True2F: Backdoor-resistant authentication tokens

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

Stanford and Google

To appear at Oakland 2019

U2F: effective hardware 2FA

U2F: effective hardware 2FA

KrebsonSecurity

In-depth security news and investigation

23 JUL 18

Google: Security Keys Neutralized Employee Phishing

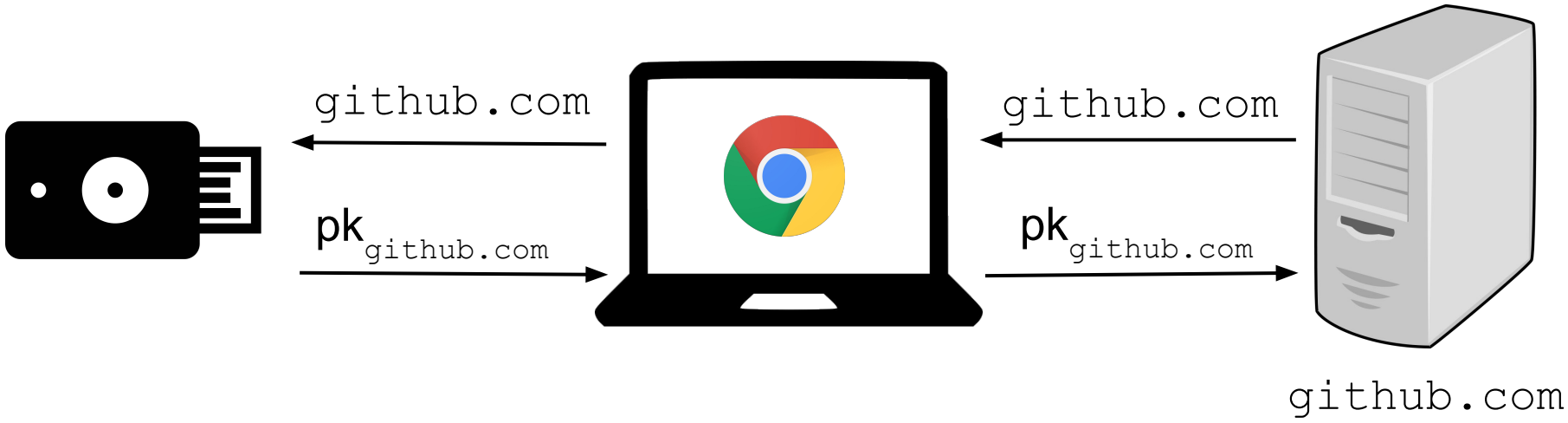
Google has not had any of its 85,000+ employees successfully phished on their work-related accounts since early 2017, when it began requiring all employees to use physical Security Keys in place of passwords and one-time codes, the company told KrebsOnSecurity.

U2F protocol steps

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

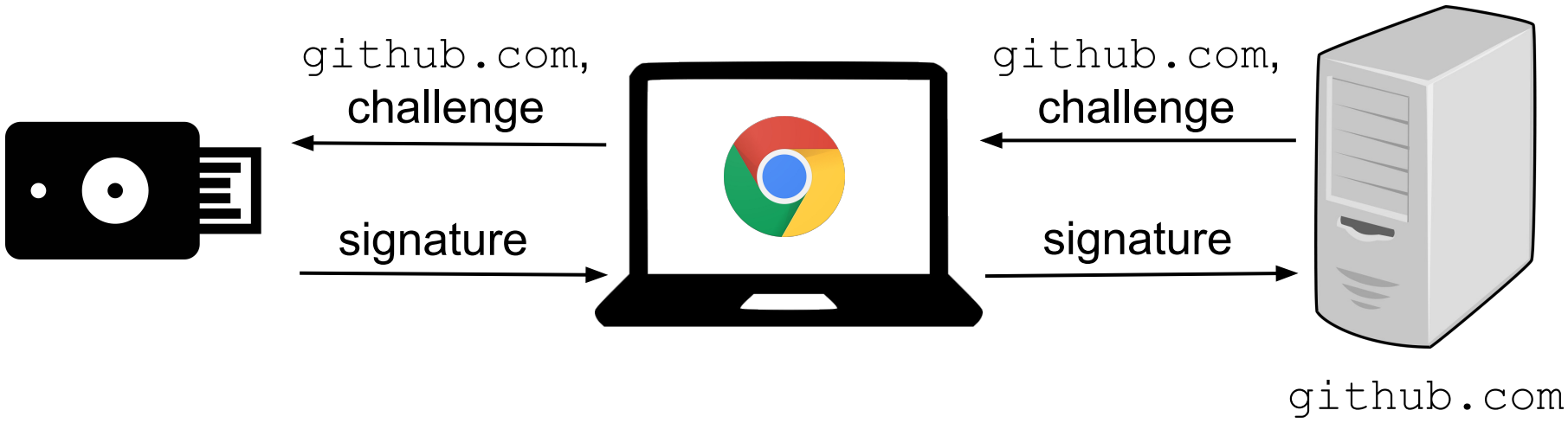
U2F Step #1: Registration

Associate a token with an account.



U2F Step #2: Authentication

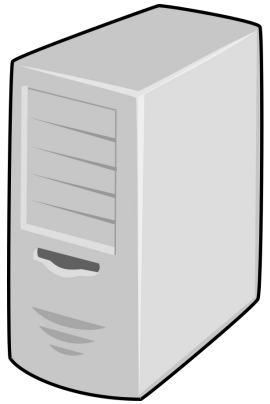
Log into an account.



U2F defends against phishing and browser compromise

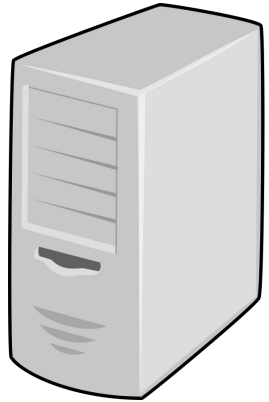
Even if malware takes over your browser, it can't authenticate without the token.

github.com,
challenge

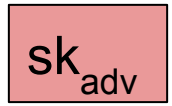


github.com

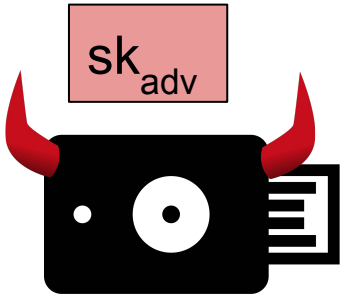
… but what about token compromise?



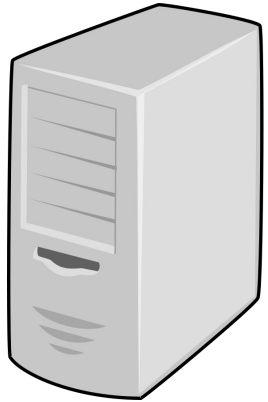
github.com



… but what about token compromise?

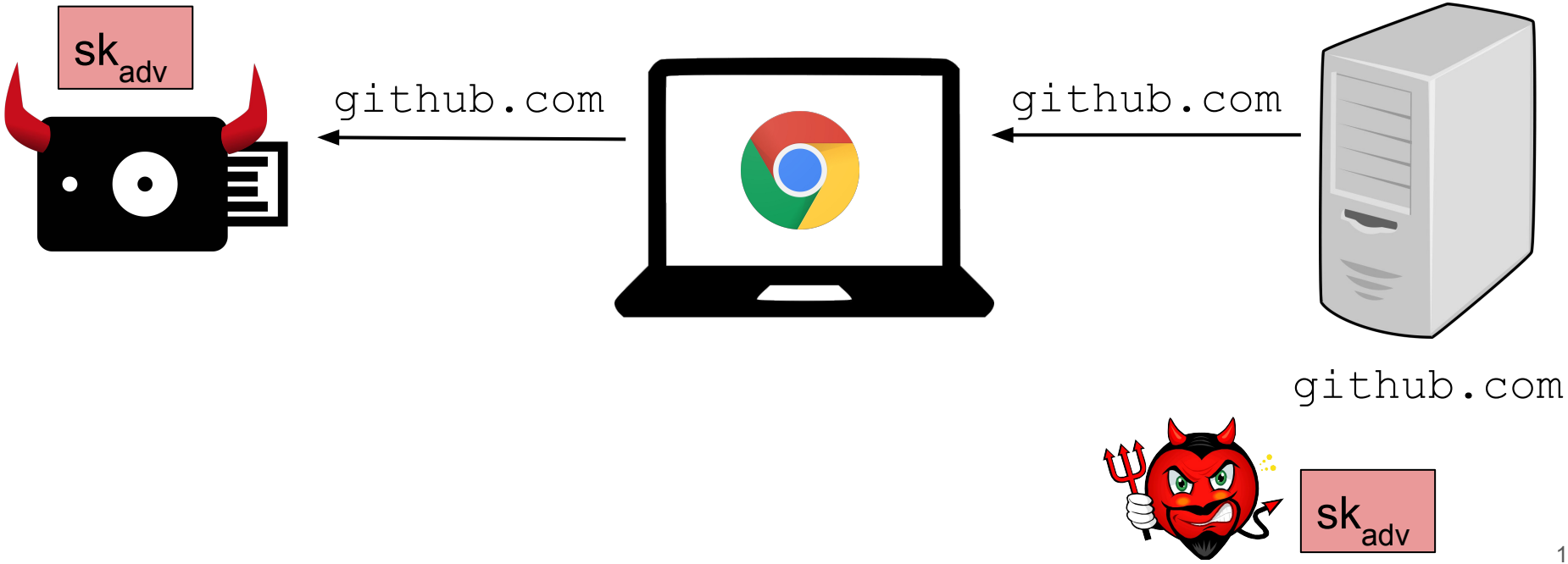


github.com

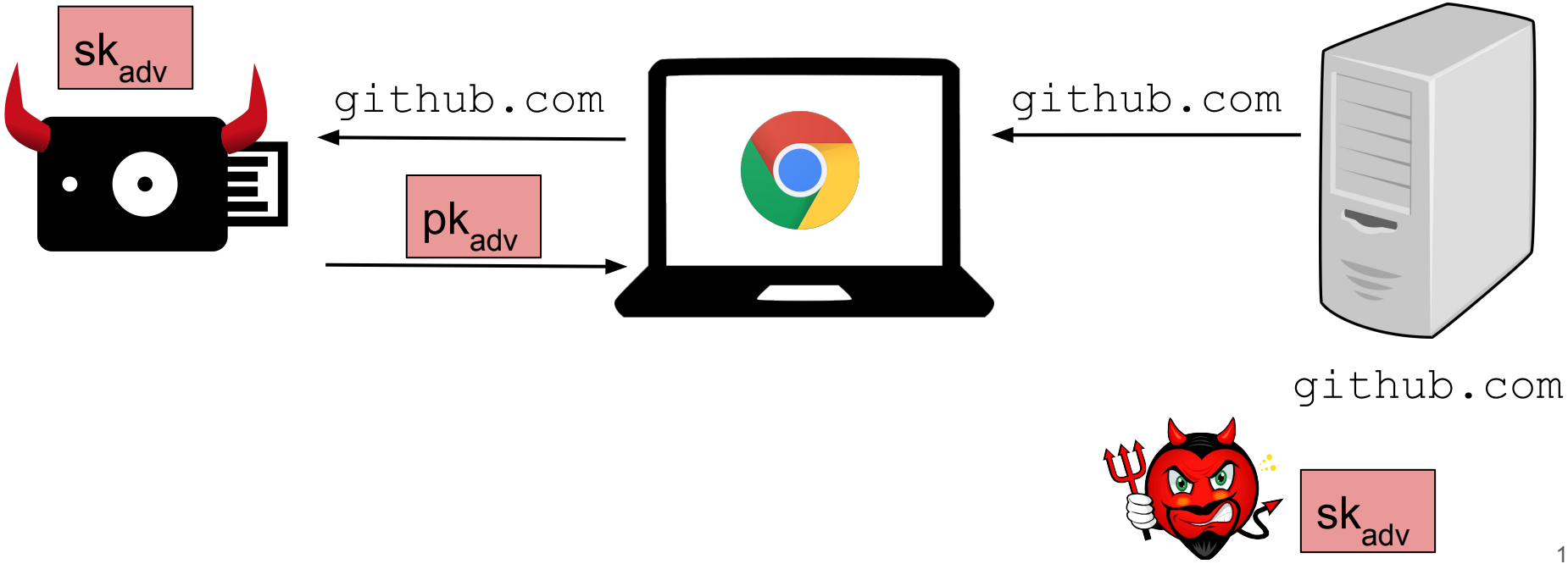


github.com

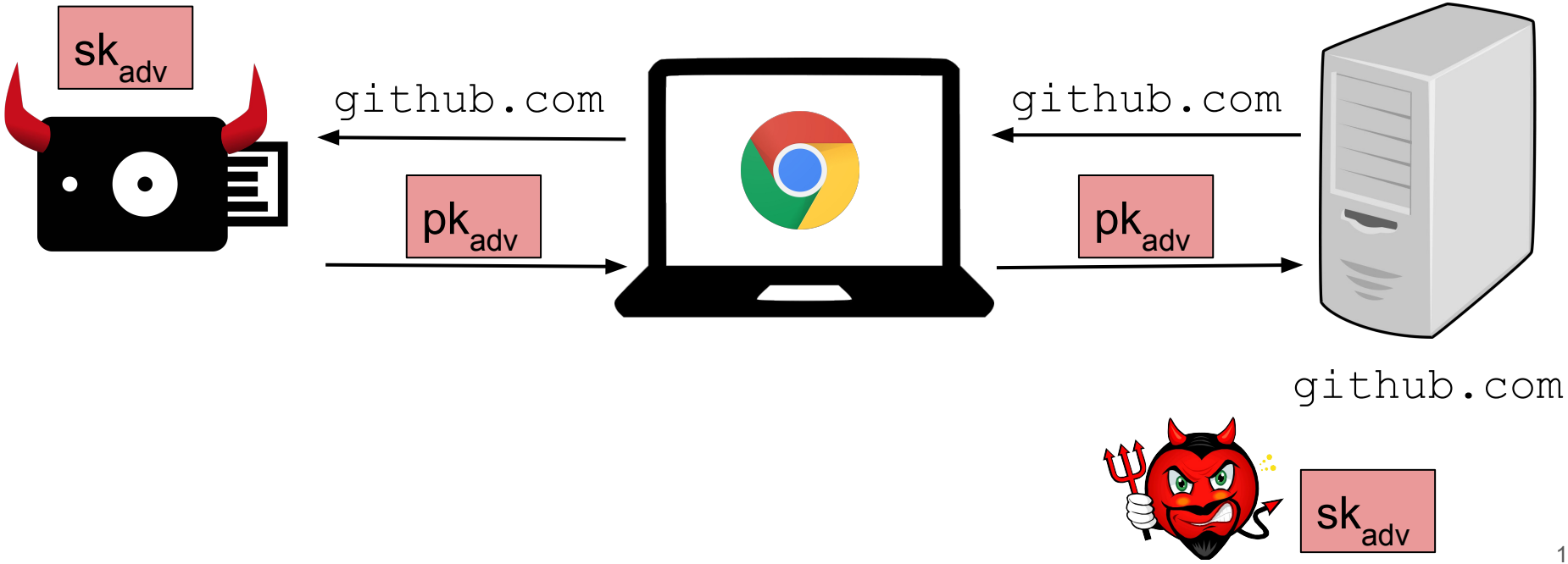
… but what about token compromise?



… but what about token compromise?



… but what about token compromise?



True2F: enhancement of U2F

Goals of True2F:

- Augment U2F to protect against **backdoored tokens**
  - Strongest-link security between the token and the browser
- **Backwards-compatible** with existing U2F servers
- **Performant** on commodity hardware tokens

True2F: enhancement of U2F

Goals of True2F:

- Augment U2F to protect against **backdoored tokens**
  - Strongest-link security between the token and the browser
- **Backwards-compatible** with existing U2F servers
- **Performant** on commodity hardware tokens

Design principles for True2F:

- Both browser and token **contribute randomness** to the protocol.
- All deterministic token operations can be **verified by the browser**.

U2F protocol steps

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

True2F protocol steps

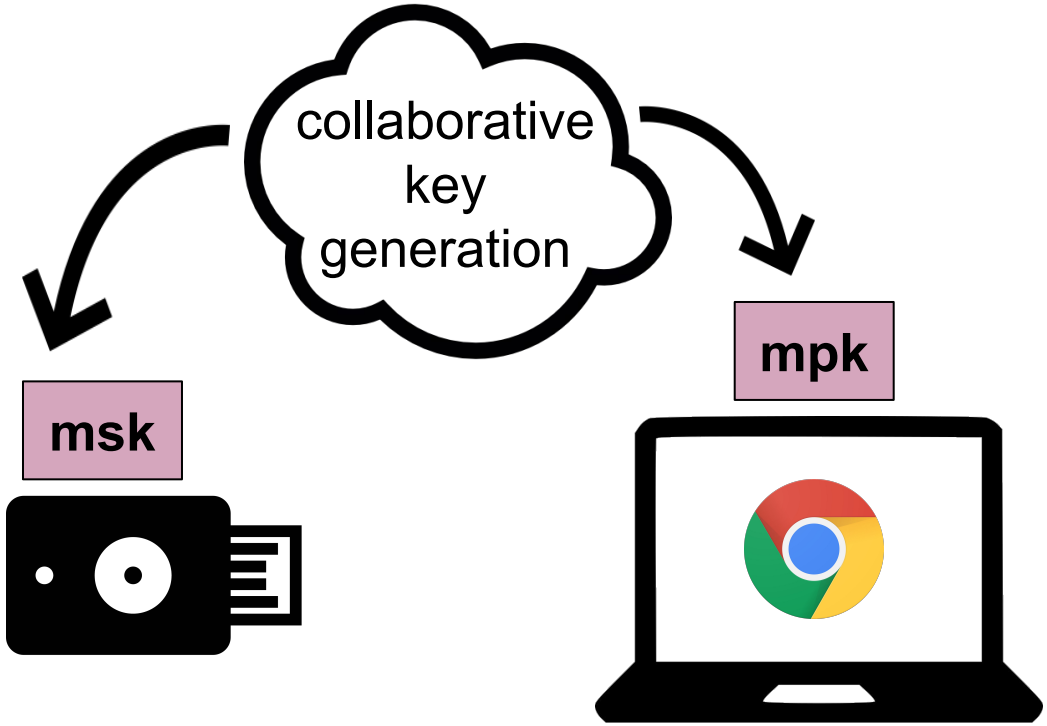
0. Initialization (after purchasing a token)
1. Registration (associating a token with an account)
2. Authentication (logging into an account)

True2F protocol steps

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

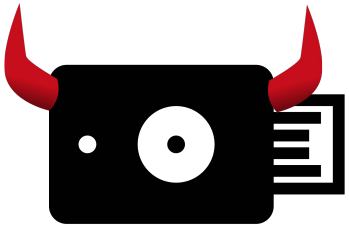
Approach: Both browser and token contribute randomness to the protocol.

Step #0: Initialization



Collaborative Key Generation properties

The token cannot bias the public key that the protocol outputs.

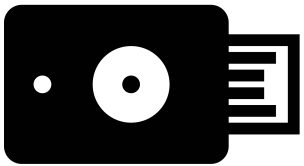


[CMBF13]

Collaborative Key Generation properties

The token cannot bias the public key that the protocol outputs.

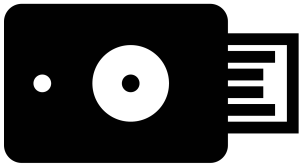
The browser learns nothing about the secret key.



[CMBF13]

Our Collaborative Key Generation protocol

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

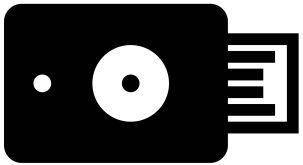


Our Collaborative Key Generation protocol

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

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

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

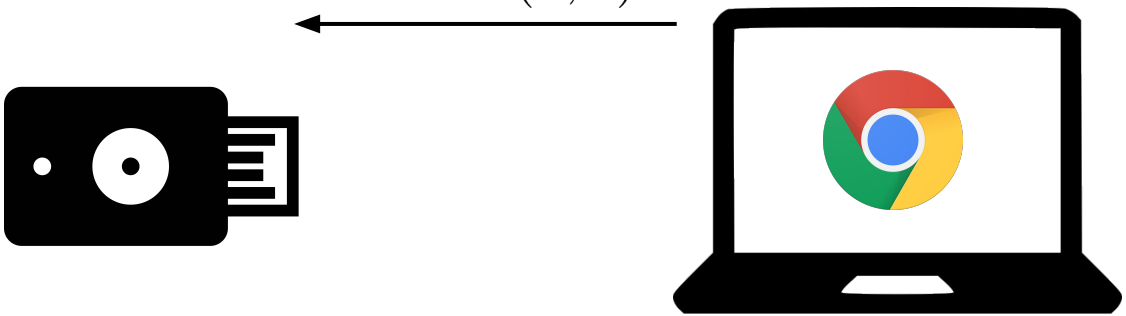


Our Collaborative Key Generation protocol

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

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

$c \leftarrow H(v, r)$



Our Collaborative Key Generation protocol

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

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

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

$c \leftarrow H(v, r)$

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

Our Collaborative Key Generation protocol

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

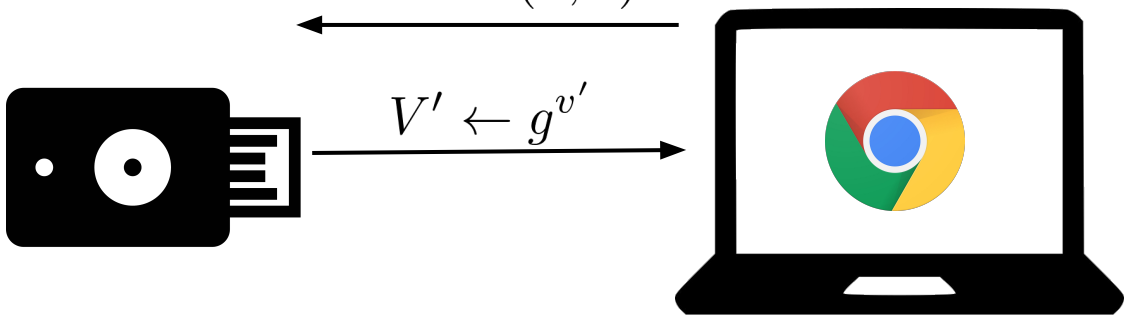
$r \xleftarrow{R} \mathbb{Z}_q$

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

$c \leftarrow H(v, r)$

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

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



Our Collaborative Key Generation protocol

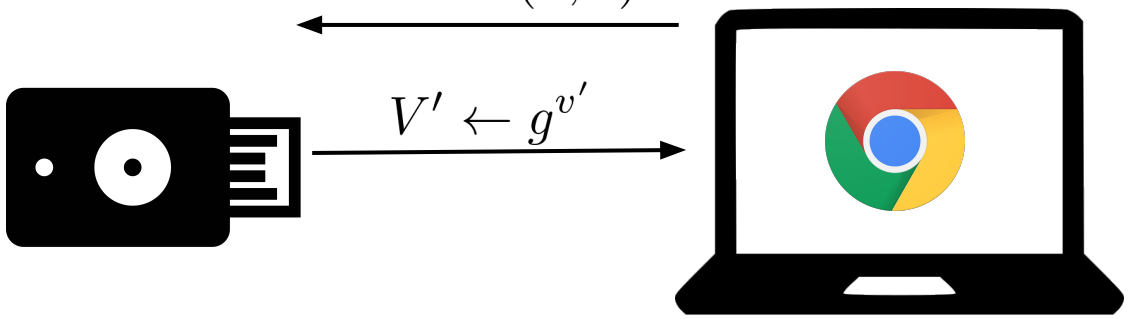
$\mathbb{G} = \langle g \rangle$ is a group of prime order q.
ECDSA keypairs have the form $(x, g^x) \in \mathbb{Z}_q \times \mathbb{G}$.

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

$c \leftarrow H(v, r)$

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

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



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

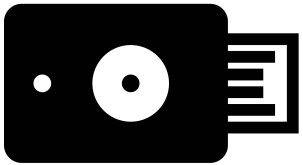
Our Collaborative Key Generation protocol

$\mathbb{G} = \langle g \rangle$ 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

$\mathsf{mpk} \leftarrow V' \cdot g^v$

$= g^{v+v'}$

Our Collaborative Key Generation protocol

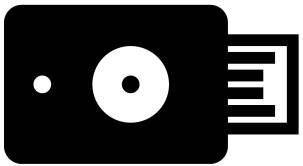
$\mathbb{G} = \langle g \rangle$ is a group of prime order q.
ECDSA keypairs have the form $(x, g^x) \in \mathbb{Z}_q \times \mathbb{G}$.

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

$c \leftarrow H(v, r)$

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

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



v, r

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

check $c = H(v, r)$

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

Our Collaborative Key Generation protocol

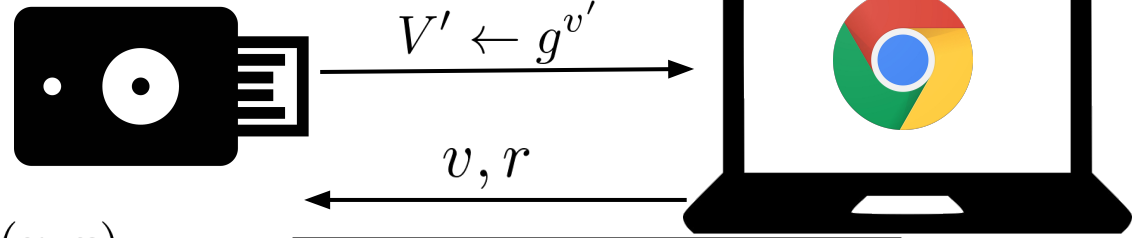
$\mathbb{G} = \langle g \rangle$ is a group of prime order q.
ECDSA keypairs have the form $(x, g^x) \in \mathbb{Z}_q \times \mathbb{G}$.

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

$c \leftarrow H(v, r)$

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

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



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 the public key that the protocol outputs.

Our Collaborative Key Generation protocol

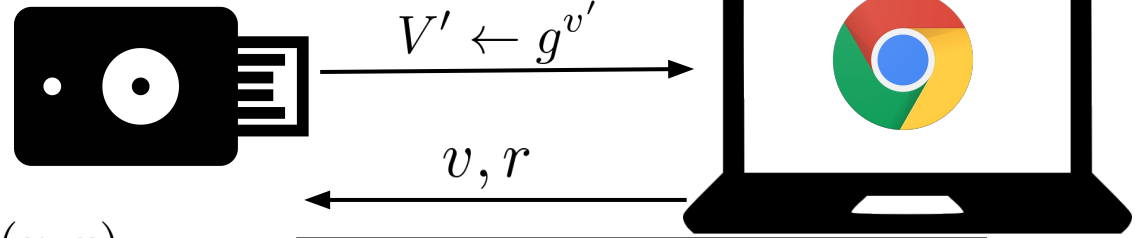
$\mathbb{G} = \langle g \rangle$ is a group of prime order q.
ECDSA keypairs have the form $(x, g^x) \in \mathbb{Z}_q \times \mathbb{G}$.

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

$c \leftarrow H(v, r)$

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

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



v, r

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

check $c = H(v, r)$

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

✓ The browser learns nothing about the secret key (except the public key).

True2F protocol steps

✓0. Initialization (after purchasing a token)

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

True2F protocol steps

✓0. Initialization (after purchasing a token)

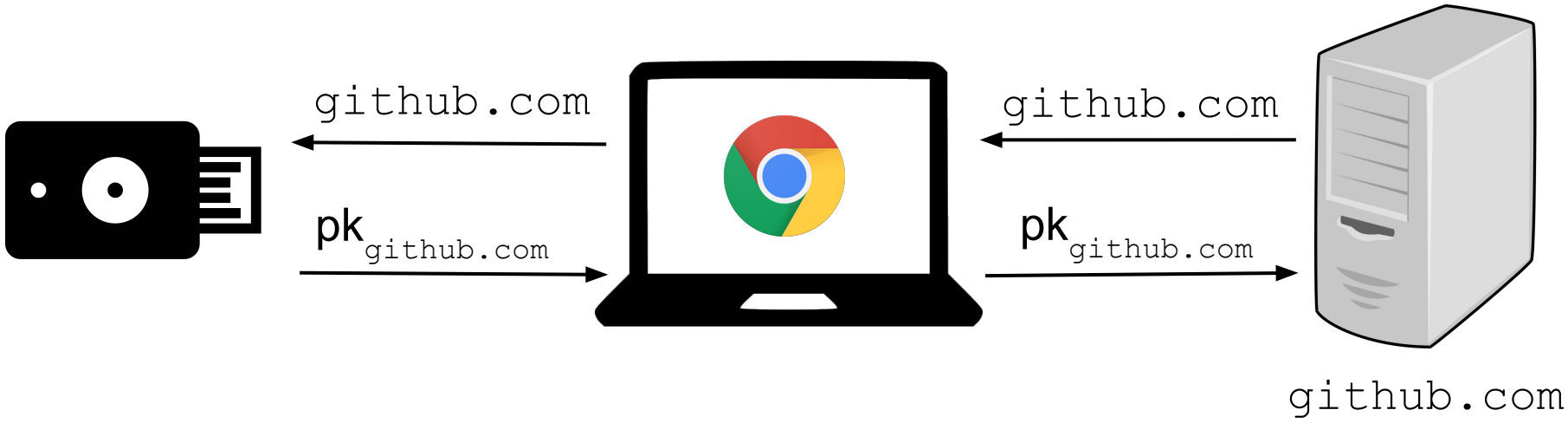
1. Registration (associating a token with an account)

2. Authentication (logging into an account)

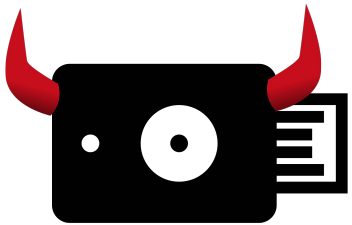
Approach: All deterministic token operations can be verified by the browser.

Step #1: U2F Registration

Associate a token with an account.

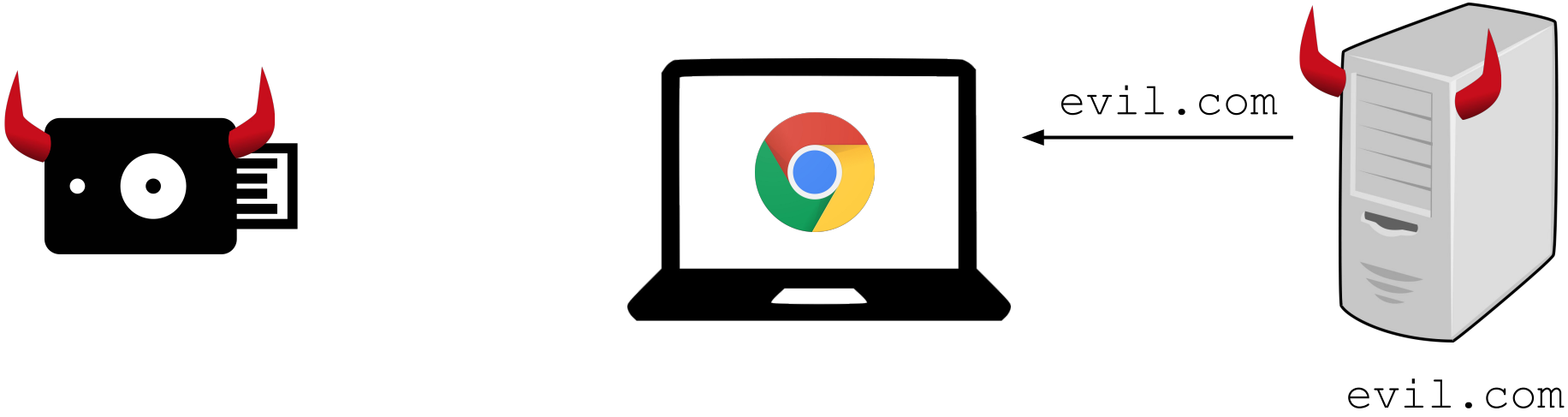


Supply-chain attack on U2F registration

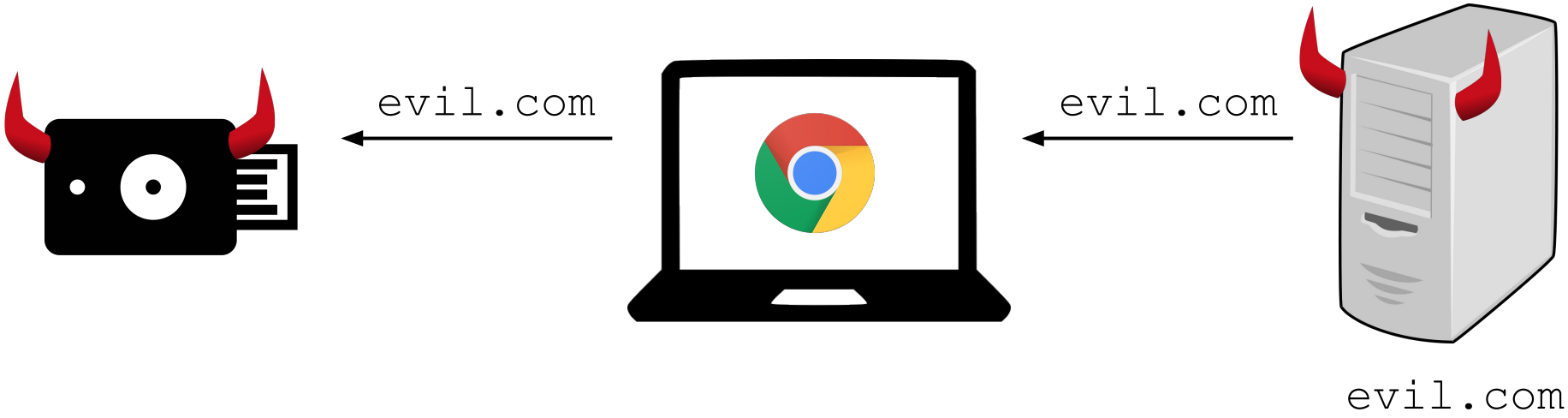


evil.com

Supply-chain attack on U2F registration



Supply-chain attack on U2F registration



Supply-chain attack on U2F registration

$pk_{evil.com} \leftarrow f(sk_{github.com})$

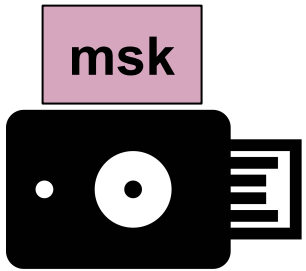
Supply-chain attack on U2F registration

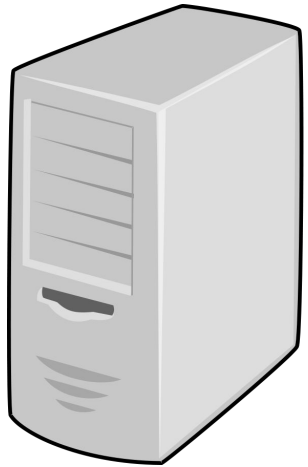
Supply-chain attack on U2F registration

Supply-chain attack on U2F registration

$pk_{evil.com} \leftarrow f(sk_{github.com})$

Verifiable Identity Families (VIFs)





Derive server-specific keypairs in a **deterministic** and **verifiable** way from a master keypair.

VIF properties

VIF properties

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

VIF properties

1. **Unique:** The token can produce the unique keypair for `github.com`.
2. **Verifiable:** The token can prove to the browser that pk is really the unique public key for `github.com`.

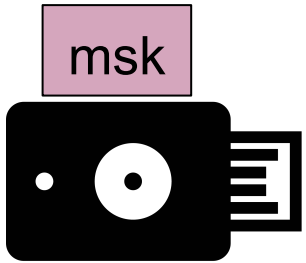
VIF properties

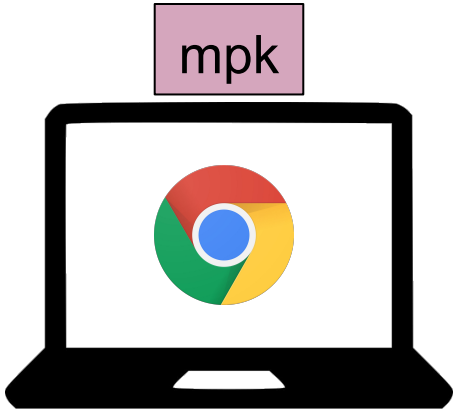
1. **Unique:** The token can produce the unique keypair for `github.com`.
2. **Verifiable:** The token can prove to the browser that pk is really the unique public key for `github.com`.
3. **Unlinkable:** The VIF public key for `github.com` is indistinguishable from a random ECDSA public key.

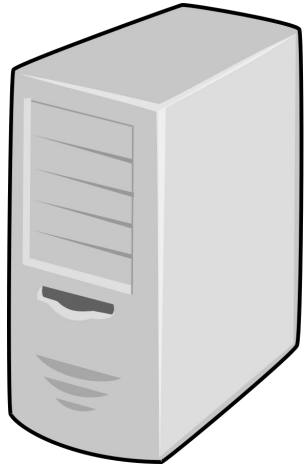
VIF properties

1. **Unique:** The token can produce the unique keypair for `github.com`.
2. **Verifiable:** The token can prove to the browser that pk is really the unique public key for `github.com`.
3. **Unlinkable:** The VIF public key for `github.com` is indistinguishable from a random ECDSA public key.
4. **Unforgeable:** The browser cannot forge signatures under VIF-generated public keys.

VIF construction overview

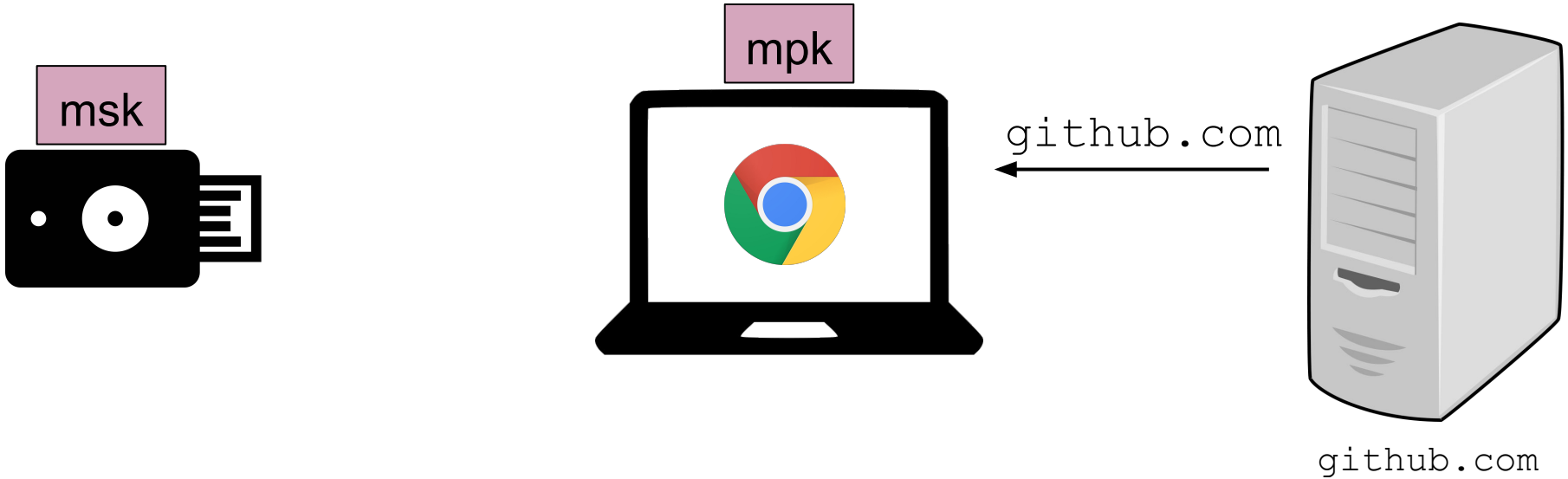




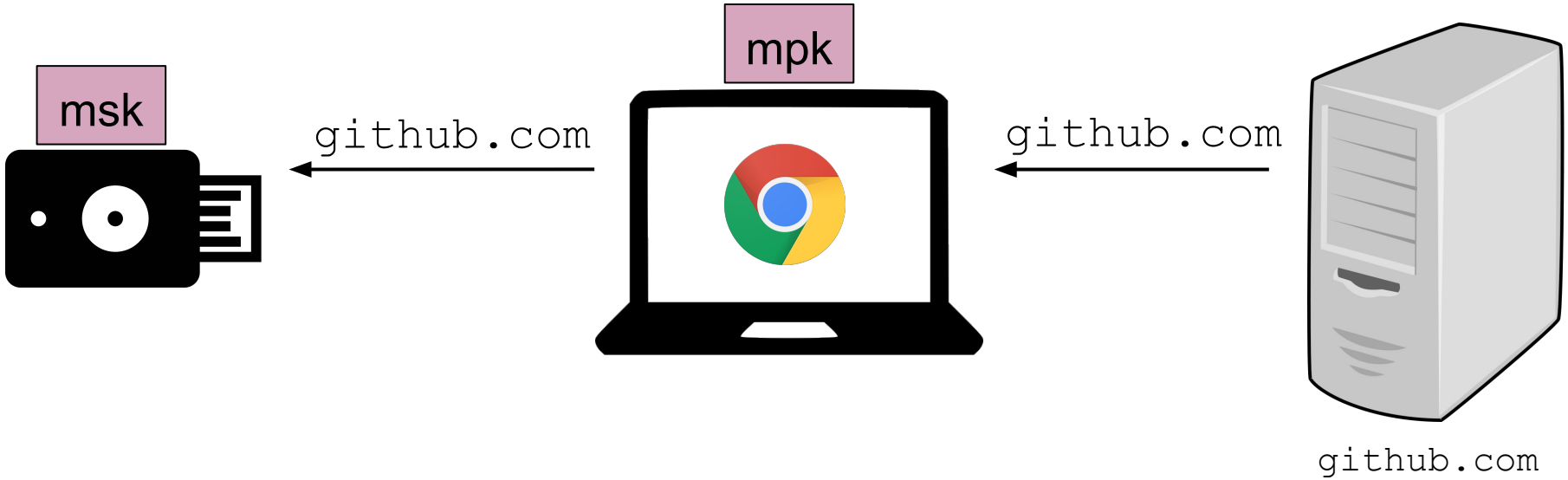


`github.com`

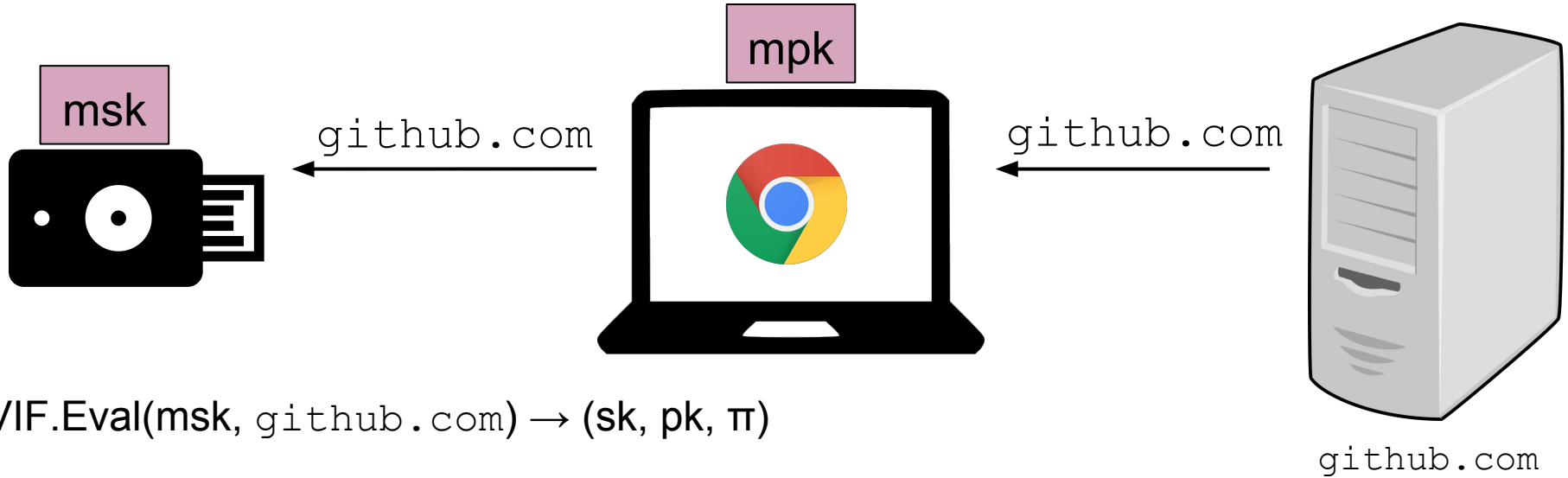
VIF construction overview



VIF construction overview

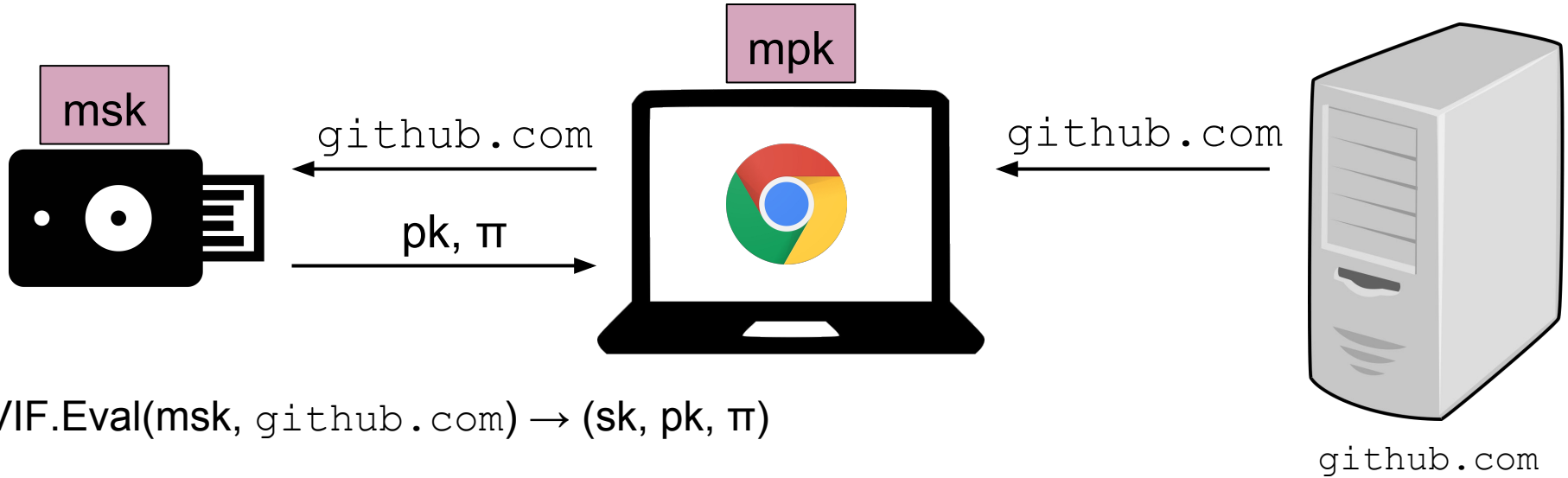


VIF construction overview



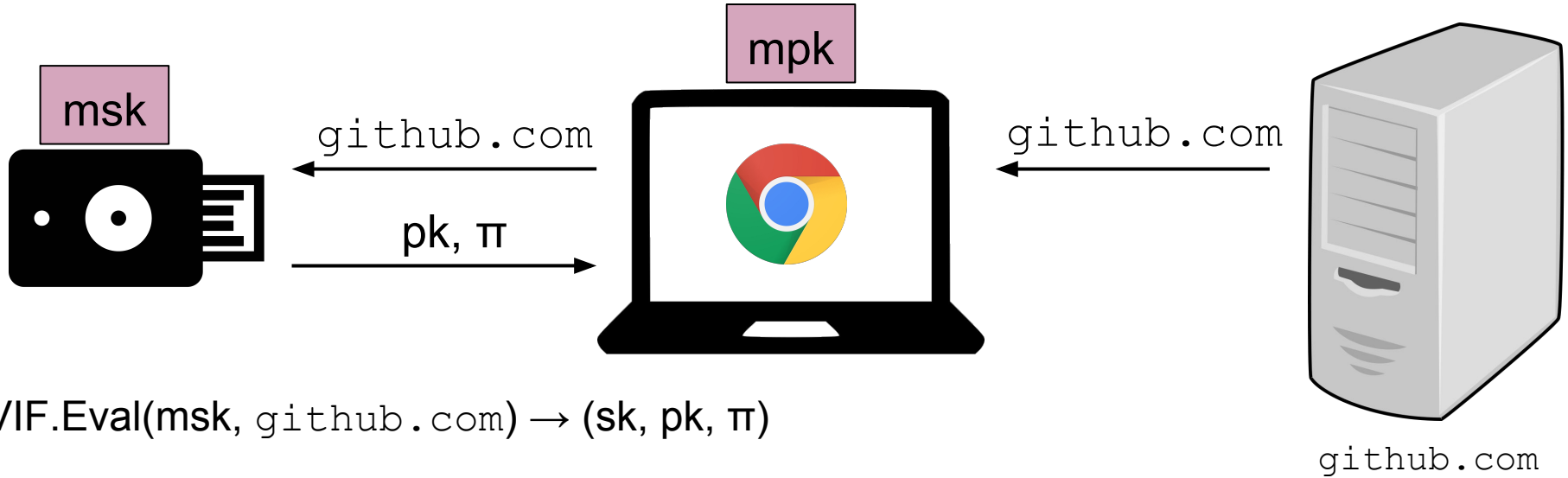
VIF.Eval(msk, `github.com`) → (sk, pk, π)

VIF construction overview



VIF.Eval(msk, `github.com`) → (sk, pk, π)

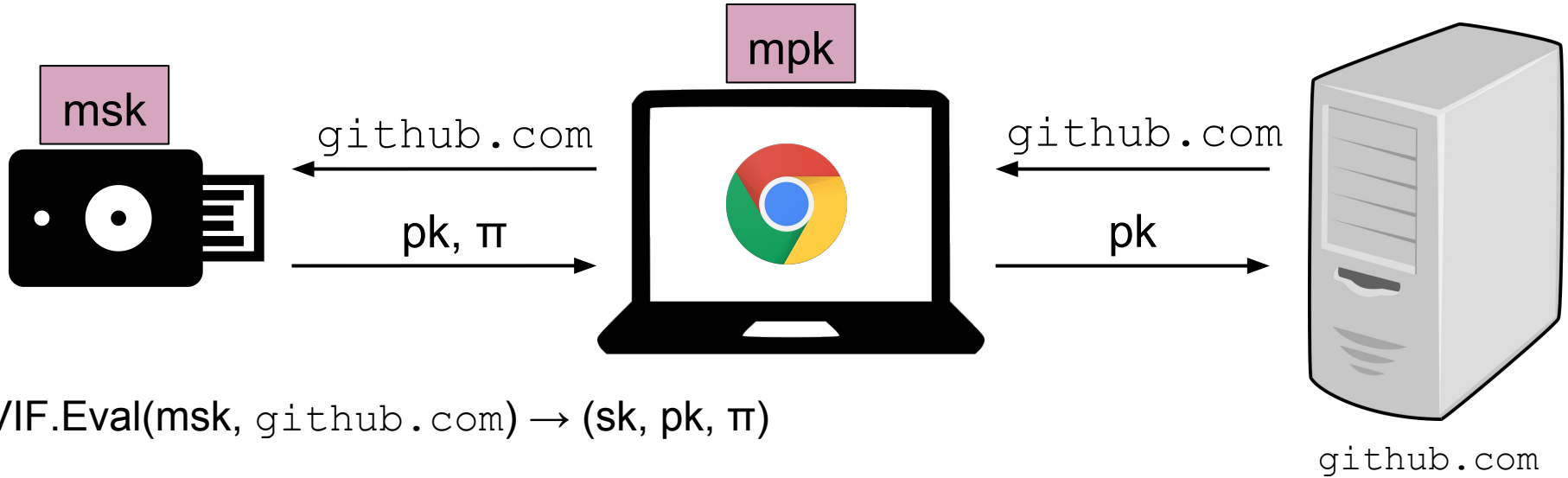
VIF construction overview



VIF.Eval(msk, `github.com`) → (sk, pk, π)

VIF.Verify(mpk, `github.com`, pk, π) → {0,1}

VIF construction overview



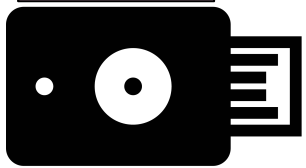
VIF.Eval(msk, `github.com`) → (sk, pk, π)

VIF.Verify(mpk, `github.com`, pk, π) → {0,1}

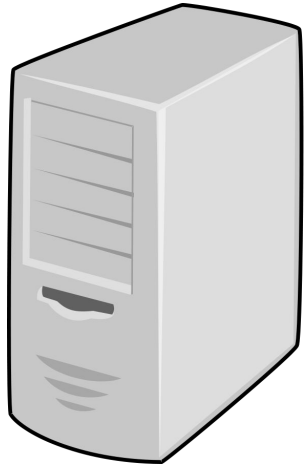
Simplified VIF construction

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

$x \in \mathbb{Z}_q$



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

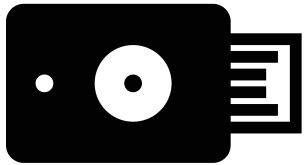


`github.com`

Simplified VIF construction

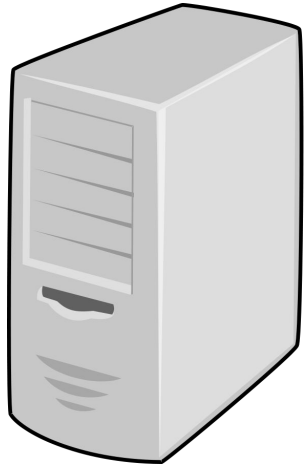
$\mathbb{G} = \langle g \rangle$ is a group of prime order q.

$x \in \mathbb{Z}_q$



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

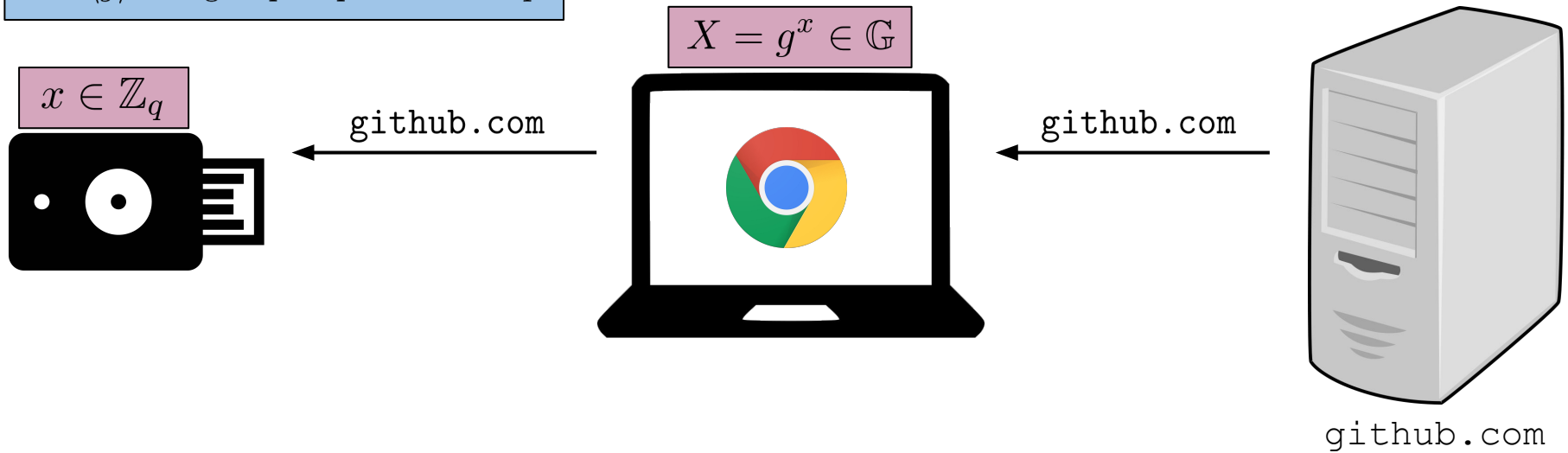
`github.com`



`github.com`

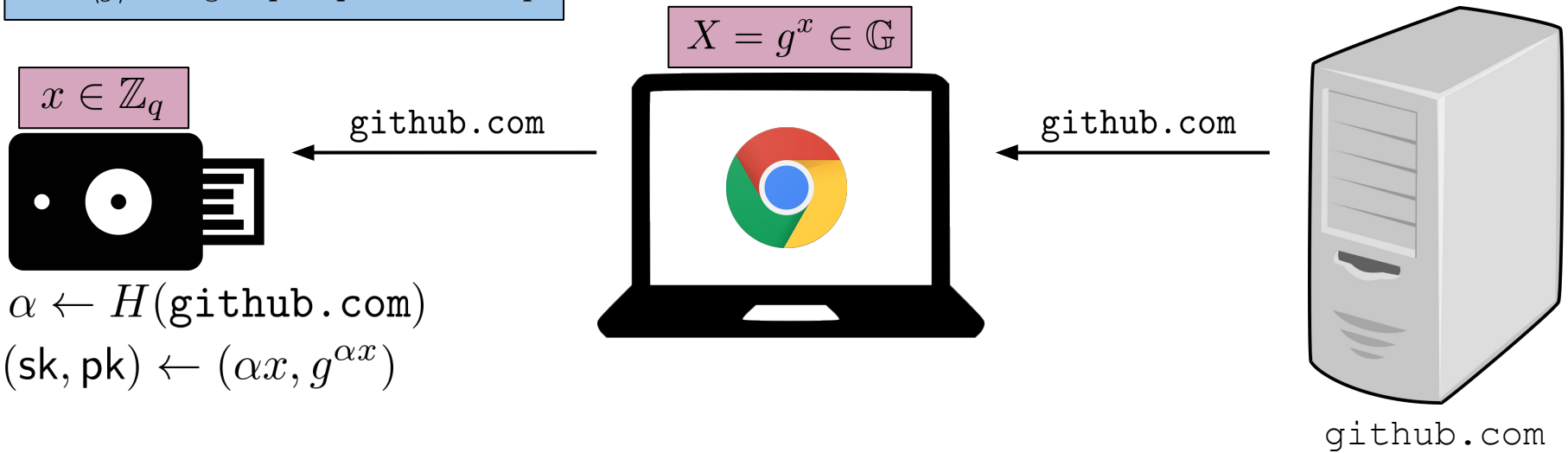
Simplified VIF construction

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



Simplified VIF construction

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

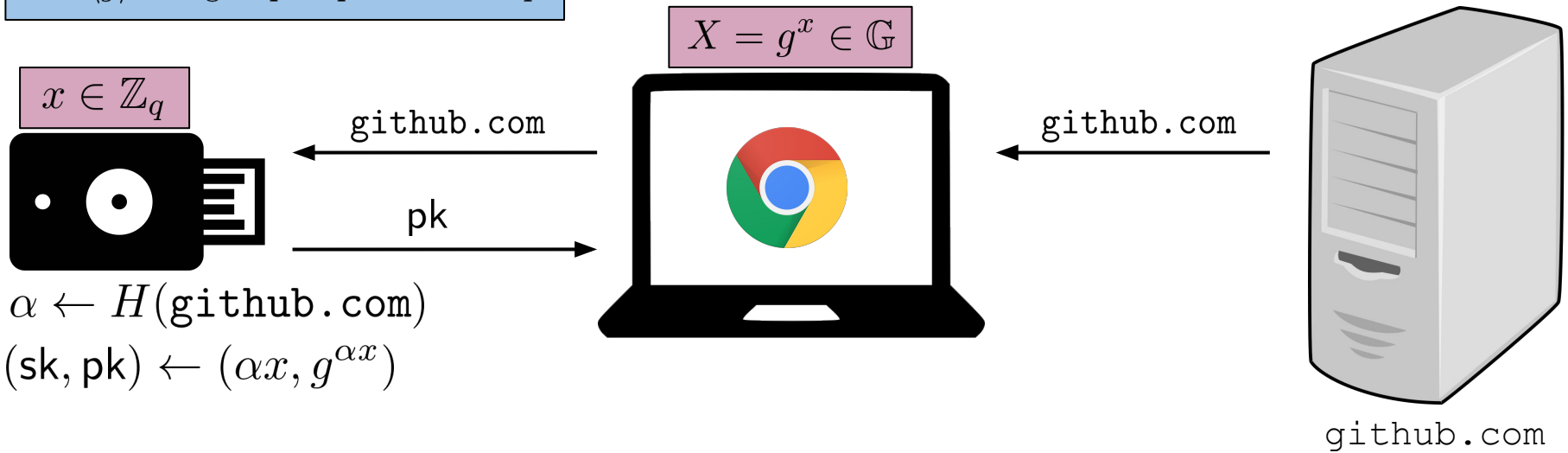


$\alpha \leftarrow H(\texttt{github.com})$

$(\mathsf{sk}, \mathsf{pk}) \leftarrow (\alpha x, g^{\alpha x})$

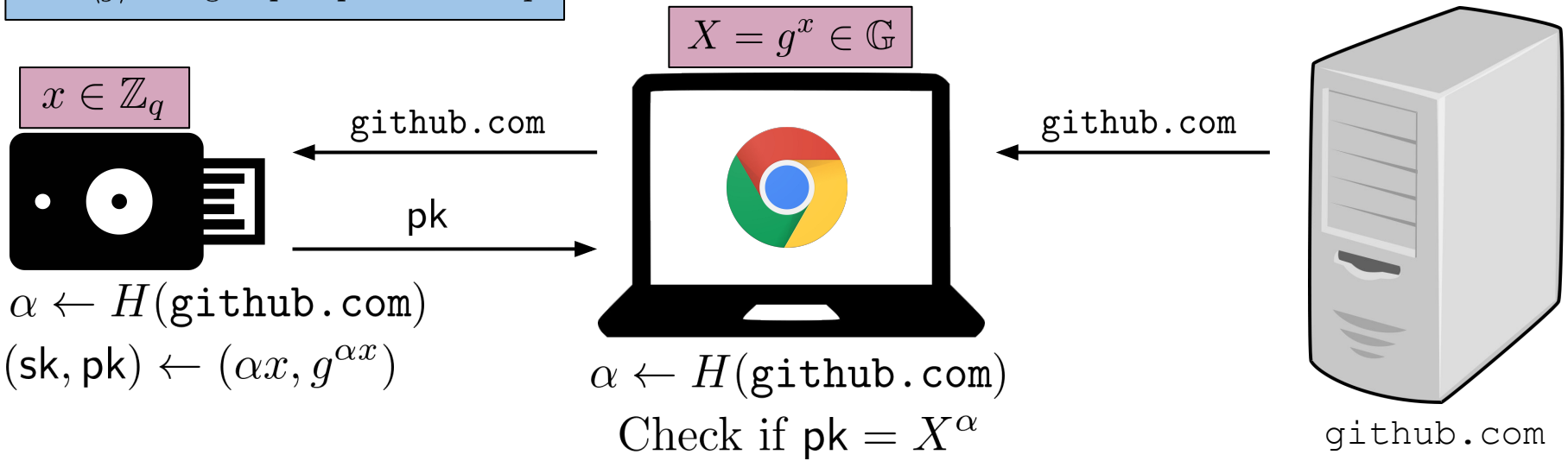
Simplified VIF construction

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



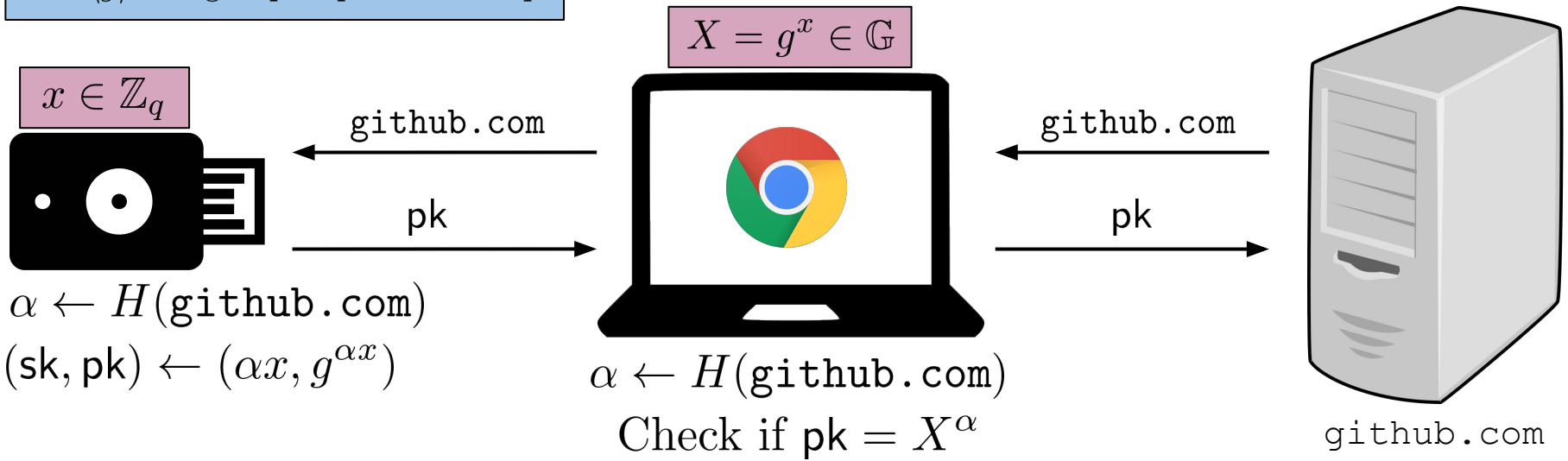
Simplified VIF construction

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



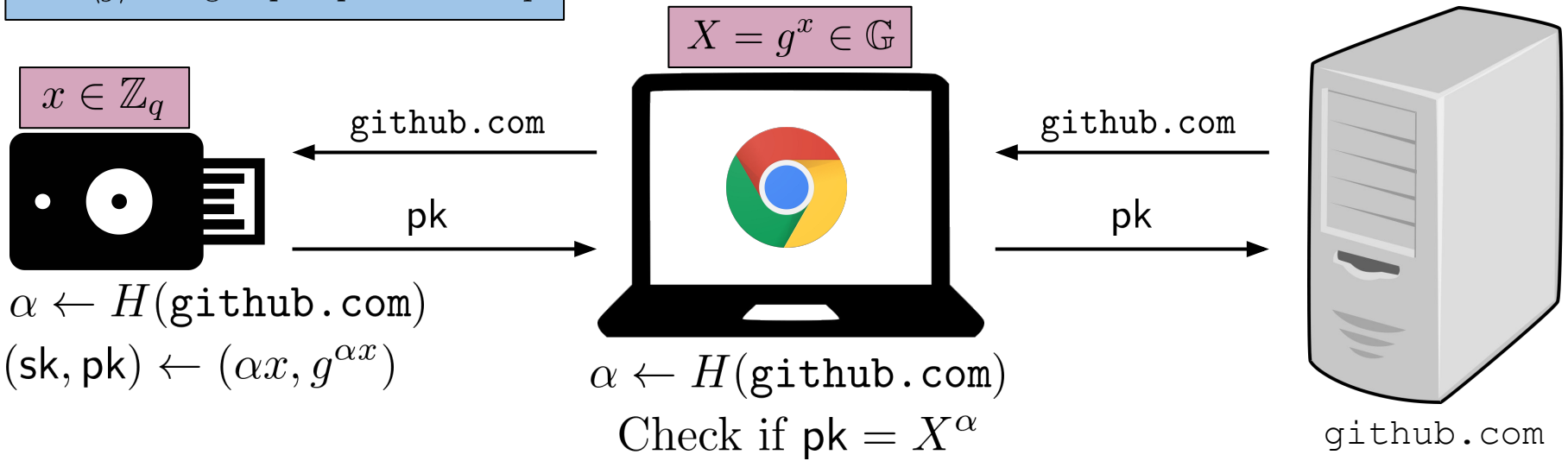
Simplified VIF construction

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



Simplified VIF construction

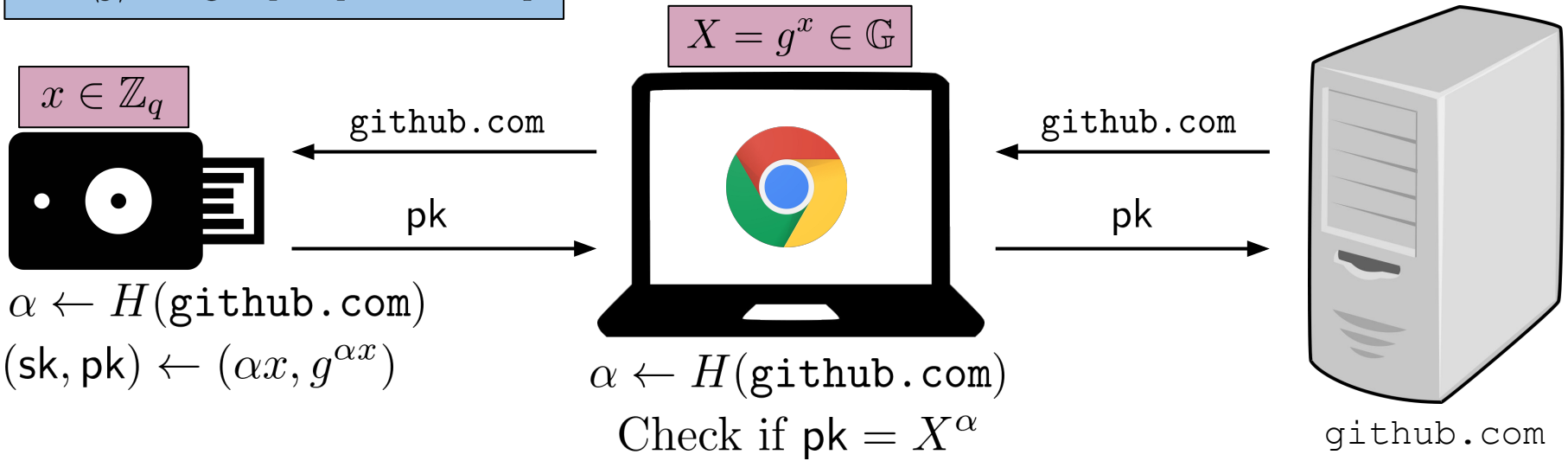
$\mathbb{G} = \langle g \rangle$ is a group of prime order q.



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

Simplified VIF construction

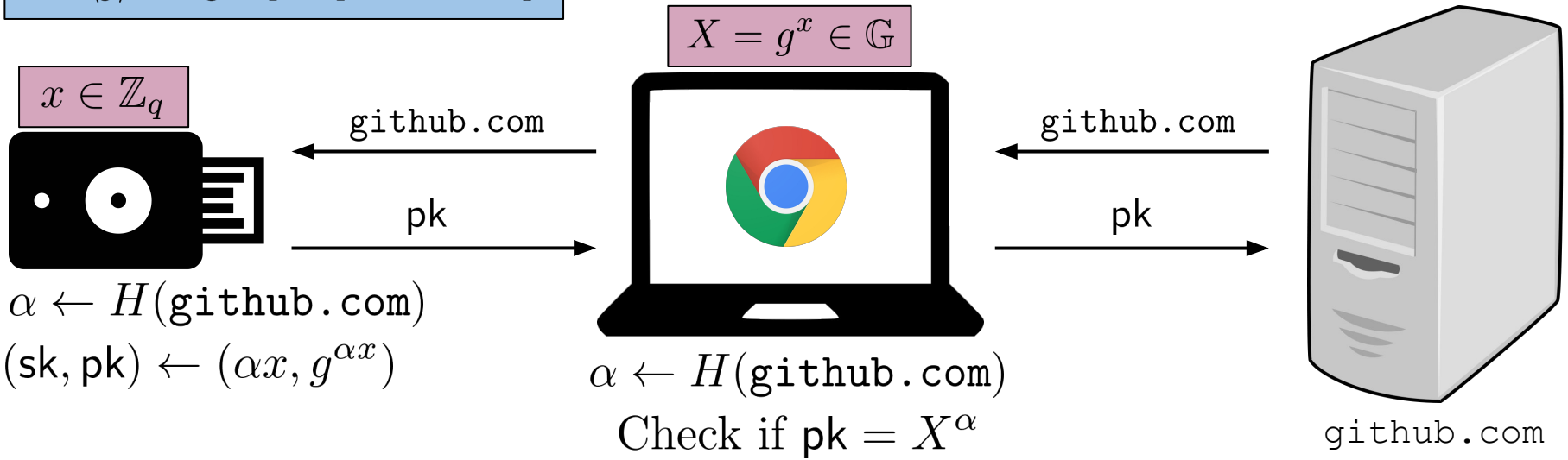
$\mathbb{G} = \langle g \rangle$ is a group of prime order q.



✓**Verifiable:** The token can prove to the browser that pk is really the unique public key for `github.com`.

Simplified VIF construction

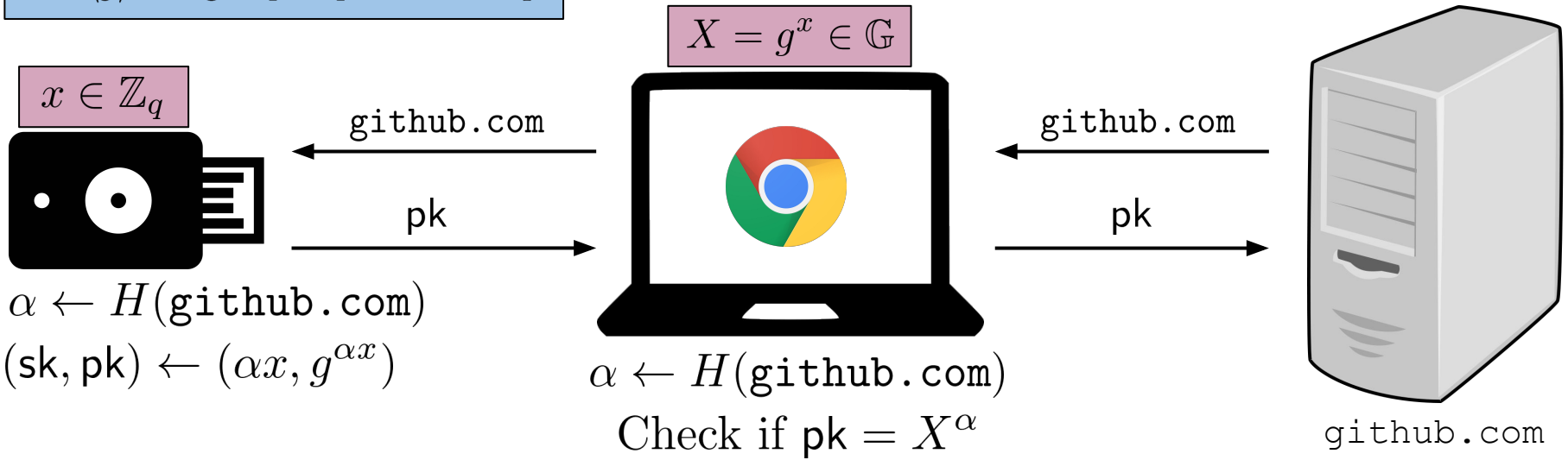
$\mathbb{G} = \langle g \rangle$ is a group of prime order q.



❌ **Unlinkable:** The VIF public key for `github.com` is indistinguishable from a random ECDSA public key.

Simplified VIF construction

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

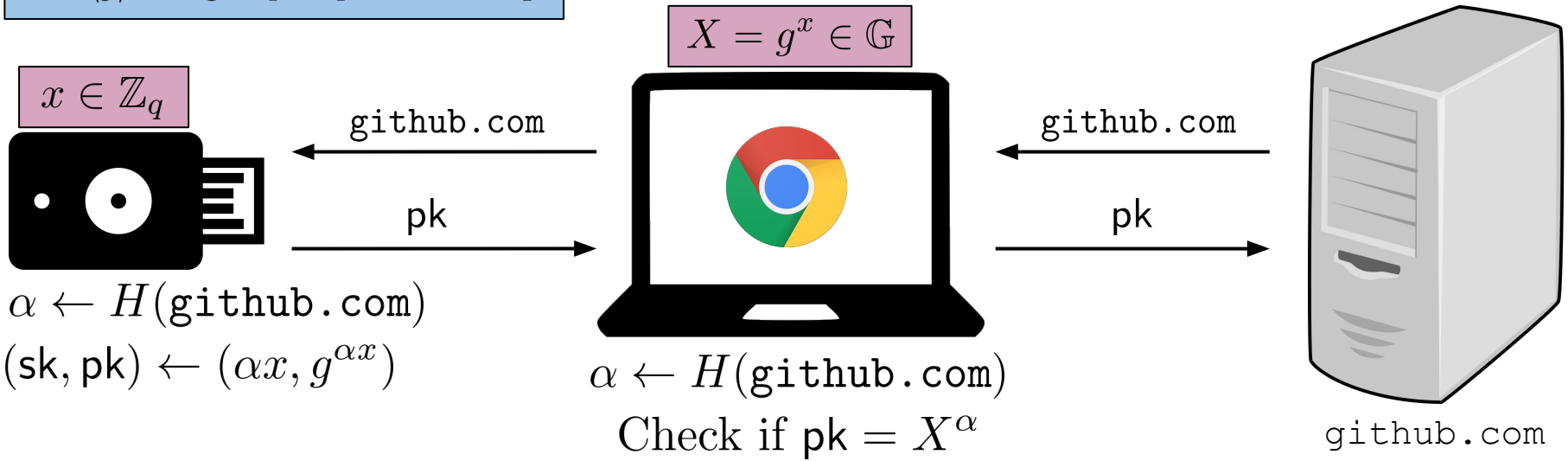


Verifiable Random Functions [MRV99], [DY05], [GRPV18]

✗ **Unlinkable:** The VIF public key for `github.com` is indistinguishable from a random ECDSA public key.

Simplified VIF construction

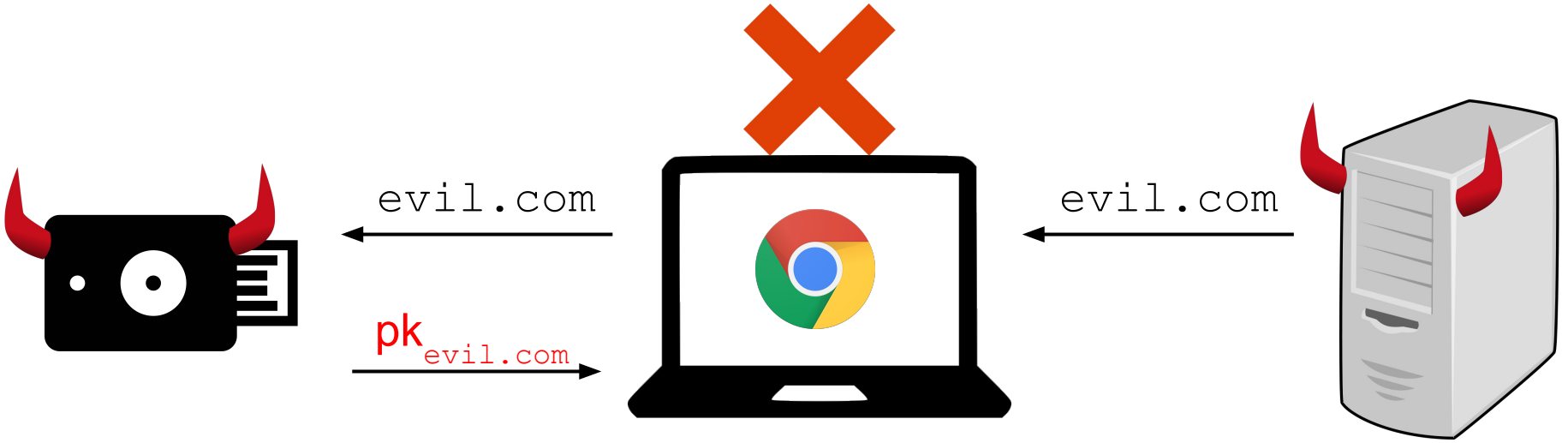
$\mathbb{G} = \langle g \rangle$ is a group of prime order q.



Verifiable Random Functions [MRV99], [DY05], [GRPV18]

✓**Unforgeable:** The browser cannot forge signatures under VIF-generated public keys.

Browser verifies public key



$pk_{evil.com} \leftarrow f(sk_{github.com})$

True2F protocol steps

✓0. Initialization (after purchasing a token)

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

2. Authentication (logging into an account)

True2F protocol steps

✓0. Initialization (after purchasing a token)

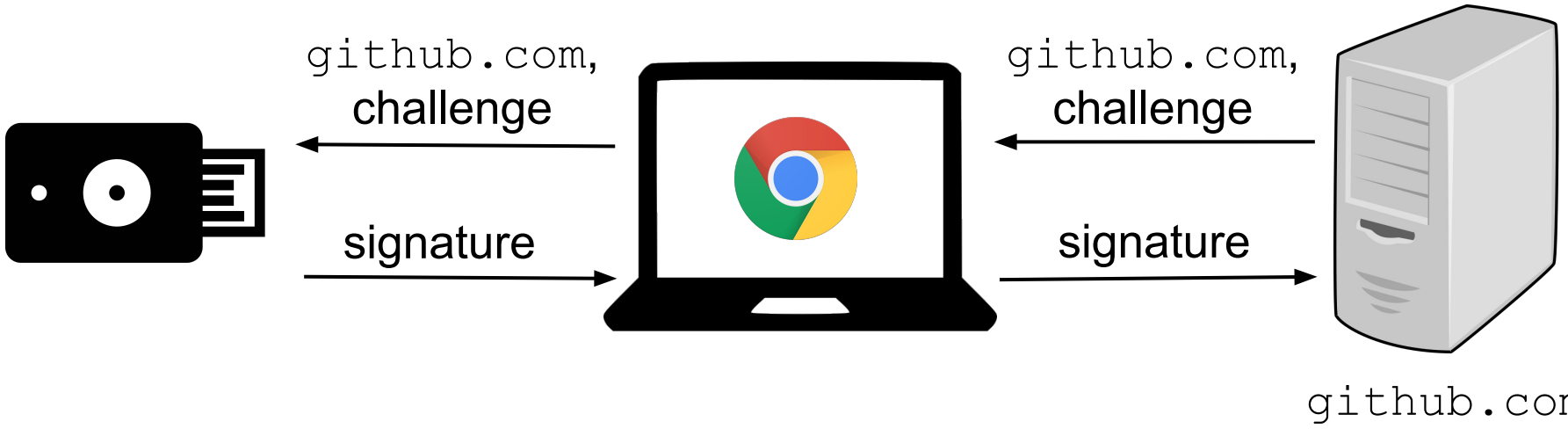
✓1. Registration (associating a token with an account)

2. Authentication (logging into an account)

Approach: Both browser and token contribute randomness to the protocol.

Step #2: U2F Authentication

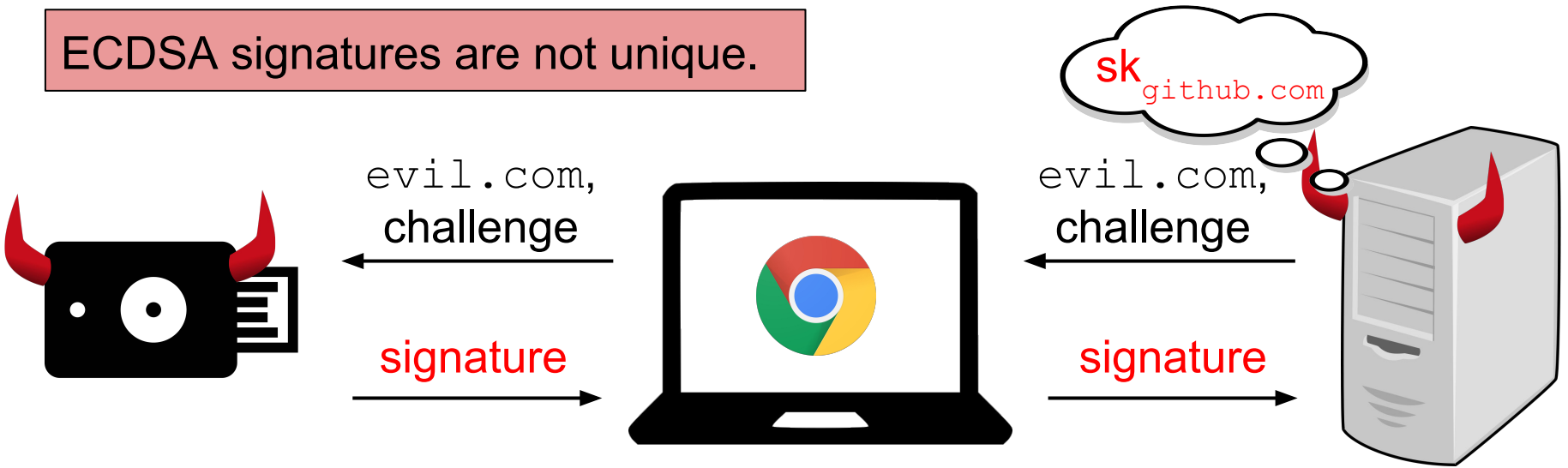
Log into an account.



Supply-chain attack on U2F authentication

Supply-chain attack on U2F authentication

ECDSA signatures are not unique.



Hide $sk_{github.com}$ in signature

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

Firewalled ECDSA Signatures

Two ideas:

1. The token and browser use **collaborative key generation** to generate a signing nonce.
2. Because of ECDSA malleability, signatures are **re-randomized** by the browser.

… see paper for details.

[AMV15], [MS15], [DMS16]

True2F protocol steps

✓0. Initialization (after purchasing a token)

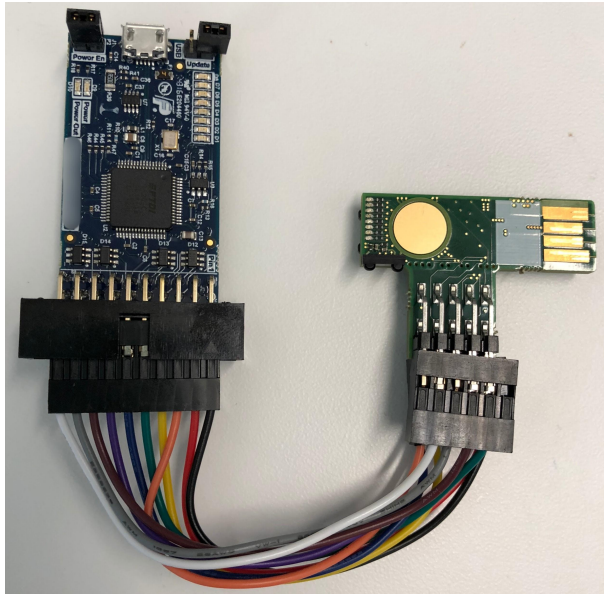
✓1. Registration (associating a token with an account)

✓2. Authentication (logging into an account)

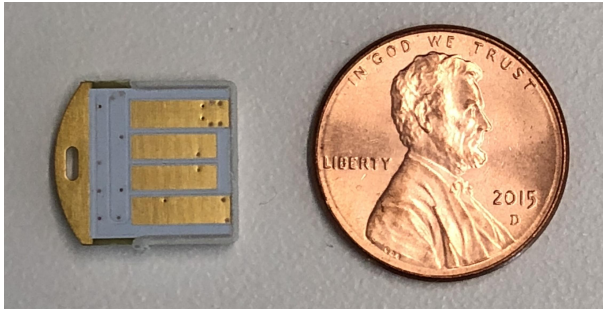
Other contributions (see paper)

- 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 browser
  - Cache Verifiable Random Function outputs at the browser

True2F implementation



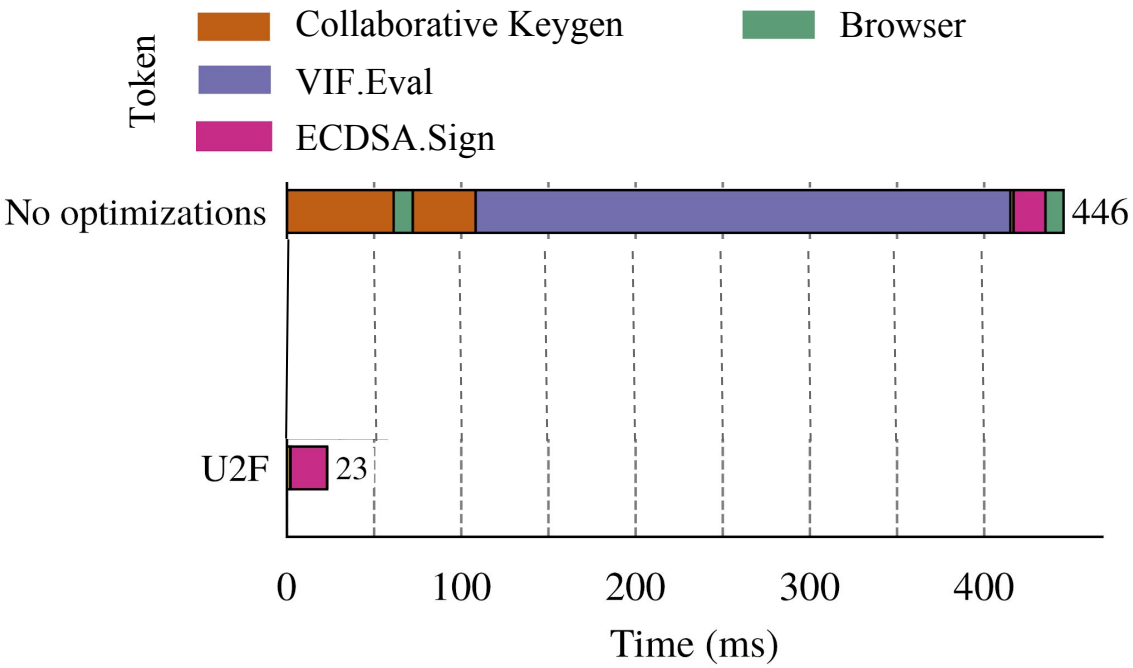
Google development board running True2F.



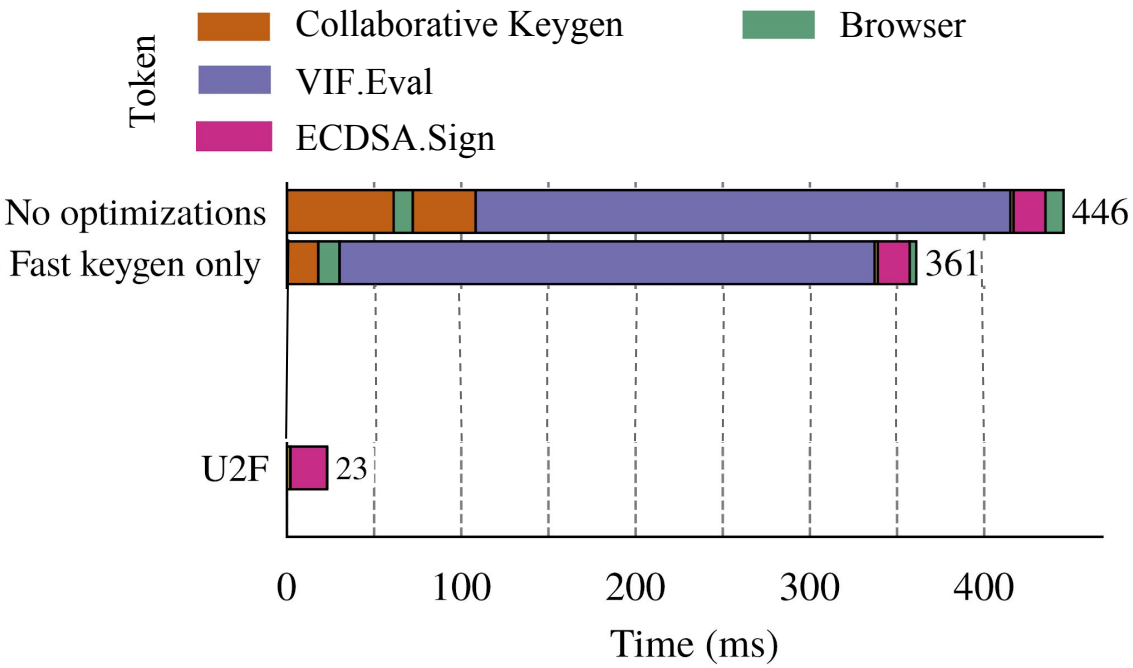
Google production USB token with same hardware specs.

ARM SC-300 processor clocked at 24 MHz

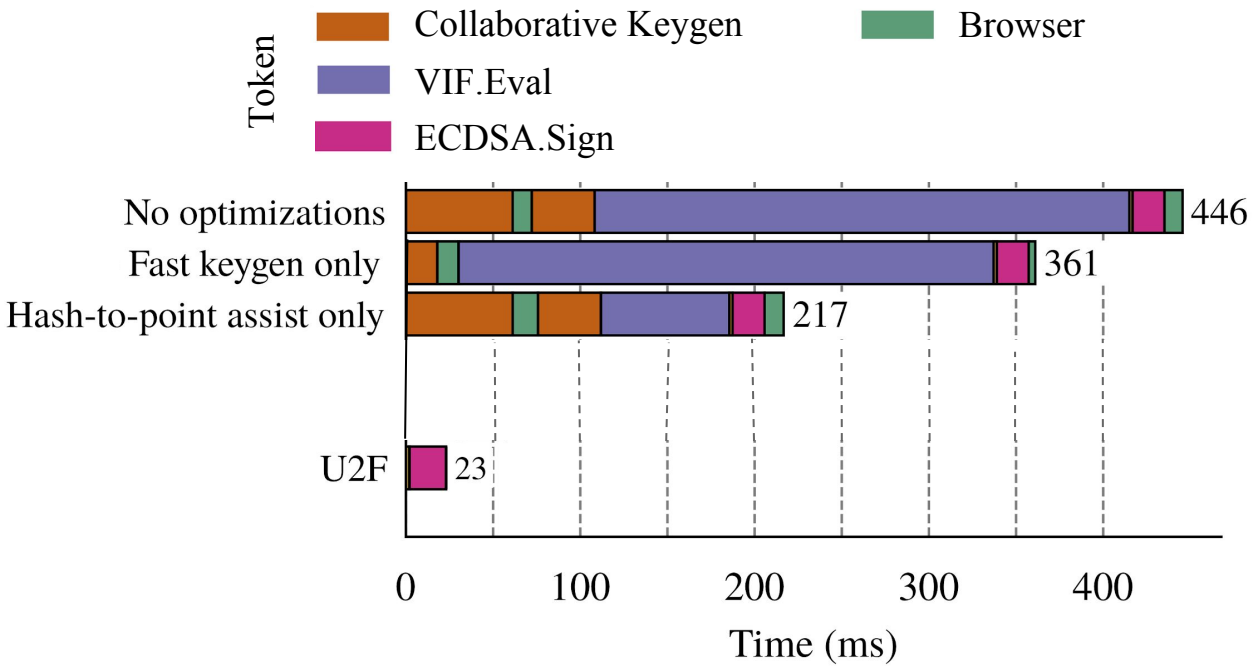
True2F imposes minimal authentication overhead



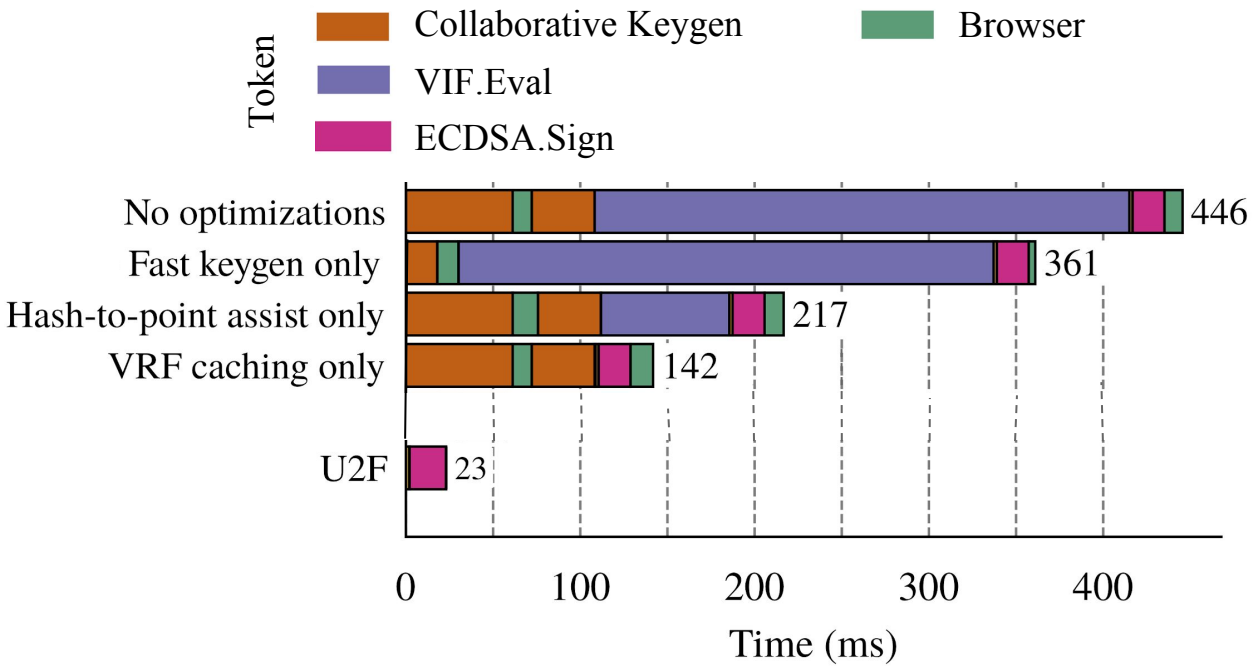
True2F imposes minimal authentication overhead



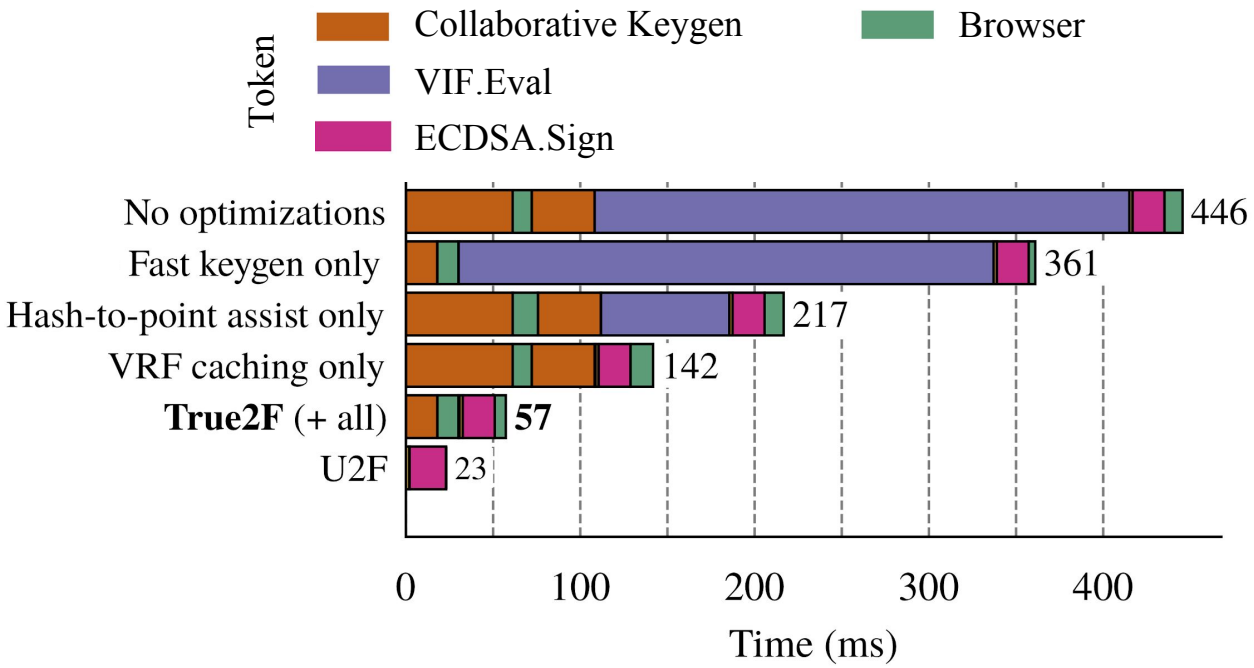
True2F imposes minimal authentication overhead



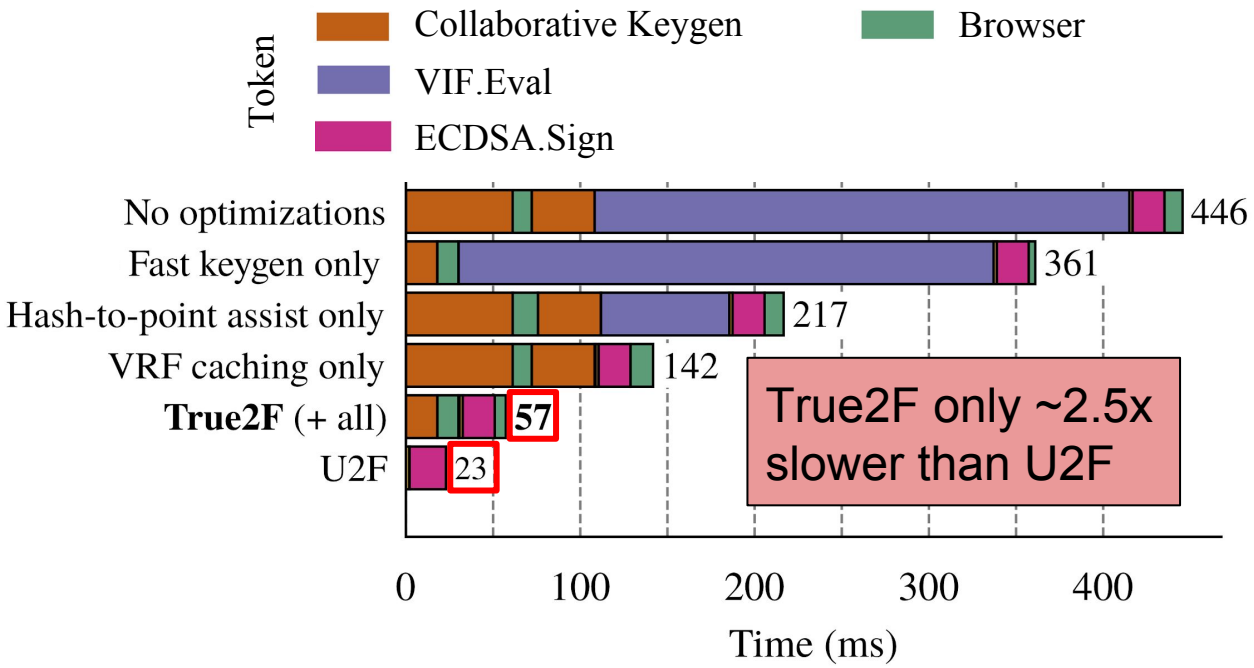
True2F imposes minimal authentication overhead



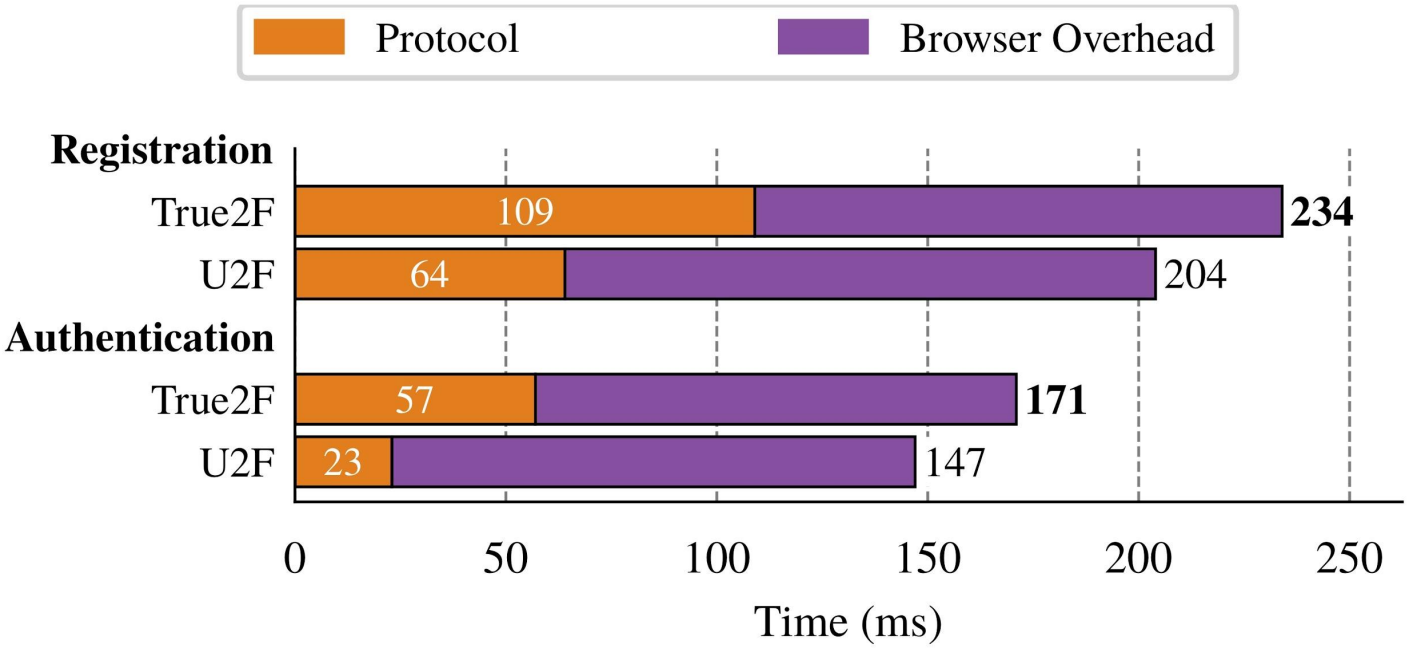
True2F imposes minimal authentication overhead



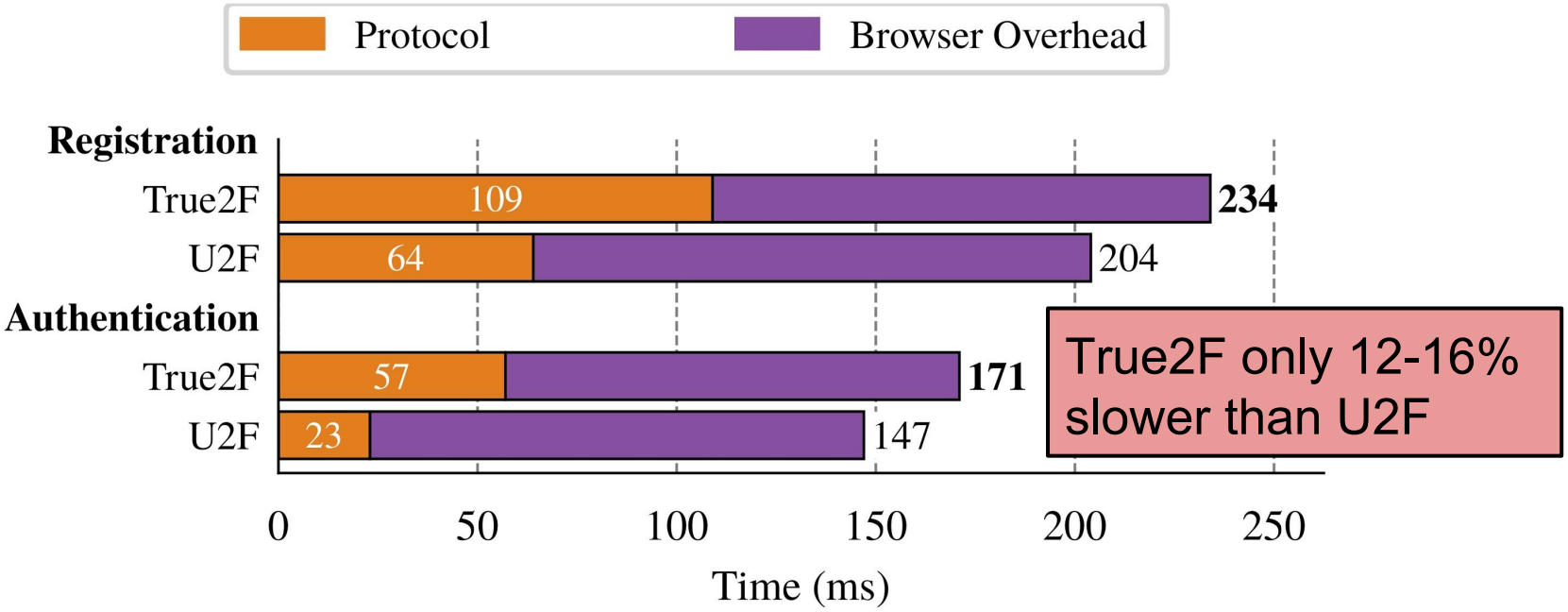
True2F imposes minimal authentication overhead



Comparatively small end-to-end slowdown



Comparatively small end-to-end slowdown



True2F: don’t settle for untrustworthy hardware

True2F

- Augments U2F to protect against **backdoored tokens**
- **Backwards-compatible** with existing U2F servers

Practical to deploy: performant on commodity hardware tokens

Next step: **FIDO standards body**

Emma Dauterman
edauterman@cs.stanford.edu
https://arxiv.org/abs/1810.04660
Paper to appear at Oakland 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.

[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.

[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