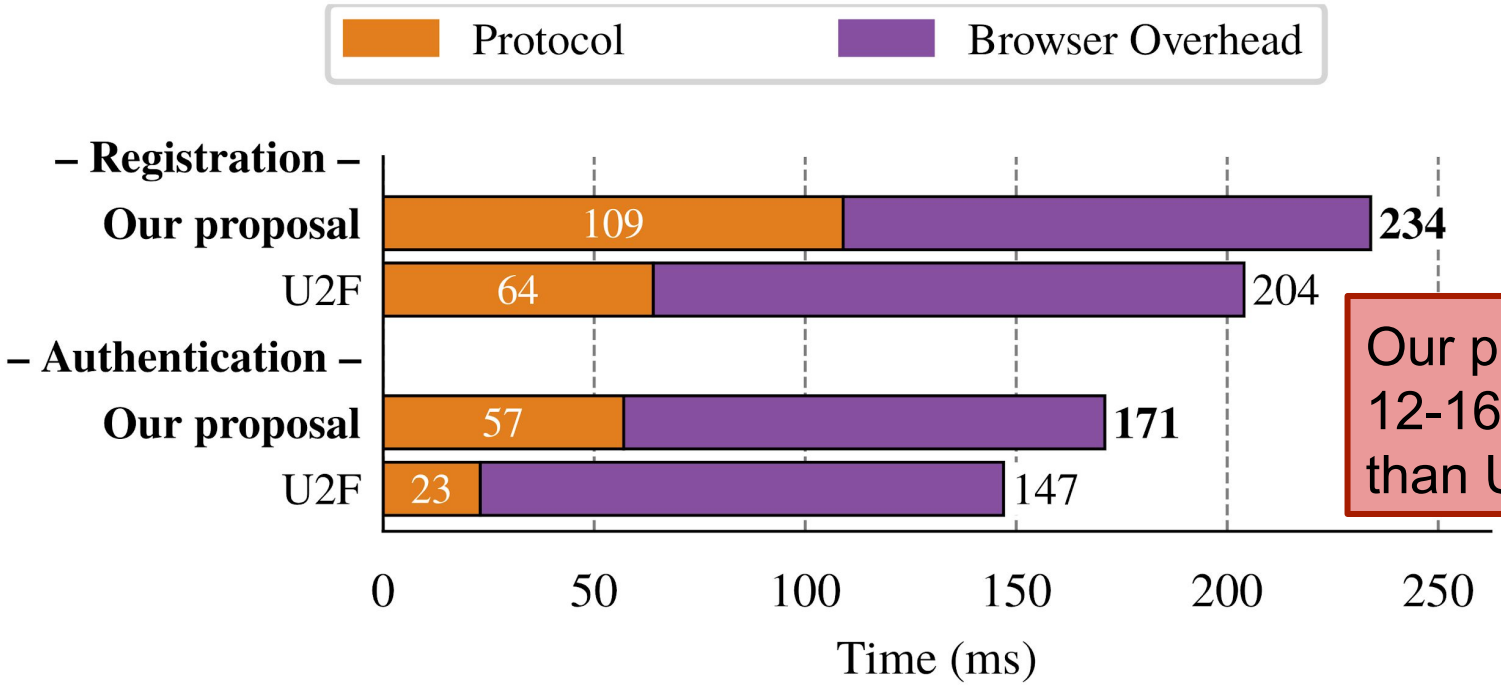
Google production USB token with same hardware specs.

ARM SC-300 processor clocked at 24 MHz

Minimal authentication overhead



Comparatively small end-to-end slowdown



Our protocol only 12-16% slower than U2F

Deployment considerations

Deployment considerations

How does the browser UI change?

- *Initialization*: prompt user to initialize unknown token
- *Errors:* warn user on token failure

Deployment considerations

How does the browser UI change?

- *Initialization*: prompt user to initialize unknown token
- *Errors:* warn user on token failure

How to support multiple browsers?

- Token gives mpk and counter to browser (protect against bugs)
- Sync mpk and counter across browser instances

CTAP2

- Cryptographic constructions carry over to CTAP2 (ECDSA P256).
- Core ideas remain, but still need full analysis.

Withstand untrustworthy hardware

Our proposal

- Augments U2F to protect against **faulty tokens**
- **Backwards-compatible** with U2F relying parties

Practical to deploy: performant on commodity hardware tokens

What is the next step?

Emma Dauterman
edauterman@cs.stanford.edu
https://arxiv.org/abs/1810.04660
Paper to appear at IEEE S&P 2019

References

[ACMT05] G. Ateniese, D. H. Chou, B. De Medeiros, and G. Tsudik. Sanitizable signatures. In *ESORICS*, 2005.

[BPR14] M.Bellare, K.G.Paterson,and P.Rogaway. Security of symmetric encryption against mass surveillance. In *CRYPTO*, 2014.

[BLS04] D. Boneh, B. Lynn, and H. Shacham. Short signatures from the Weil pairing. *Journal of cryptology*, 17(4), 2004.

[CBS04] S .Cabuk, C.E. Brodley, and C. Shields. IP covert timing channels: design and detection. In *CCS*, 2004.

[Des88] Y. Desmedt. Subliminal-free authentication and signature. In *EUROCRYPT*, 1988.

[DMS16] Y. Dodis, I. Mironov, and N. Stephens-Davidowitz. Message transmission with reverse firewalls—secure communication on corrupted machines. In *CRYPTO*, 2016.

[DY05] Y. Dodis and A. Yampolskiy. A verifiable random function with short proofs and keys. In *PKC*, 2005.

[GRPV18] S. Goldberg, L. Reyzin, D. Papadopoulos, and J. Vcelak. Verifiable random functions (VRFs). IETF CFRG Internet-Draft (Standards Track), Mar. 2018. https://tools.ietf.org/html/ draft-irtf-cfrg-vrf-01.

[Hu92] W.-M. Hu. Reducing timing channels with fuzzy time. *Journal of computer security*, 1(3-4):233–254, 1992.

[MRV99] S. Micali, M. Rabin, and S. Vadhan. Verifiable random functions. In *FOCS*, 1999.

[MS15] I. Mironov and N. Stephens-Davidowitz. Cryptographic reverse firewalls. In *EUROCRYPT*, 2015.

[Sim84] G. J. Simmons. The Prisoners' Problem and the Subliminal Channel. In *CRYPTO*, 1984.

Table 7: Cost of various operations on the token, averaged over 100 runs, and the expected number of each operation required per authentication attempt. "HW?" indicates use of the token's crypto accelerator.

Operation	**HW?**	**Time** (µs)	**Ops. per auth.**					
			No opts.	+ Fast keygen	+ VRF caching	+ Hash assist	**True2F** (+ all)	U2F
SHA256 (128 bytes)	Y	19	5	5	3	5	3	1
$x + y \in \mathbb{Z}_q$	N	36	17	16	2	15	1	0
$x \cdot y \in \mathbb{Z}_q$	N	409	9	8	2	11	1	0
$g^x \in \mathbb{G}$	Y	17,400	7	5	3	7	1	0
ECDSA.Sign	Y	18,600	1	1	1	1	1	1
$g \cdot h \in \mathbb{G}$	N	25,636	1	0	1	1	0	0
$\sqrt{x} \in \mathbb{Z}_q$	N	105,488	2	2	0	0	0	0