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
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.

github.com, challenge
signature

Log into an account.

github.com, challenge
signature
github.com
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?
... but what about vulnerabilities in the token itself?

1. Implementation bugs
2. Supply-chain tampering
Security threat #1: Implementation bugs in token

[NSS+17]

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 - 10/15/2017, 4:00 AM
Security threat #1: Implementation bugs in token

[NSS+17]
Security threat #1: Implementation bugs in token

There is a bug in certain Infineon TPM firmware versions which results in RSA keys generated by the TPM being vulnerable to an attack that allows to recover the private half of the RSA key from just the public key. The researchers who found the vulnerability have published high-level information here:

[NSS+17]
Security threat #1: Implementation bugs in token

Infineon Technologies, one of Yubico’s secure element vendors, has informed us of a security issue in their cryptographic firmware library. The issue affects TPMs in millions of computers, and multiple smart card and security token vendors.

NSS+17
Security threat #1: Implementation bugs in token

[NSS+17]
Security threat #2: Supply-chain tampering

ars TECHNOICA

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/5I/NF) Left: Intercepted packages are opened carefully; Right: A “load station” implants a beacon

MOTHERBOARD

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

U2F 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

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

Design principles:
● 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)  
   ➔ Ensure token master secret incorporates good randomness.  [New]

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

collaborative key generation
Step #0: Initialization
Initialization: Security properties

The token cannot bias mpk.

[GJKR99], [CMBF13]
The token cannot bias mpk.
The browser learns nothing about msk.

[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

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

Bad randomness in embedded devices:
[EZJ+14], [LHA+14], [NDWH14], [YRS+09]
Security threat #2: Supply-chain tampering

\[ \text{pk}_{\text{evil.com}} \leftarrow f(\text{sk}_{\text{github.com}}) \]

\[ \text{sk}_{\text{github.com}} \leftarrow f^{-1}(\text{pk}_{\text{evil.com}}) \]
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

\[ G = \langle g \rangle \text{ is a group of prime order } q. \]
Contribution: Simple (weak) VIF construction

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

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

$mpk = X = g^x \in G$

github.com
Contribution: Simple (weak) VIF construction

\( G = \langle g \rangle \) is a group of prime order \( q \).

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

\( \text{mpk} = X = g^x \in G \)

\( k = H(X) \)
Contribution: Simple (weak) VIF construction

\[ G = \langle g \rangle \text{ is a group of prime order } q. \]

\[ \text{msk} = x \in \mathbb{Z}_q \]
\[ k = H(X) \]

\[ \text{mpk} = X = g^x \in G \]
\[ k = H(X) \]

\[ \alpha \leftarrow \text{PRF}(k, \text{github.com}) \]

\[ (\text{sk}, \text{pk}) \leftarrow (\alpha x, g^{\alpha x}) \]

\[ \alpha \leftarrow \text{PRF}(k, \text{github.com}) \]

Check if \( \text{pk} = X^\alpha \)
Contribution: Simple (weak) VIF construction

\[ G = \langle g \rangle \text{ is a group of prime order } q. \]

\[ \text{msk} = x \in \mathbb{Z}_q \]

\[ k = H(X) \]

\[ \alpha \leftarrow \text{PRF}(k, \text{github.com}) \]

\[ (\text{sk}, \text{pk}) \leftarrow (\alpha x, g^{\alpha x}) \]

\[ \text{Check if } \text{pk} = X^\alpha \]

\[ \text{Unique: The token can produce the unique keypair for } \text{github.com.} \]
Contribution: Simple (weak) VIF construction

\[ G = \langle g \rangle \text{ is a group of prime order } q. \]

\[ \text{msk} = x \in \mathbb{Z}_q \]
\[ k = H(X) \]

\[ \alpha \leftarrow \text{PRF}(k, \text{github.com}) \]
\[ (\text{sk}, \text{pk}) \leftarrow (\alpha x, g^{\alpha x}) \]

\[ \text{mpk} = X = g^x \in G \]
\[ k = H(X) \]

Check if \( \text{pk} = X^\alpha \)

\[ \alpha \leftarrow \text{PRF}(k, \text{github.com}) \]

✅ **Verifiable:** The token can prove to the browser that \( \text{pk}_{\text{github.com}} \) is really the unique public key for github.com.
Contribution: Simple (weak) VIF construction

\( G = \langle g \rangle \) is a group of prime order \( q \).

\[
\text{msk} = x \in \mathbb{Z}_q \\
k = H(X)
\]

\[
\alpha \leftarrow \text{PRF}(k, \text{github.com}) \\
(\text{sk}, \text{pk}) \leftarrow (\alpha x, g^{\alpha x})
\]

\[
\text{Check if } \text{pk} = X^\alpha
\]

\( \checkmark \text{ Unforgeable: The browser cannot forge a signature under pk. } \text{github.com} \).
**Contribution:** Simple (weak) VIF construction

\[ G = \langle g \rangle \text{ is a group of prime order } q. \]

\[ \text{msk} = x \in \mathbb{Z}_q \]

\[ k = H(X) \]

\[ \text{mpk} = X = g^x \in G \]

\[ \alpha \leftarrow \text{PRF}(k, \text{github.com}) \]

\[ (\text{sk}, \text{pk}) \leftarrow (\alpha x, g^{\alpha x}) \]

\[ \text{Check if pk} = X^\alpha \]

\[ \checkmark \text{ Weak unlinkability: github.com cannot distinguish pk from a random ECDSA public key.} \]
**Contribution: Simple (weak) VIF construction**

\[ G = \langle g \rangle \text{ is a group of prime order } q. \]
\[ \text{msk} = x \in \mathbb{Z}_q \]
\[ k = H(X) \]

\[ \alpha \leftarrow \text{PRF}(k, \text{github.com}) \]
\[ (sk, pk) \leftarrow (\alpha x, g^{\alpha x}) \]

\[ \text{mpk} = X = g^x \in G \]
\[ k = H(X) \]

\[ \alpha \leftarrow \text{PRF}(k, \text{github.com}) \]
\[ \text{Check if } pk = X^\alpha \]

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

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

github.com, challenge

signature

Choose signing nonce with weak randomness

Bad randomness in embedded devices:
[EZJ+14], [LHA+14], [NDWH14], [YRS+09]
Security threat #2: Supply-chain tampering

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

- **Collaborative Keygen**
- **VIF.Eval**
- **ECDSA.Sign**

No optimizations: 446 ms

U2F: 23 ms
True2F imposes minimal authentication overhead
True2F imposes minimal authentication overhead

- **Collaborative Keygen**
- **VIF.Eval**
- **ECDSA.Sign**

<table>
<thead>
<tr>
<th>Token Type</th>
<th>Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>No optimizations</td>
<td>446</td>
</tr>
<tr>
<td>Fast keygen only</td>
<td>361</td>
</tr>
<tr>
<td>Hash-to-point assist only</td>
<td>217</td>
</tr>
<tr>
<td>U2F</td>
<td>23</td>
</tr>
</tbody>
</table>
True2F imposes minimal authentication overhead
True2F imposes minimal authentication overhead
True2F imposes minimal authentication overhead

True2F only ~2.5x slower than U2F
Comparatively small end-to-end slowdown

- **Registration**
  - True2F: Protocol 109ms, Browser Overhead 125ms, Total 234ms
  - U2F: Protocol 64ms, Browser Overhead 140ms, Total 204ms

- **Authentication**
  - True2F: Protocol 57ms, Browser Overhead 114ms, Total 171ms
  - U2F: Protocol 23ms, Browser Overhead 124ms, Total 147ms
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