True2F: Backdoor-resistant authentication tokens

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

Stanford and Google

IEEE Security & Privacy 2019

U2F: Effective hardware 2FA

fido ALLIANCE

U2F: Effective hardware 2FA

KrebsonSecurity

In-depth security news and investigation

23 JUL 18

Google: Security Keys Neutralized Employee Phishing

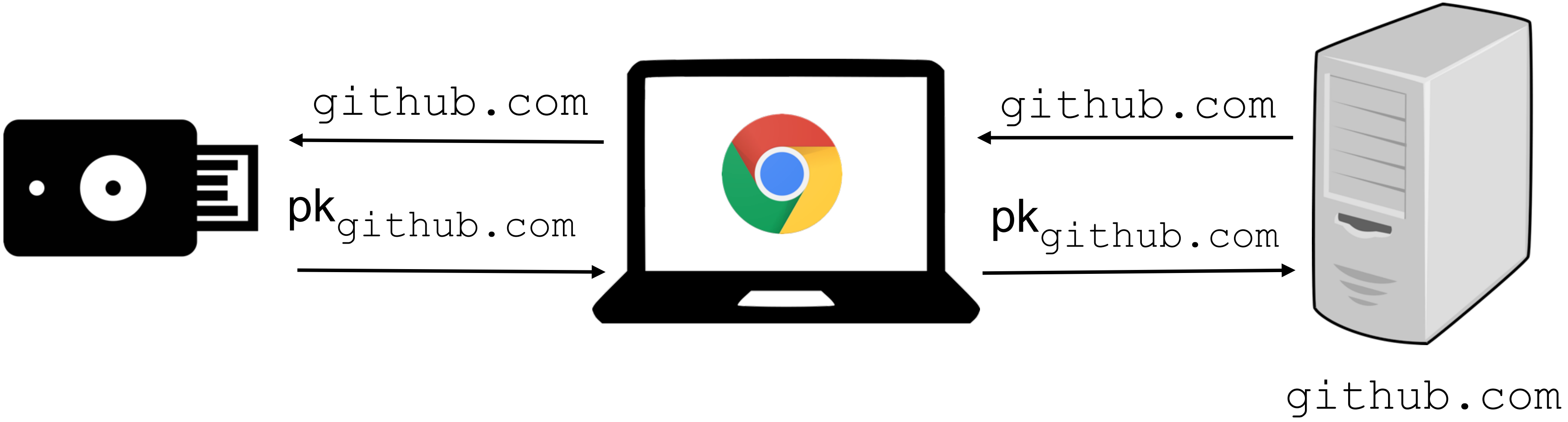
Google has not had any of its 85,000+ employees successfully phished on their work-related accounts since early 2017, when it began requiring all employees to use physical Security Keys in place of passwords and one-time codes, the company told KrebsOnSecurity.

U2F protocol steps

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

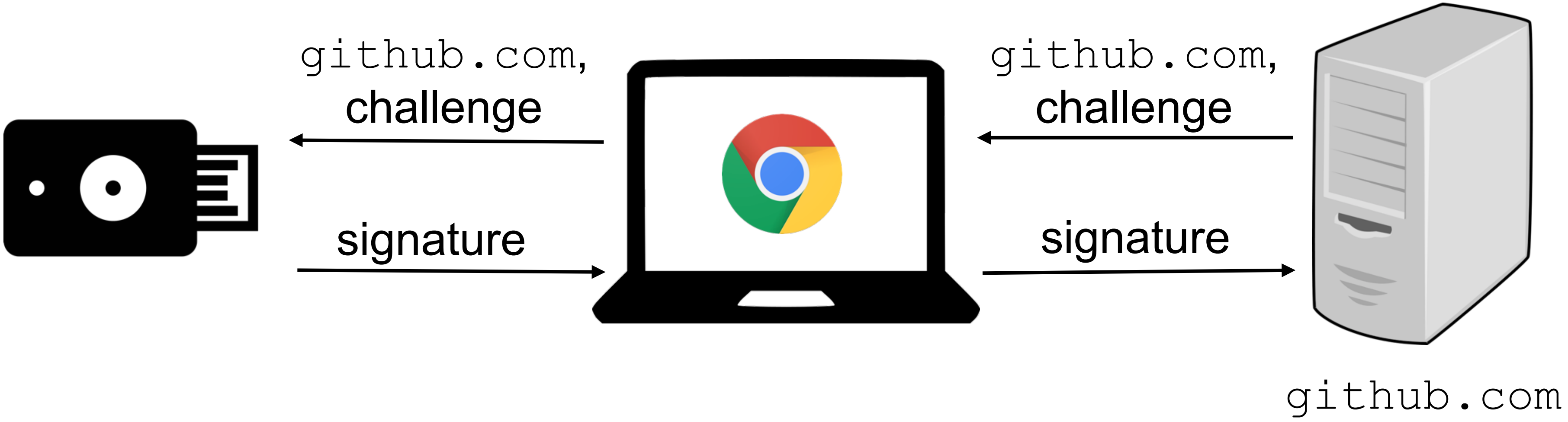
U2F Step #1: Registration

Associate a token with an account.



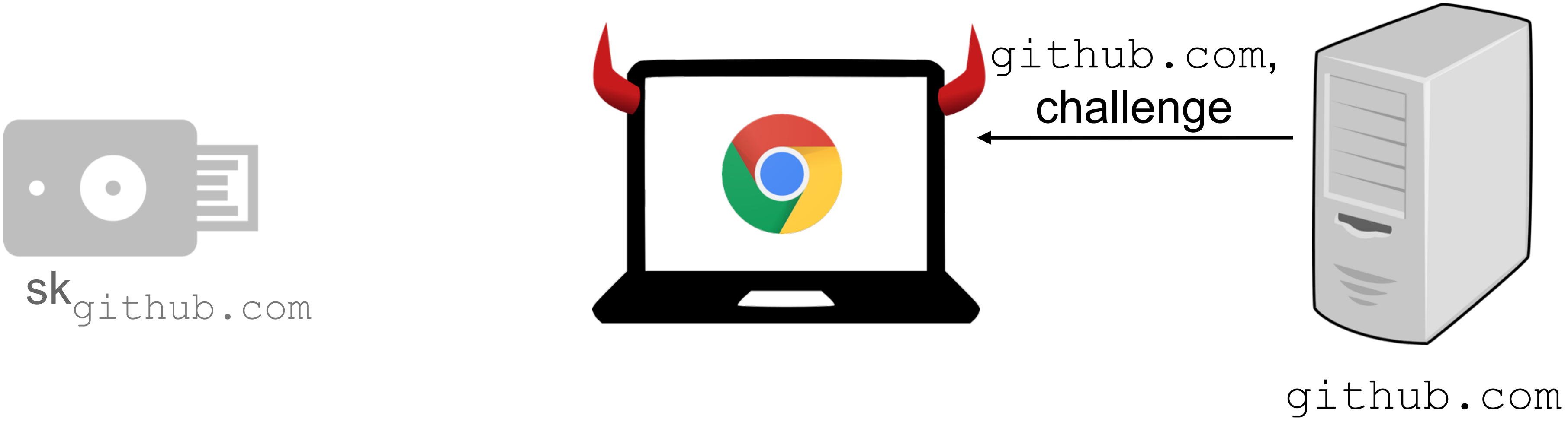
U2F Step #2: Authentication

Log into an account.

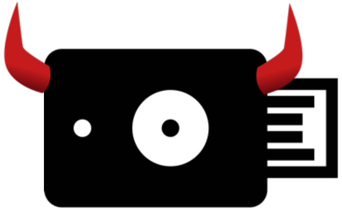


U2F defends against phishing and browser compromise

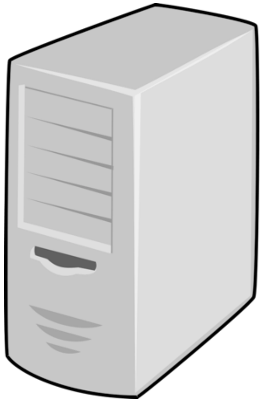
Even if malware takes over your browser, it can't authenticate without the token.



… but what about vulnerabilities in the token itself?



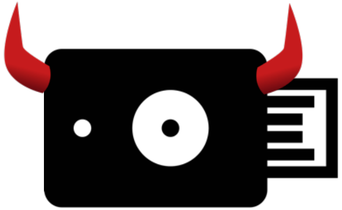
$sk_{\texttt{github.com}}$



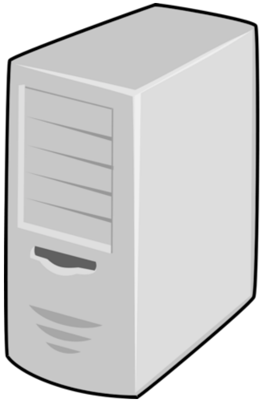
github.com

… but what about vulnerabilities in the token itself?

1. Implementation bugs
2. Supply-chain tampering



$sk_{\texttt{github.com}}$



`github.com`

Security threat #1: Implementation bugs in token

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

[NSS+17]

Security threat #1: Implementation bugs in token

ars TECHNICA SUBSCRIBE SIGN IN

COMPLETI

Milli

crip

Factoriz
data.

DAN GOODI

BANK INFO SECURITY®

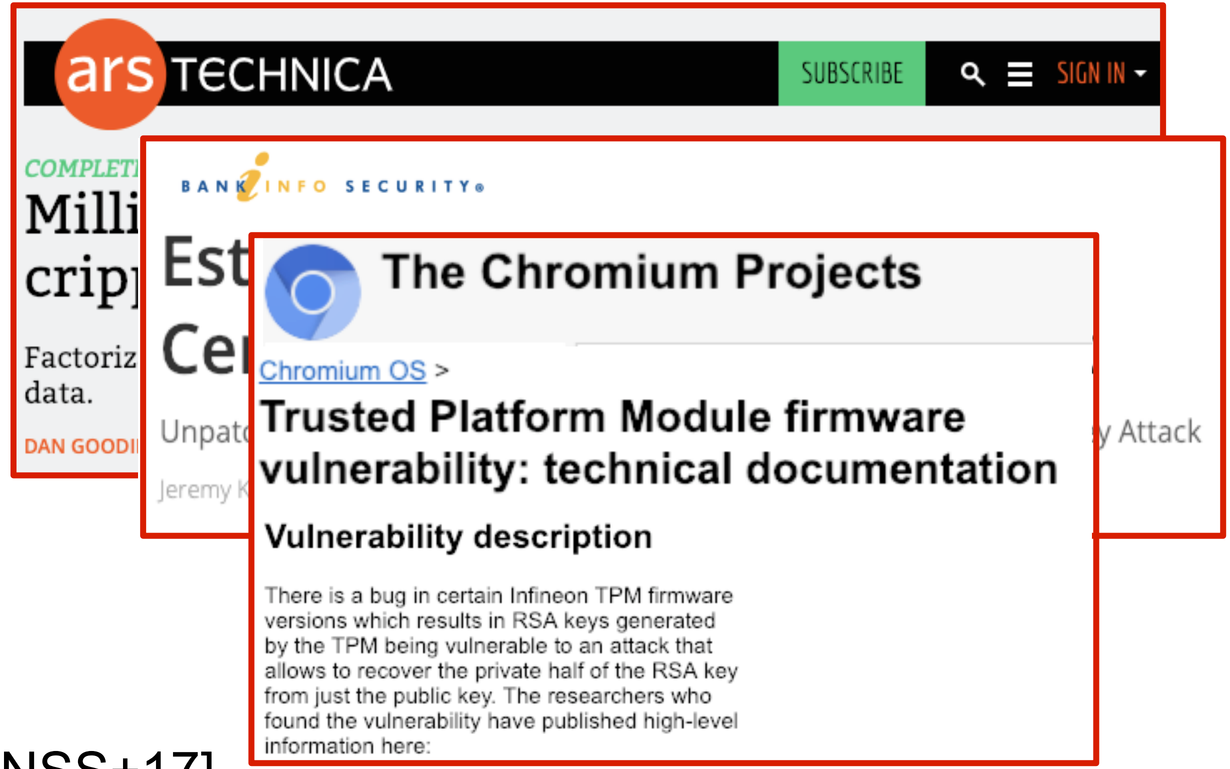
Estonia Invalidates Digital Certificates Over Crypto Crack

Unpatched Infineon Chip Peril as Researchers Speed Up Encryption Key Attack

Jeremy Kirk (jeremy_kirk) • November 8, 2017 0 Comments

[NSS+17]

Security threat #1: Implementation bugs in token

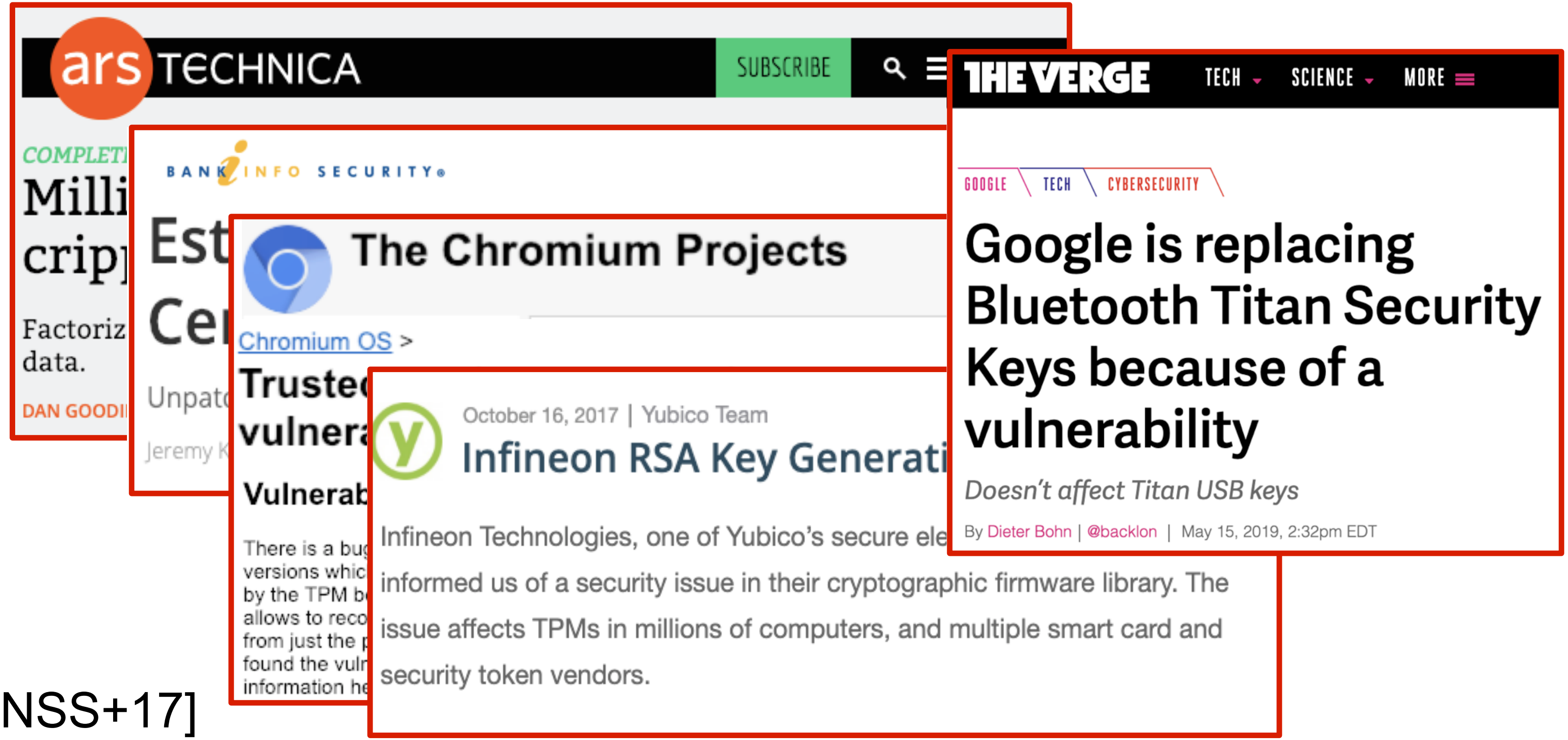


[NSS+17]

Security threat #1: Implementation bugs in token

[NSS+17]

Security threat #1: Implementation bugs in token



[NSS+17]

Security threat #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

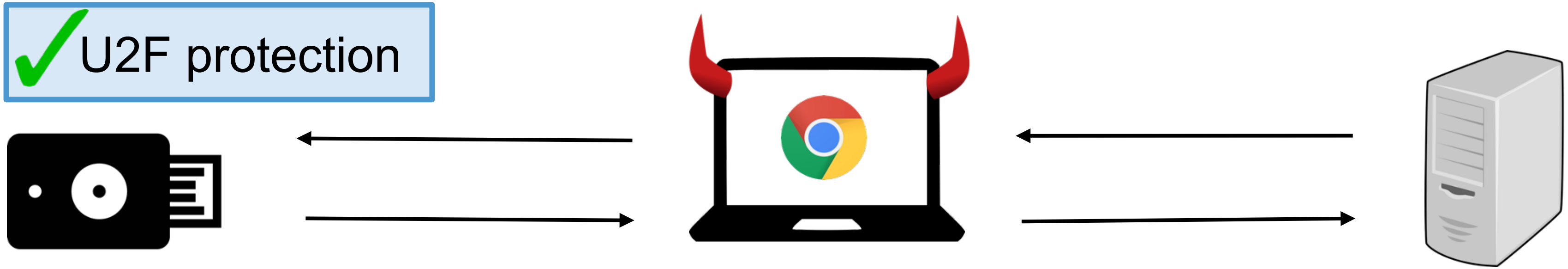
CHINA | By Joseph Cox | Aug 31 2018, 5:05am

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.

True2F: U2F protections + faulty-token protection

True2F: U2F protections + faulty-token protection



Browser learns no secrets.

True2F: U2F protections + faulty-token protection

✓U2F protection

Browser learns no secrets.

✓True2F addition: Faulty-token protection

Browser enforces correct behavior
to prevent token leaking secrets.

True2F: U2F protections + faulty-token protection

Goals:

- Augment U2F to protect against **faulty tokens**
  - Same protections as U2F even if token is buggy or backdoored
- **Backwards-compatible** with U2F server
  - Only requires changes to token and browser, not server
- **Practical** on commodity hardware tokens
  - Evaluated on Google hardware

True2F: U2F protections + faulty-token protection

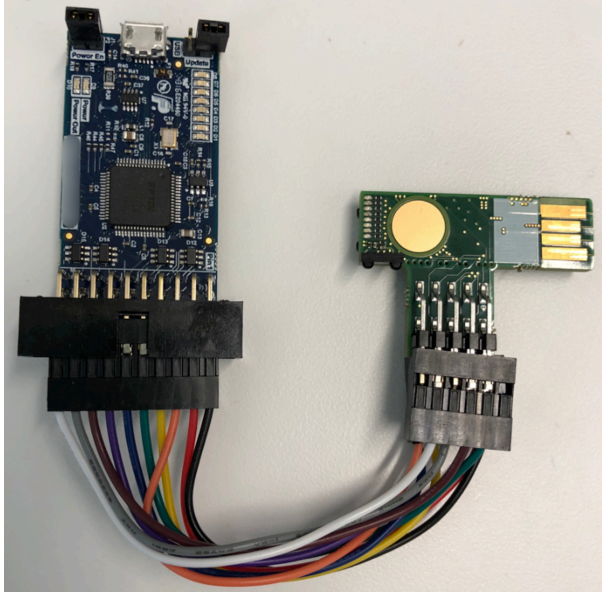
<u>Goals:</u>

- Augment U2F to protect against **faulty tokens**
  - Same protections as U2F even if token is buggy or backdoored
- **Backwards-compatible** with U2F server
  - Only requires changes to token and browser, not server
- **Practical** on commodity hardware tokens
  - Evaluated on Google hardware

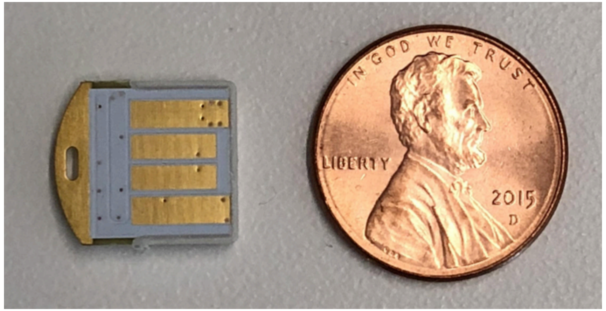
<u>Design principles:</u>

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

True2F implementation



Google development board running True2F.



Google production USB token with same hardware specs.

ARM SC-300 processor clocked at 24 MHz

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) [New]
1. Registration (associating a token with an account) [Modified]
2. Authentication (logging into an account) [Modified]

True2F protocol steps

0. Initialization (after purchasing a token) [New]

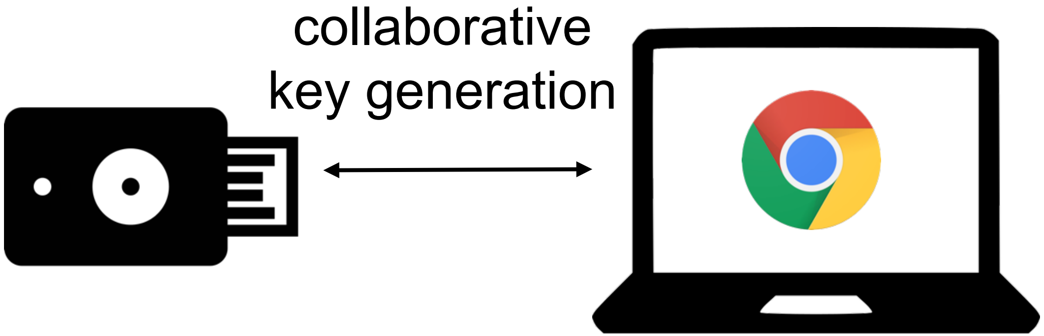
- ➔ **Ensure token master secret incorporates good randomness.**

1. Registration (associating a token with an account) [Modified]

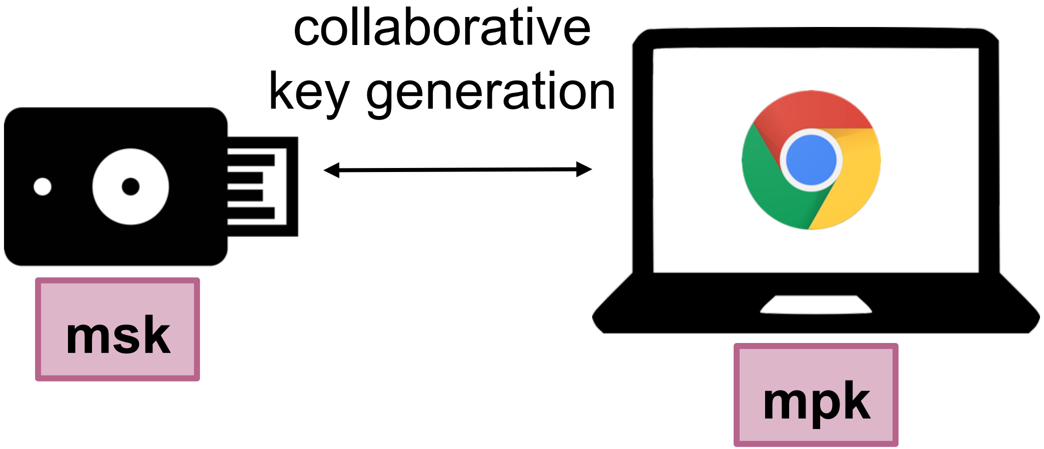
2. Authentication (logging into an account) [Modified]

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

Step #0: Initialization



Step #0: Initialization



Initialization: Security properties

The token cannot bias mpk.

msk

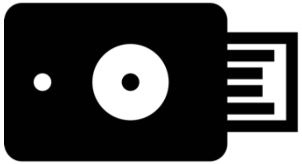
mpk

[GJKR99], [CMBF13]

Initialization: Security properties

The token cannot bias mpk.

The browser learns nothing about msk.



msk

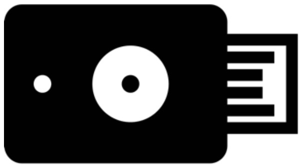
mpk

[GJKR99], [CMBF13]

Initialization properties

The token cannot bias mpk.

The browser learns nothing about msk.



Our protocol reduces the number of group operations by 3x compared to [CMBF13] (see paper).

True2F protocol steps

✓0. Initialization (after purchasing a token) [New]
   ➔ Ensure token master secret incorporates good randomness.

1. Registration (associating a token with an account) [Modified]

2. Authentication (logging into an account) [Modified]

True2F protocol steps

✓ 0. Initialization (after purchasing a token) [New]
   ➔ Ensure token master secret incorporates good randomness.

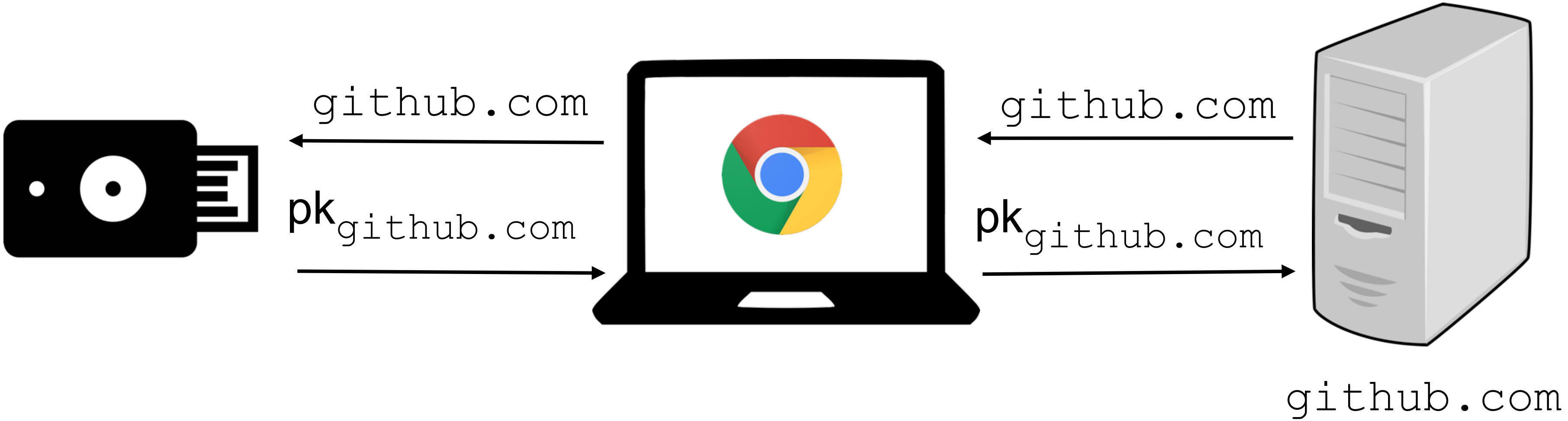
1. Registration (associating a token with an account) [Modified]
   ➔ **Ensure per-site keys generated correctly.**

2. Authentication (logging into an account) [Modified]

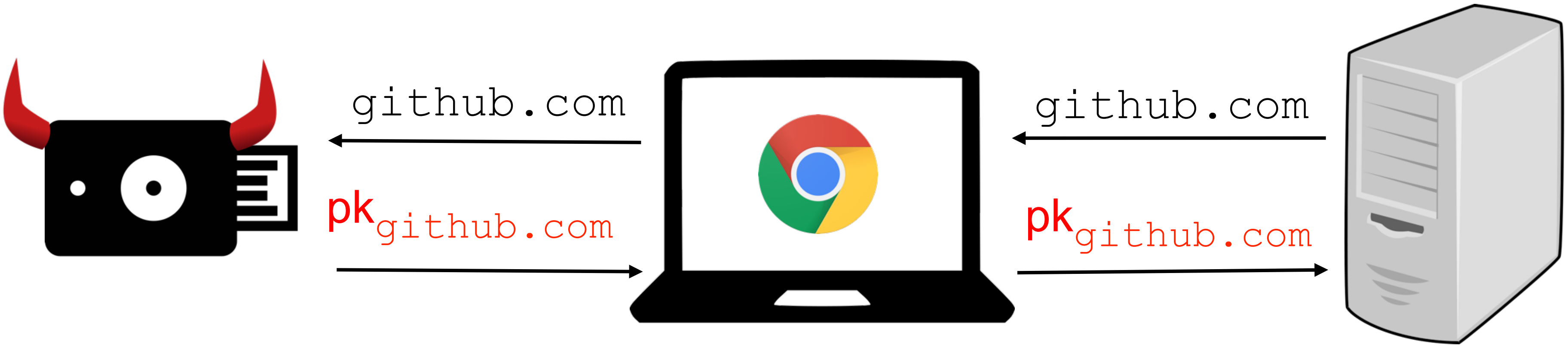
Principle: Browser can verify all deterministic token operations.

Step #1: U2F Registration

Associate a token with an account.



Security threat #1: Implementation bugs in token

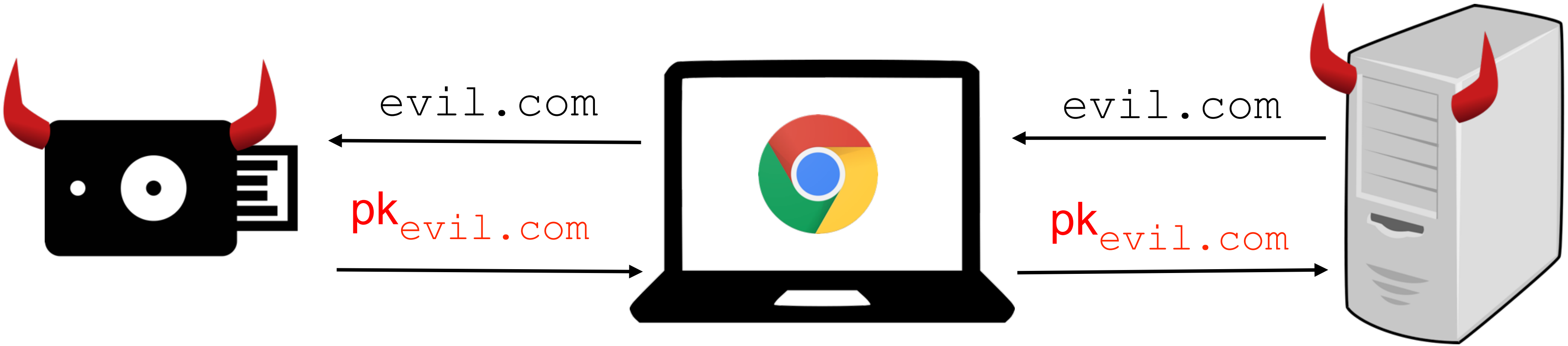


github.com

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

Bad randomness in embedded devices:
[EZJ+14], [LHA+14], [NDWH14], [YRS+09]

Security threat #2: Supply-chain tampering



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

evil.com

$sk_{\texttt{github.com}} \leftarrow f^{-1}(pk_{\texttt{evil.com}})$

Verifiable Identity Families (VIFs)

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

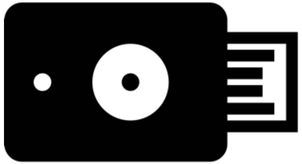
Verifiable Identity Families (VIFs)

Formally, we prove that VIFs are **unique**, **verifiable**, **unlinkable**, and **unforgeable**.

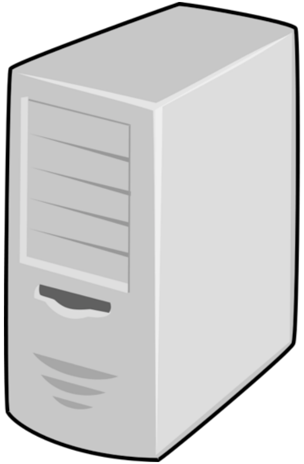
Contribution: Simple (weak) VIF construction

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

msk



mpk



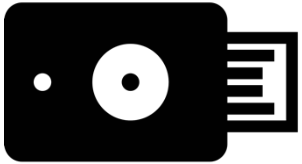
github.com

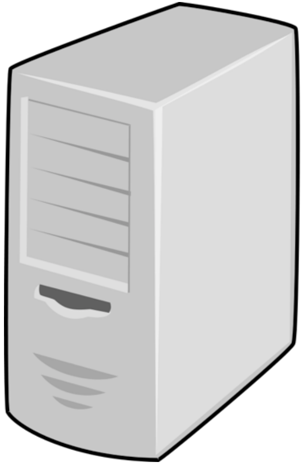
Contribution: Simple (weak) VIF construction

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

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

$\mathsf{msk} = x \in \mathbb{Z}_q$



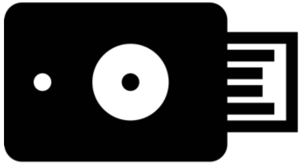


github.com

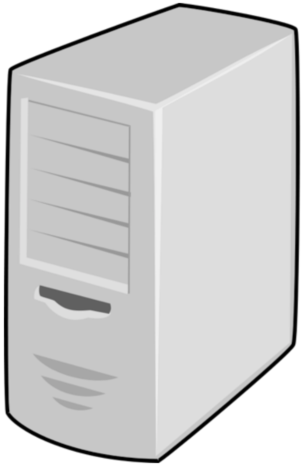
Contribution: Simple (weak) VIF construction

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

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



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



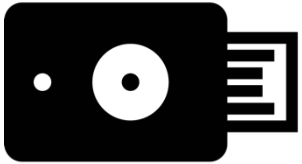
github.com

Contribution: Simple (weak) VIF construction

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

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

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

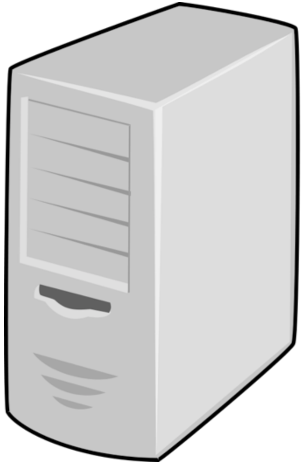


github.com ←

pk →

github.com ←

pk →



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

$\alpha \leftarrow \mathsf{PRF}(k, \texttt{github.com})$
Check if $\mathsf{pk} = X^\alpha$

github.com

Contribution: Simple (weak) VIF construction

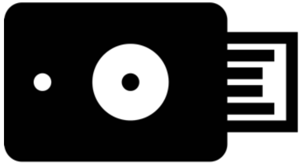
$\mathbb{G} = \langle g \rangle$ is a group of prime order q.

$\mathsf{msk} = x \in \mathbb{Z}_q$

$k = H(X)$

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

$k = H(X)$

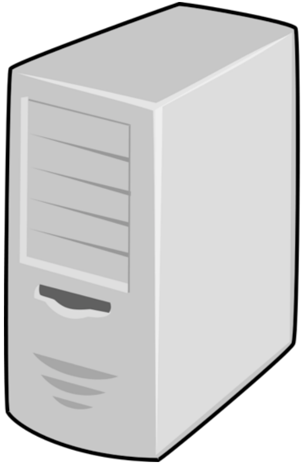


`github.com`

pk

`github.com`

pk



$\alpha \leftarrow \mathsf{PRF}(k, \texttt{github.com})$

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

$\alpha \leftarrow \mathsf{PRF}(k, \texttt{github.com})$

Check if $\mathsf{pk} = X^\alpha$

`github.com`

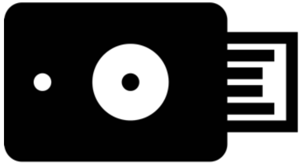
✓ **Unique:** The token can produce the unique keypair for `github.com`.

Contribution: Simple (weak) VIF construction

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

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

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

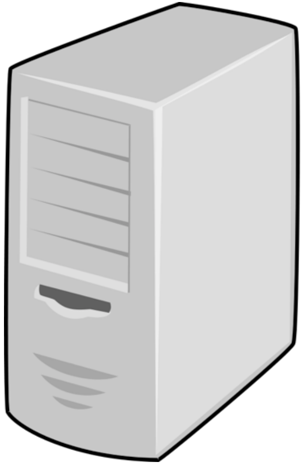


`github.com`

pk

`github.com`

pk



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

$\alpha \leftarrow \mathsf{PRF}(k, \texttt{github.com})$
Check if $\mathsf{pk} = X^\alpha$

`github.com`

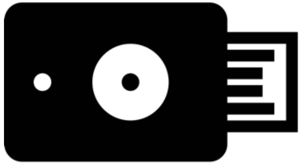
✓ **Verifiable:** The token can prove to the browser that $\mathsf{pk}_{\texttt{github.com}}$ is really the unique public key for `github.com`.

Contribution: Simple (weak) VIF construction

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

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

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

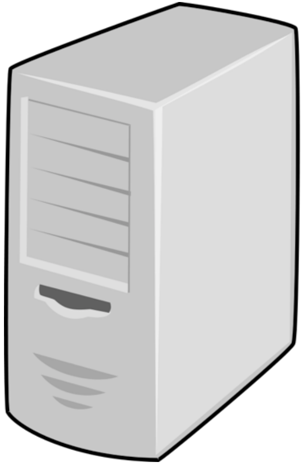


`github.com` ←

pk →

`github.com` ←

pk →



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

$\alpha \leftarrow \mathsf{PRF}(k, \texttt{github.com})$
Check if $\mathsf{pk} = X^{\alpha}$

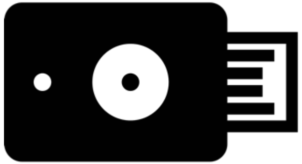
`github.com`

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

Contribution: Simple (weak) VIF construction

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

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



`github.com`

pk

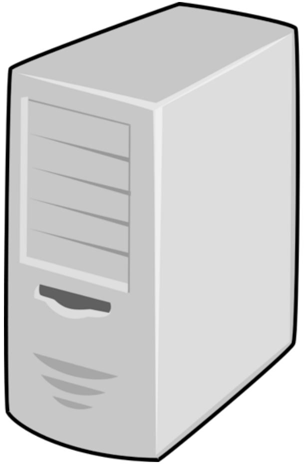
$\alpha \leftarrow \mathsf{PRF}(k, \texttt{github.com})$
$(\mathsf{sk}, \mathsf{pk}) \leftarrow (\alpha x, g^{\alpha x})$

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

$\alpha \leftarrow \mathsf{PRF}(k, \texttt{github.com})$
Check if $\mathsf{pk} = X^\alpha$

`github.com`

pk



`github.com`

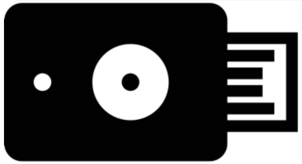
✓**Weak unlinkability:** `github.com` cannot distinguish $\mathsf{pk}_{\texttt{github.com}}$ from a random ECDSA public key.

Contribution: Simple (weak) VIF construction

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

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

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

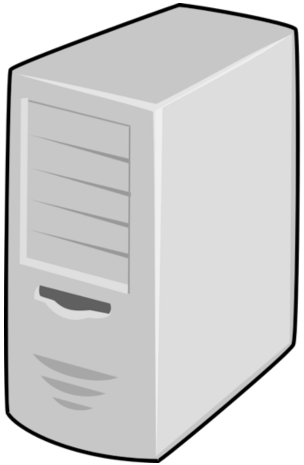


$\xleftarrow{\texttt{github.com}}$

$\xrightarrow{\mathsf{pk}}$

$\xleftarrow{\texttt{github.com}}$

$\xrightarrow{\mathsf{pk}}$



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

$\alpha \leftarrow \mathsf{PRF}(k, \texttt{github.com})$
Check if $\mathsf{pk} = X^\alpha$

`github.com`

❌ **Full unlinkability:** Informally, browser cannot generate public keys without the token (see paper).

True2F protocol steps

✓0. Initialization (after purchasing a token) [New]
   - ➔ Ensure token master secret incorporates good randomness.

✓1. Registration (associating a token with an account) [Modified]
   - ➔ Ensure per-site keys generated correctly.

2. Authentication (logging into an account) [Modified]

True2F protocol steps

✓0. Initialization (after purchasing a token) [New]
   ➔ Ensure token master secret incorporates good randomness.

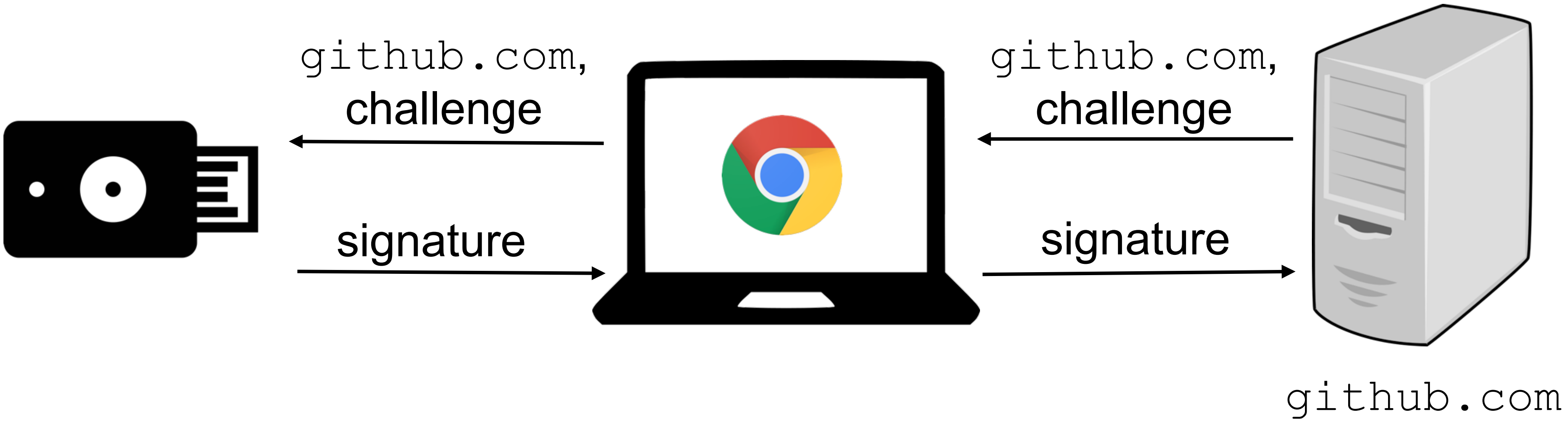
✓1. Registration (associating a token with an account) [Modified]
   ➔ Ensure per-site keys generated correctly.

2. Authentication (logging into an account) [Modified]
   ➔ **Ensure authentication leaks no data.**

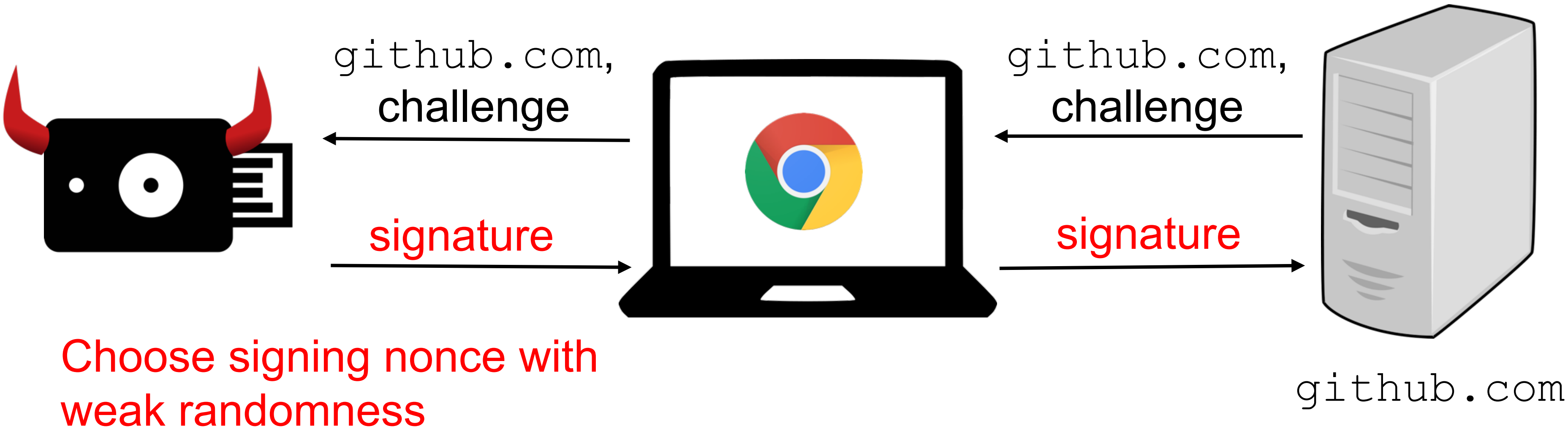
Principle: Both browser and token contribute randomness to the protocol.

Step #2: U2F Authentication

Log into an account.

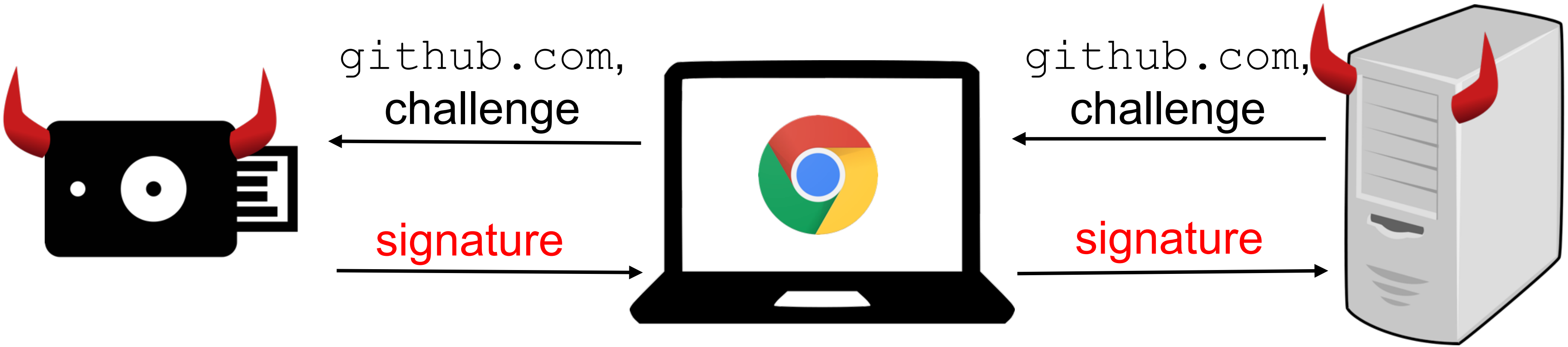


Security threat #1: Implementation bugs in token



Bad randomness in embedded devices:
[EZJ+14], [LHA+14], [NDWH14], [YRS+09]

Security threat #2: Supply-chain tampering



evil.com

Hide $sk_{\texttt{github.com}}$ in signature

Subliminal channels: [Sim84], [Des88]
Unique signatures: [BLS01]

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) [New]
- ➔ Ensure token master secret incorporates good randomness.

✓1. Registration (associating a token with an account) [Modified]
- ➔ Ensure per-site keys generated correctly.

✓2. Authentication (logging into an account) [Modified]
- ➔ Ensure authentication leaks no data.

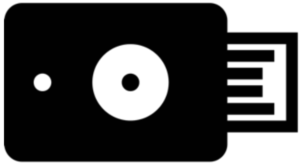
Other contributions (see paper)

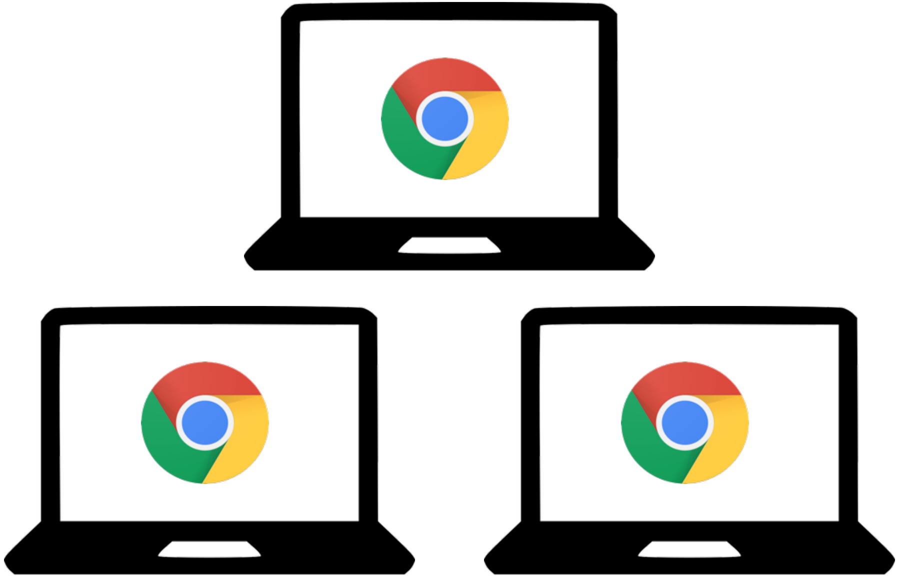
- Cryptographic optimizations tailored to token hardware
  - Offload hash-to-point to the browser
  - Cache Verifiable Random Function outputs at the browser

- 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

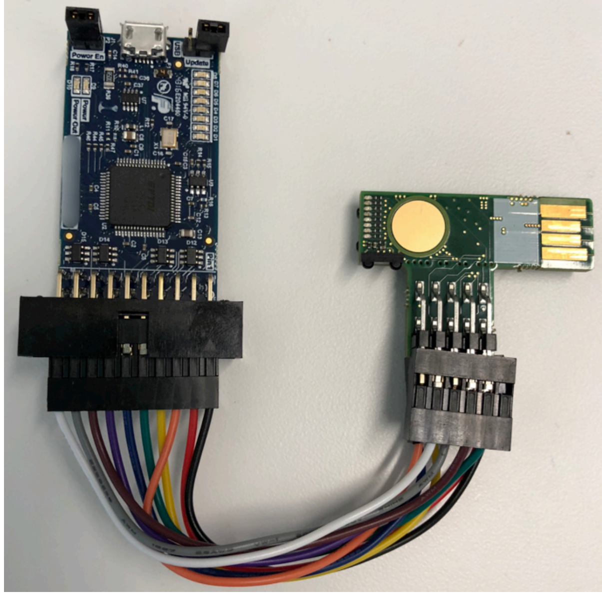
Multiple Browsers

1. Token gives mpk to browser (protect against bugs)
2. Sync mpk across browser instances

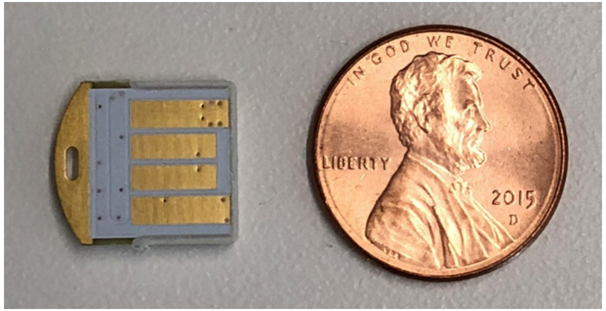




True2F evaluation



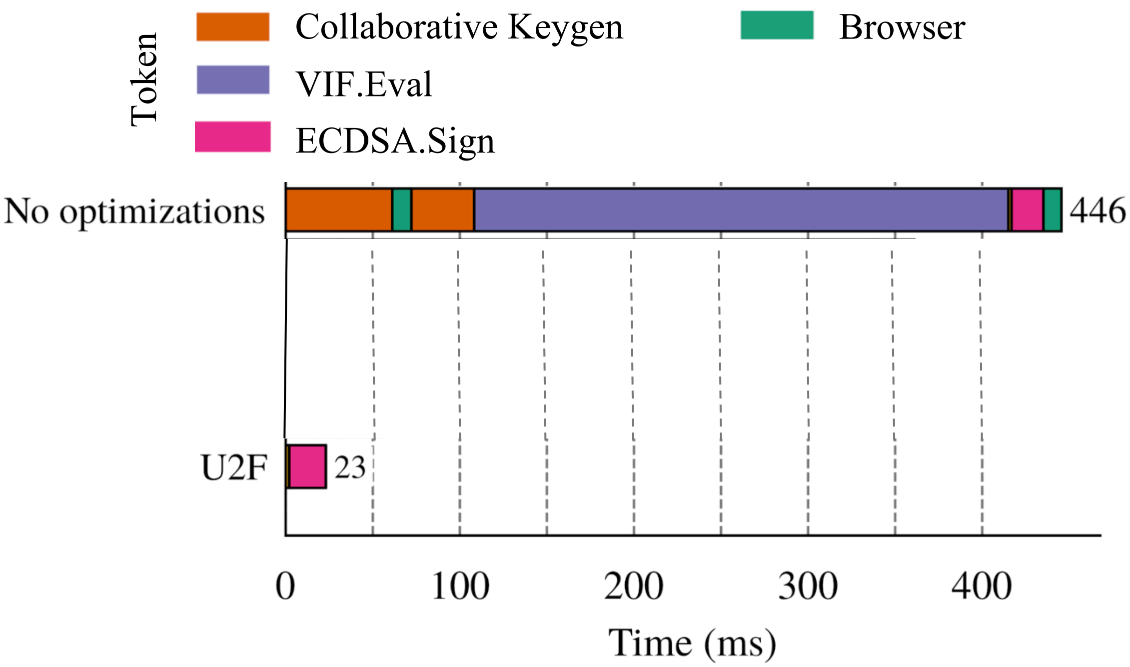
Google development board running True2F.



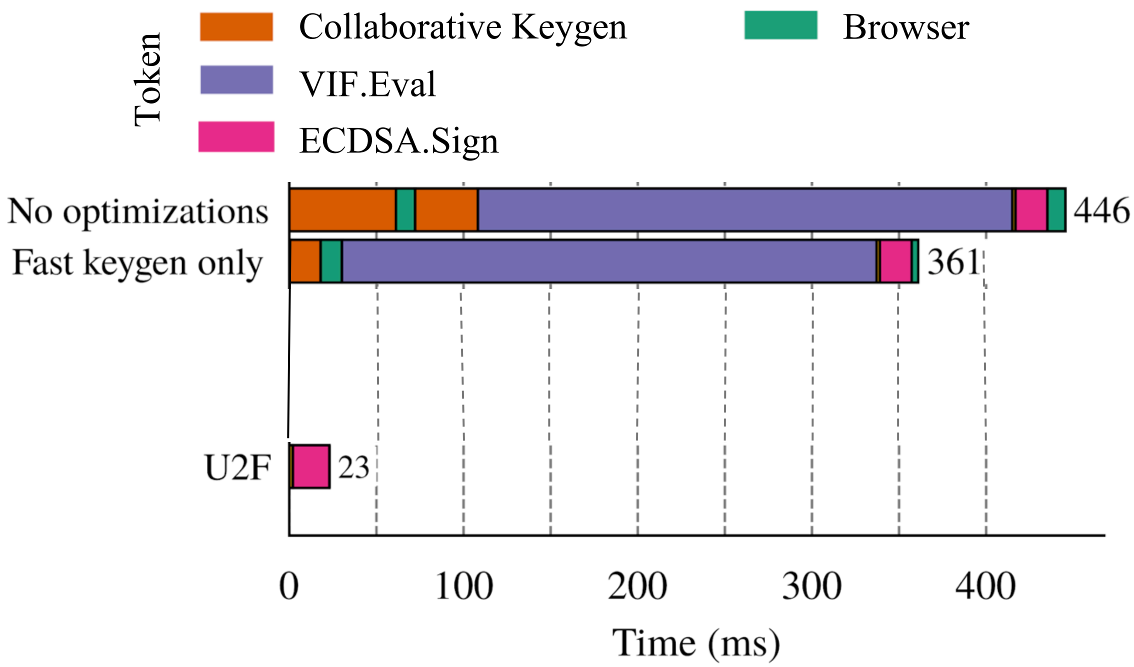
Google production USB token with same hardware specs.

ARM SC-300 processor clocked at 24 MHz

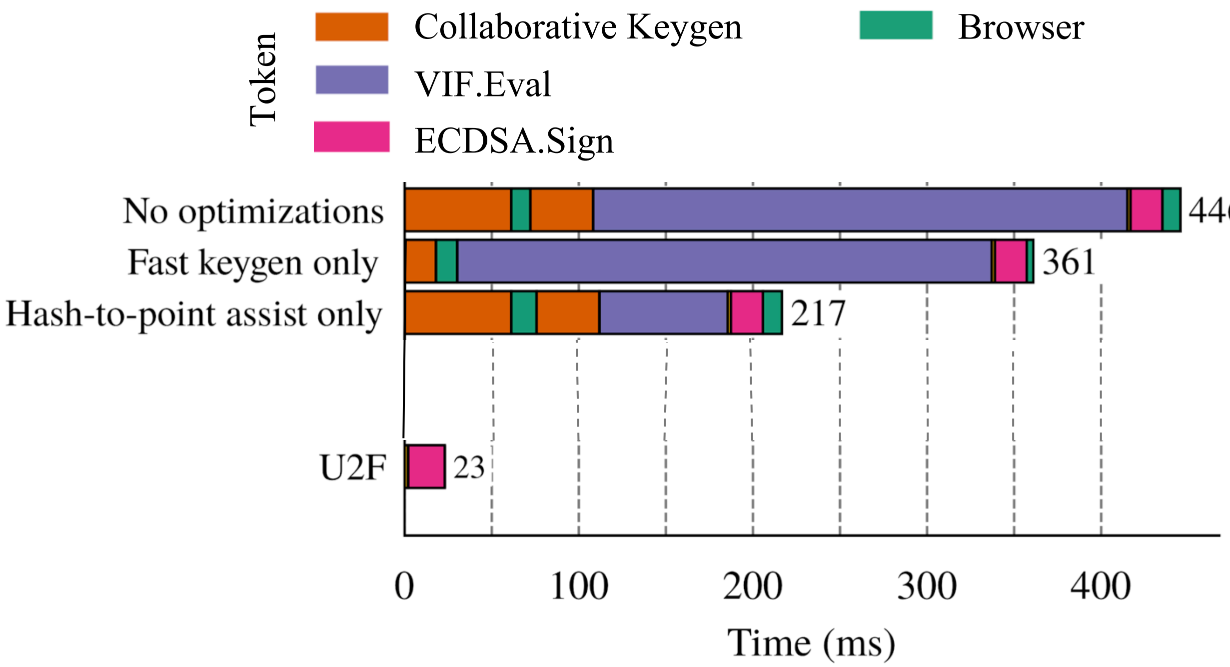
True2F imposes minimal authentication overhead



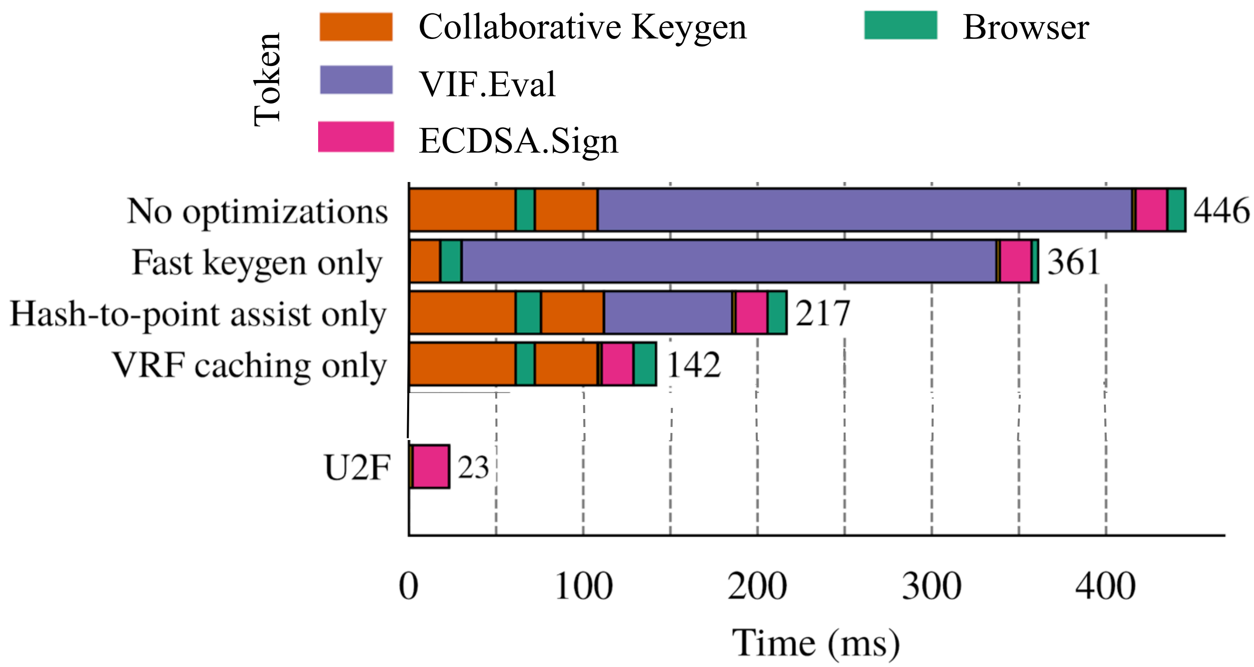
True2F imposes minimal authentication overhead



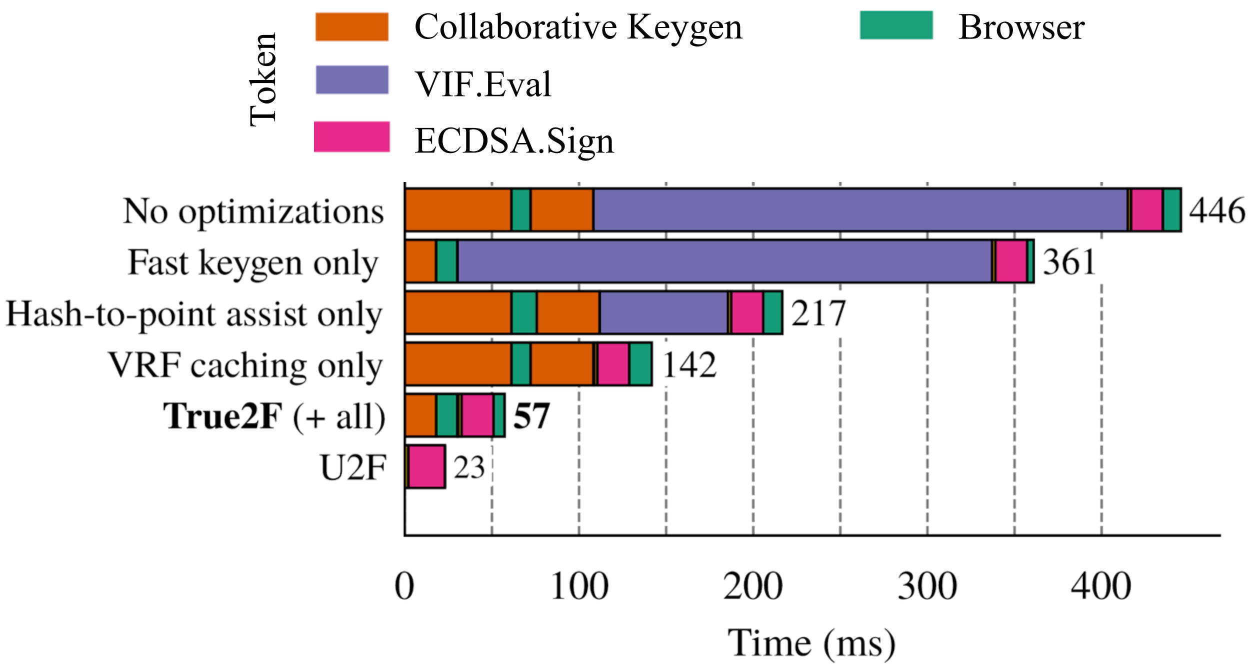
True2F imposes minimal authentication overhead



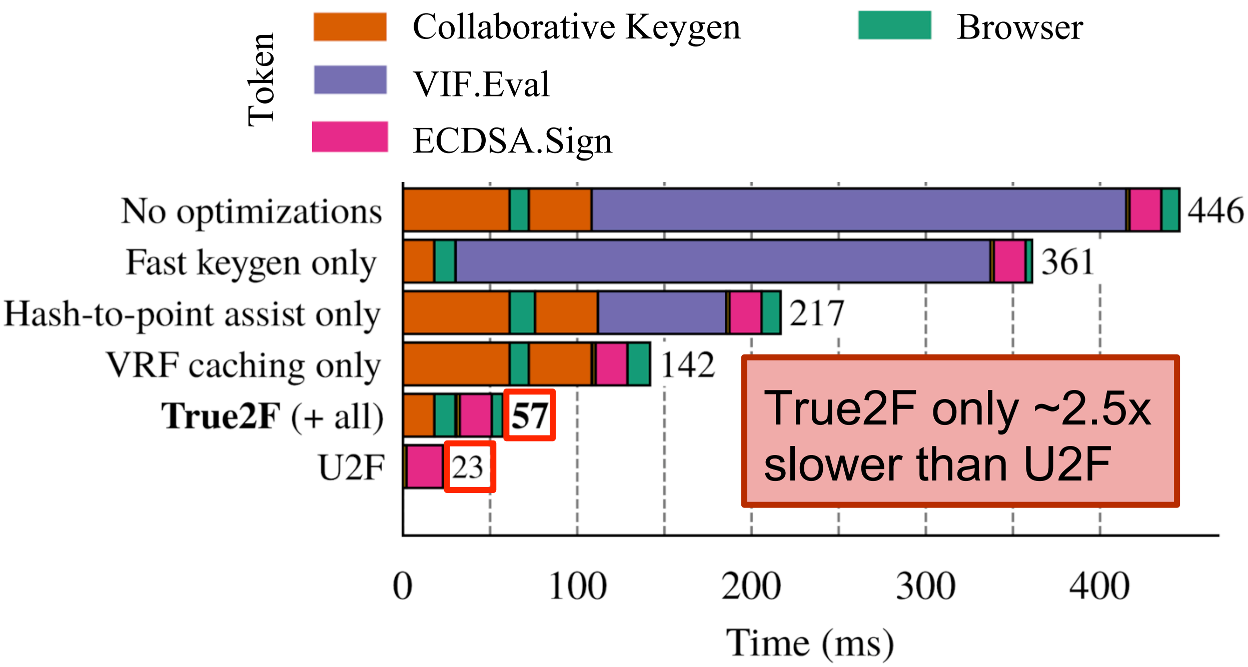
True2F imposes minimal authentication overhead



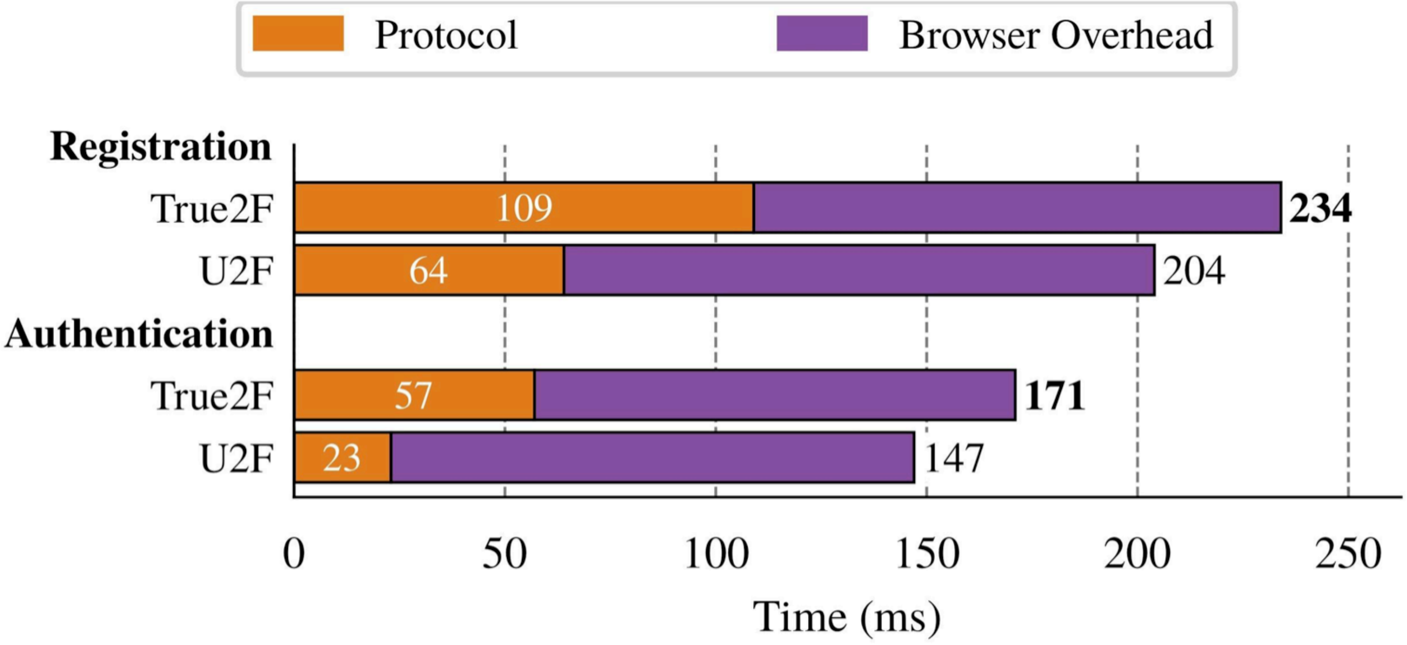
True2F imposes minimal authentication overhead



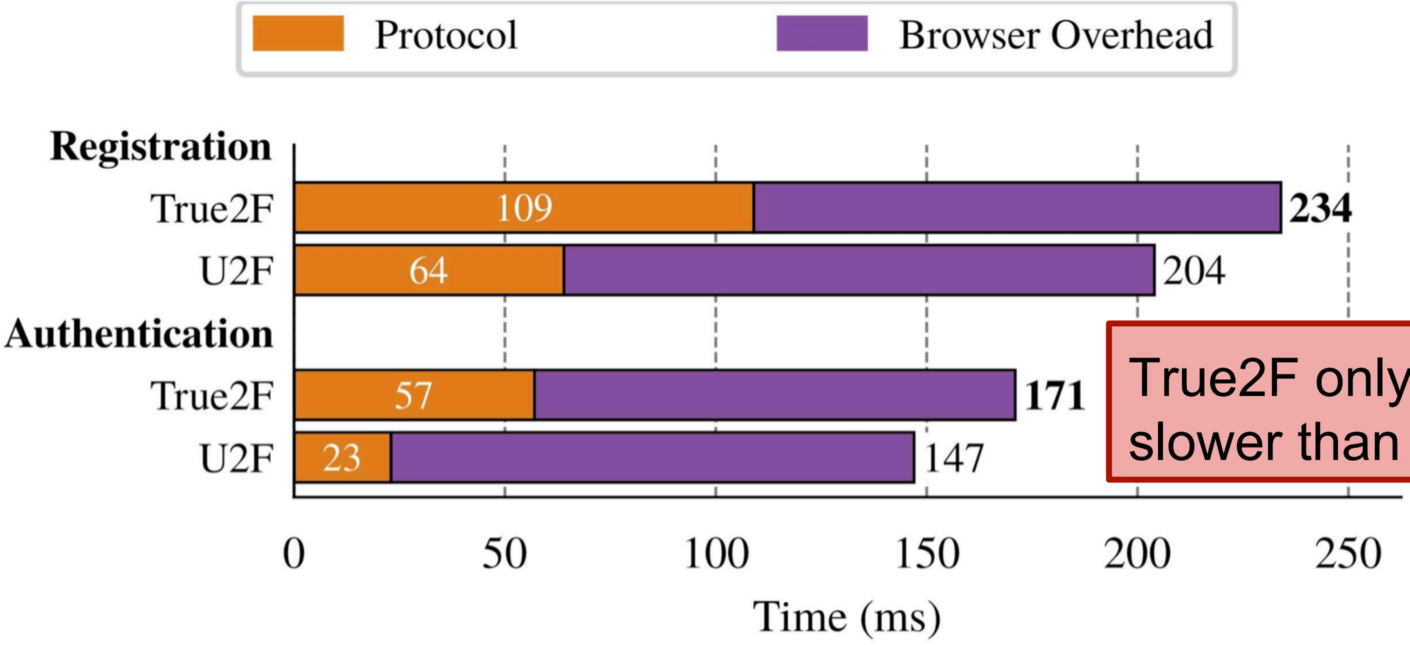
True2F imposes minimal authentication overhead



Comparatively small end-to-end slowdown



Comparatively small end-to-end slowdown



True2F only 12-16% slower than U2F

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 steps: help with FIDO adoption

Emma Dauterman
edauterman@cs.stanford.edu
https://arxiv.org/abs/1810.04660
https://github.com/edauterman/true2f
https://github.com/edauterman/u2f-ref-code

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.

[EZJ+14] A. Everspaugh, Y. Zhai, R. Jellinek, T. Ristenpart, and M. Swift. Not-so-random numbers in virtualized Linux and the Whirlwind RNG. In Security and Privacy. IEEE, 2014.

[GJKR99] Gennaro, Rosario, et al. "Secure distributed key generation for discrete-log based cryptosystems." *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, Berlin, Heidelberg, 1999.

[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.

[LHA+12] A. K. Lenstra, J. P. Hughes, M. Augier, J. W. Bos, T. Kleinjung, and C. Wachter. Ron was wrong, Whit is right. Cryptology ePrint Archive, Report 2012/064, 2012.

[NDWH12] N. Heninger, Z. Durumeric, E. Wustrow, and J. A. Halderman. Mining your Ps and Qs: Detection of widespread weak keys in network devices. In USENIX Security Symposium, volume 8, page 1, 2012.

[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.

[NSS+17] M. Nemec, M. Sys, P. Svenda, D. Klinec, and V. Matyas. The return of Coppersmith’s Attack: Practical factorization of widely used RSA moduli. In CCS, 2017.

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

[YRS+09] S. Yilek, E. Rescorla, H. Shacham, B. Enright, and S. Savage. When private keys are public: results from the 2008 Debian OpenSSL vulnerability. In IMC, 2009.