Protecting privacy by splitting trust

Henry Corrigan-Gibbs
Stanford University

Based on joint work with: Dan Boneh, Elette Boyle, Emma Dauterman, Niv Gilboa, Yuval Ishai, Dima Kogan, David Mazières, and Dominic Rizzo.
We put a large amount of trust in a small number of entities.

<table>
<thead>
<tr>
<th>Applications</th>
<th>50% of U.S. medical patients</th>
</tr>
</thead>
<tbody>
<tr>
<td>Libraries</td>
<td>60% of web sites</td>
</tr>
<tr>
<td>OS</td>
<td>80% of desktop OS</td>
</tr>
<tr>
<td>Hardware</td>
<td>50% of crypto chips</td>
</tr>
</tbody>
</table>
These entities are single points of privacy failure.

One breach...
One bug...
One backdoor...
These entities are single points of privacy failure.

One breach...

One bug...

One backdoor...
These entities are single points of privacy failure.

One breach...

One bug...

One backdoor...
These entities are single points of privacy failure.

One breach...
One bug...
One backdoor...
Solution? Don’t trust anyone.
Solution?  Don’t trust anyone.
Solution? Don’t trust anyone.
Solution? Don’t trust anyone.
Classic idea: Eliminate single points of failure by splitting trust

Challenge: Split trust to protect privacy, without sacrificing functionality
Combine ideas from **systems, security, crypto** to make it practical to split trust.
Eliminate single points of privacy failure

Applications

- **Data-collection systems** (NSDI ’17)
  Shipping in the Firefox browser

Libraries

- **Messaging services** (SOSP ’17),
  (IEEE S&P ’15), (USENIX Sec. ’13), (OSDI ’12), (CCS ’10)
  S&P distinguished paper, PET award

- **Cryptographic standards** (Eurocrypt ’18), (ECCC ’18)
  Best young researcher paper

OS

- **Password storage** (Asiacrypt ’16)

Hardware

- **Hardware components** (IEEE S&P ’19), (CCS ’13)
## Eliminate single points of privacy failure

<table>
<thead>
<tr>
<th>Applications</th>
<th>Data-collection systems (NSDI ’17)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Shipping in the Firefox browser</td>
</tr>
<tr>
<td>Libraries</td>
<td>Messaging services (SOSP ’17),</td>
</tr>
<tr>
<td></td>
<td>(IEEE S&amp;P ’15) (USENIX Sec. ’13) (OSDI ’12) (CCS ’10)</td>
</tr>
<tr>
<td></td>
<td>S&amp;P distinguished paper, PET award</td>
</tr>
<tr>
<td>OS</td>
<td>Cryptographic standards (Eurocrypt ’18) (ECCC ’18)</td>
</tr>
<tr>
<td></td>
<td>Best young researcher paper</td>
</tr>
<tr>
<td>Hardware</td>
<td>Password storage (Asiacrypt ’16)</td>
</tr>
<tr>
<td>Hardware components</td>
<td>(IEEE S&amp;P ’19) (CCS ’13)</td>
</tr>
</tbody>
</table>
Eliminate single points of privacy failure

- **Applications**
  - **Data-collection systems** (NSDI ’17)  
    - Shipping in the Firefox browser
  - **Messaging services** (SOSP ’17), (IEEE S&P ’15), (USENIX Sec. ’13), (OSDI ’12), (CCS ’10)  
    - S&P distinguished paper, PET award
  - **Cryptographic standards** (Eurocrypt ’18), (ECCC ’18)  
    - Best young researcher paper
  - **Password storage** (Asiacrypt ’16)
  - **Hardware components** (IEEE S&P ’19), (CCS ’13)

- **Libraries**
- **OS**
- **Hardware**
Running example:
Measuring effectiveness of content blocking
Running example:
Measuring effectiveness of content blocking
Running example:
Measuring effectiveness of content blocking

Mozilla wants to know:
“How many users disable content blocking on each of the top 500 websites?”
Manufacturers often answer these questions by collecting your sensitive data.

→ Single point of failure.
Manufacturers often answer these questions by collecting your sensitive data.

→ Single point of failure.
Mozilla’s previous attempt
Mozilla’s previous attempt

“I’ll have to file a complaint with the relevant Landes- and Bundesbeauftragten für Datenschutz”
Mozilla’s previous attempt

“I’ll have to file a complaint with the relevant Landes- and Bundesbeauftragten für Datenschutz”

“I wish you the worst of luck in your new venture to infringe even further upon the privacy of your users.”
Mozilla’s previous attempt

“I’ll have to file a complaint with the relevant Landes- and Bundesbeauftragten für Datenschutz”

“I wish you the worst of luck in your new venture to infringe even further upon the privacy of your users.”

“Do the Mozilla thing, not the Google thing.”
**Prio: Aggregate data without the privacy risks**

- **Built system:** C-G and Boneh (NSDI 2017)
- **Follow-up theoretical work:** Boneh, Boyle, C-G, Gilboa, and Ishai (preprint, 2019)

- Collect aggregate usage data **without seeing any single user’s data.**

- New cryptography makes this system practical
  - Proofs on secret-shared data

- Our Prio code ships to 200m+ Firefox users
  - In pilot phase: Enabled by default in Nightly
  - Largest deployment of technology based on PCPs (probabilistically checkable proofs)
Running example:
Measuring effectiveness of content blocking

• User \(i\) has a bit \(x_i^{\text{site.com}} \in \{0,1\}\)
  – Bit is “1” iff user disabled content blocking on site.com
  – These \(x_i\)s are sensitive – reveal user’s browsing history

• For each site in Top 500, Mozilla wants the sum of users’ bits:

\[
\text{How often content blocking breaks site.com} = \sum_{\text{users } i} x_i^{\text{site.com}}
\]

• Let’s focus on one site... \(x_i^{\text{site.com}}\)
  – Each user \(i\) has a single bit \(x_i\). Mozilla wants \(\sum_i x_i\).
Prio: System goals

1. Correctness. If clients and servers are honest, servers learn $f(\cdot)$
   Extension: Maintain correctness in spite of server faults

2. $f$-Privacy. Attacker must compromise all servers to learn more than $f(\cdot)$
   Extension: Differential privacy [DMNS06]

3. Disruption resistance. The worst that a malicious client can do is lie about her input.

4. Efficiency. Handle millions of submissions per server per hour
Prio: System goals

1. **Correctness.** If clients and servers are honest, servers learn $f(\cdot)$
   - Extension: Maintain correctness in spite of server faults

2. **$f$-Privacy.** Attacker must compromise all servers to learn more than $f(\cdot)$
   - Extension: Differential privacy [DMNS06]

3. **Disruption resistance.** The worst that a malicious client can do is lie about her input.

4. **Efficiency.** Handle millions of submissions per server per hour
Prio: System goals

1. Correctness. If clients and servers are honest, servers learn $f(\cdot)$
   Extension: Maintain correctness in spite of server faults

2. $f$-Privacy. Attacker must compromise all servers to learn more than $f(\cdot)$
   Extension: Differential privacy [DMNS06]

3. Disruption resistance. The worst that a malicious client can do is lie about her input.

4. Efficiency. Handle millions of submissions per server per hour
**Prio: System goals**

1. **Correctness.** If clients and servers are honest, servers learn $f(\cdot)$
   
   Extension: Maintain correctness in spite of server faults

2. **$f$-Privacy.** Attacker must compromise all servers to learn more than $f(\cdot)$
   
   Extension: Differential privacy [DMNS06]

3. **Disruption resistance.** The worst that a malicious client can do is lie about her input.

4. **Efficiency.** Handle millions of submissions per server per hour
Prio: System goals

1. **Correctness.** If clients and servers are honest, servers learn $f(\cdot)$
   Extension: Maintain correctness in spite of server faults

2. **$f$-Privacy.** Attacker must compromise all servers to learn more than $f(\cdot)$
   Extension: Differential privacy [DMNS06]

3. **Disruption resistance.** The worst that a malicious client can do is lie about her input.

4. **Efficiency.** Handle millions of submissions per server per hour
Prio: System goals

1. **Correctness.** If clients and servers are honest, servers learn $f(\cdot)$
   Extension: Maintain correctness in spite of server faults

2. **$f$-Privacy.** Attacker must compromise all servers to learn more than $f(\cdot)$
   Extension: Differential privacy [DMNS06]

3. **Disruption resistance.**
   The worst that a malicious client can do is lie about her input.

4. **Efficiency.** Handle millions of submissions per server per hour
Prio: System goals

1. **Correctness.** If clients and servers are honest, servers learn $f(\cdot)$
   
   Extension: Maintain correctness in spite of server faults

2. **$f$-Privacy.** Attacker must compromise all servers to learn more than $f(\cdot)$
   
   Extension: Differential privacy [DMNS06]

3. **Disruption resistance.**
   The worst that a malicious client can do is lie about her input.

4. **Efficiency.** Handle millions of submissions per server per hour
Prio: System goals

1. **Correctness.** If clients and servers are honest, servers learn $f(\cdot)$
   Extension: Maintain correctness in spite of server faults

2. **$f$-Privacy.** Attacker must compromise all servers to learn more than $f(\cdot)$
   Extension: Differential privacy [DMNS06]

3. **Disruption resistance.** The worst that a malicious client can do is lie about her input.

4. **Efficiency.** Handle millions of submissions per server per hour
1. **Correctness.** If clients and servers are honest, servers learn $f(\cdot)$
   Extension: Maintain correctness in spite of server faults

2. **$f$-Privacy.** Attacker must compromise all servers to learn more than $f(\cdot)$
   Extension: Differential privacy [DMNS06]

3. **Disruption resistance.** The worst that a malicious client can do is lie about her input.

4. **Efficiency.** Handle millions of submissions per server per hour

Focus on sums, for now
Relax correctness
Randomized response: [W65], [DMNS06], [DJW13], [BS15]
RAPPOR (2014, 2016), Wang et al. (2017),
Ding et al. (2017)...

Relax privacy model
Tor: PrivStats (2011), ANONIZE (2014), ...
SGX: Prochlo (2017), SGX-BigMatrix (2017), ...
Honest but curious: PDDP (2012), SplitX (2013), ...

Relax disruption resistance
Private metering (2011), PrivEx-S2 (2014),
PrivCount (2016), Federated ML (2016, 2017), ...

Relax efficiency
P4P (2010), Grid aggregation (2011), Haze (2013),
PrivEx-D2 (2014), Succinct sketches (2016), HisTorε (2017), ...
General MPC [GMW87], [BGW88]: FairPlay (2004), FairplayMP (2008), SEPIA (2010), Private matrix factorization (2013),
Private ridge regression (2018), ...
Straw-man scheme
Private sums without disruption resistance

[C88], [BGW88], ...
[KDK11] [DFKZ13] [PrivEx14] ...

\[x_1 = 1\]
Straw-man scheme
Private sums without disruption resistance
[C88], [BGW88], ...
[KDK11] [DFKZ13] [PrivEx14] ...

\[ x_1 = 1 \]

Pick three random “shares” that sum to \( x_1 = 1 \).

\[ 1 = 15 + (-12) + (-2) \pmod{p} \]

Send one share to each server.
Server A

Pick three random “shares” that sum to $x_1 = 1$.

$1 = 15 + (-12) + (-2) \pmod{p}$

Send one share to each server.
Straw-man scheme
Private sums without disruption resistance

\[ [C88], [BGW88], \ldots \]
\[ [KDK11] [DFKZ13] [PrivEx14] \ldots \]

Pick three random “shares” that sum to \( x_1 = 1 \).

\[
1 = 15 + (-12) + (-2) \quad \text{(mod } p)\]

Send one share to each server.
Straw-man scheme
Private sums without disruption resistance

Pick three random “shares” that sum to $x_1 = 1$.

$1 = 15 + (-12) + (-2)$ (mod $p$)

Send one share to each server.
Straw-man scheme
Private sums without disruption resistance

$x_1 = 1$

Pick three random “shares” that sum to $x_1 = 1$.

$1 = 15 + (-12) + (-2) \quad (\text{mod } p)$

Send one share to each server.
Straw-man scheme
Private sums without disruption resistance

\[ x_2 = 0 \]
Straw-man scheme
Private sums without disruption resistance

\[ x_2 = 0 \]
Straw-man scheme
Private sums without disruption resistance

\[ x_2 = 0 = (-10) + 7 + 3 \]
Straw-man scheme
Private sums without disruption resistance

Server A
15

Server B
−12

Server C
−2

\[ x_2 = 0 \quad \begin{array}{c} -10 \\ 7 \\ 3 \end{array} \]
Straw-man scheme
Private sums without disruption resistance

\( x_2 = 0 \)
Straw-man scheme
Private sums without disruption resistance

Server A  
15 – 10

Server B  
−12 + 7

Server C  
−2 + 3
Straw-man scheme
Private sums without disruption resistance

Server A
15 − 10

Server B
−12 + 7

Server C
−2 + 3
Straw–man scheme
Private sums without disruption resistance
Straw-man scheme
Private sums without disruption resistance

$$\begin{align*}
(15 - 10 + \cdots) + (-12 + 7 + \cdots) + (-2 + 3 + \cdots) &= x_1 + x_2 + x_3 + \cdots
\end{align*}$$

Servers learn the sum of the clients’ values and nothing else.
Straw-man scheme
Private sums without disruption resistance

Servers learn the sum of the clients' values and nothing else.

\[
(15 - 10 + \cdots) + (-12 + 7 + \cdots) + (-2 + 3 + \cdots) = x_1 + x_2 + x_3 + \cdots
\]

e.g., 15 phones disabled content blocking on mysite.com, but not which phones
Why this works: Shares are additive.

Say servers have shares of integers $x_1$ and $x_2$:

\[
x_1 = [x_1]_A + [x_1]_B + [x_1]_C
\]
\[
x_2 = [x_2]_A + [x_2]_B + [x_2]_C
\]

By adding, each server can compute a share of $x_1 + x_2$:

\[
[x_1 + x_2]_A = [x_1]_A + [x_2]_A
\]

Also, for a constant $C$, can compute a share of $C \cdot x$:

\[
[C \cdot x]_A = C \cdot [x]_A
\]
Why this works: Shares are additive.

Say servers have shares of integers $x_1$ and $x_2$:

\[
x_1 = [x_1]_A + [x_1]_B + [x_1]_C
\]
\[
x_2 = [x_2]_A + [x_2]_B + [x_2]_C
\]

By adding, each server can compute a share of $x_1 + x_2$:

\[
[x_1 + x_2]_A = [x_1]_A + [x_2]_A
\]

Also, for a constant $C$, can compute a share of $C \cdot x$:

\[
[C \cdot x]_A = C \cdot [x]_A
\]
Private sums: Straw-man scheme

Correctness. Servers learn the sum of the $x_i$'s

$f$-Privacy. Attacker must compromise all servers to learn more than sum of $x_i$'s

Efficiency. No heavy cryptographic operations

Disruption resistance. One malicious client can corrupt the output.
**Straw-man scheme**

One malicious client can corrupt output

\[
x_2 = 53
\]

Should be a value in the set \{0,1\}

Evil ad network
**Straw-man scheme**

One malicious client can corrupt output

\[
x_2 = 53 = 19 + 16 + 18
\]

Should be a value in the set \{0,1\}

Evil ad network
Straw-man scheme
One malicious client can corrupt output

\[ x_2 = 53 \]

Should be a value in the set \( \{0, 1\} \)

Evil ad network
Straw-man scheme
One malicious client can corrupt output

Evil ad network
Powerful but costly tools...

Multiparty computation

[GMW87], [BGW88]
Powerful but costly tools...

- Multiparty computation
  [GMW87], [BGW88]
- Traditional zero-knowledge proofs
  [GMR89]
Powerful but costly tools...

Multiparty computation
[GMW87], [BGW88]

Traditional zero-knowledge proofs
[GMR89]

New tool: Proof on secret-shared data
# Techniques for providing disruption resistance

Testing that a length-$n$ vector (e.g., data for $n$ sites) consists of secret-shared 0/1 integers.

<table>
<thead>
<tr>
<th></th>
<th>Public-key ops.</th>
<th>Communication</th>
<th>Slow-down</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Client</td>
<td>Server</td>
<td>C-to-S</td>
</tr>
<tr>
<td>Dishonest-maj. MPC</td>
<td>0</td>
<td>$\tilde{\Theta}(n)$</td>
<td>0</td>
</tr>
<tr>
<td>General succinct zero knowledge</td>
<td>$\tilde{\Theta}(n)$</td>
<td>$\tilde{O}(1)$</td>
<td>$\tilde{O}(1)$</td>
</tr>
<tr>
<td>General zero knowledge</td>
<td>$\tilde{\Theta}(n)$</td>
<td>$\tilde{\Theta}(n)$</td>
<td>$\tilde{\Theta}(n)$</td>
</tr>
</tbody>
</table>

(Table hides log factors.)
Techniques for providing disruption resistance
Testing that a length-\(n\) vector (e.g., data for \(n\) sites) consists of secret-shared 0/1 integers.

<table>
<thead>
<tr>
<th></th>
<th>Public-key ops.</th>
<th>Communication</th>
<th>Slow-down</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Client</td>
<td>Server</td>
<td>C-to-S</td>
</tr>
<tr>
<td>Dishonest-maj. MPC</td>
<td>0</td>
<td>(\tilde{\Theta}(n))</td>
<td>0</td>
</tr>
<tr>
<td>General succinct zero knowledge</td>
<td>(\tilde{\Theta}(n))</td>
<td>(\tilde{O}(1))</td>
<td>(\tilde{O}(1))</td>
</tr>
<tr>
<td>General zero knowledge</td>
<td>(\tilde{\Theta}(n))</td>
<td>(\tilde{\Theta}(n))</td>
<td>(\tilde{\Theta}(n))</td>
</tr>
<tr>
<td>Prio (latest version)</td>
<td>0</td>
<td>0</td>
<td>(\tilde{O}(1))</td>
</tr>
</tbody>
</table>

(Table hides log factors.)
Contribution: Proofs on secret-shared data

- Client sends proof to servers that \textit{Valid}(x) holds
  - For our example, \textit{Valid}(x) = “x ∈ \{0,1\}^n”
  - Servers exchange $O(1)$ bytes to check proof (e.g., 64 bytes)
- Prevents disruption in Prio
  - Servers detect and reject invalid client submissions
Contribution: Proofs on secret-shared data

Client (prover) \[ \pi \in \mathbb{F}^m \]

Servers (verifiers) \[ \pi \in \mathbb{F}^n \]

\( \vec{x} \in \mathbb{F}^n \)

vector of ints mod \( p \)
(e.g., data for \( n \) sites)

Complete. Honest servers accept valid \( \vec{x}s \)

Sound. Honest servers reject invalid \( \vec{x}s \) \( \Pr[\text{error}] \approx 2^{-128} \)

Privacy preserving. Any proper subset of servers learns nothing about \( \vec{x} \), apart from the fact that \( \vec{x} \) is valid
Diversion: The Box Game

You must decide: VALID or INVALID.

After asking three “linear questions” of the box.

Can you win?
Diversion:
The Box Game

You must decide:
VALID or INVALID.

After asking three “linear questions” of the box.

Can you win?
Diversion: The Box Game

You must decide: VALID or INVALID.

After asking three "linear questions" of the box.

Can you win?

\[ x = 1 \mid 0 \mid 1 \mid \cdots \mid 1 \]

\[ n \text{ integers } \mod \text{ prime } p \]

\[ 5 \mid 1 \mid 2 \mid \cdots \mid 9 \]

\[ 34 = 5 + 0 + 2 + \cdots \]
Diversion:
The Box Game

What if I can give you some help?

We show: It’s possible!
Diversion:
The Box Game

What if I can give you some help?

We show: It’s possible!
Diversion: The Box Game

What if I can give you some help?

We show: It’s possible!
We show: For any efficient predicate $\text{Valid}(\cdot)$...

...can check that $\text{Valid}(x)$ holds.
(Without learning anything else about $x$.)
Diversion: The Box Game

And what is in the box?
Our work: A new type of proof.

**Fully linear probabilistically checkable proof (PCP)**

Linear PCPs: [IKO07], [BCIOP13]
PCPs: [AS92], [ALMSS92]
Can construct fully linear PCPs from plain linear PCPs
Linear PCPs: [IK07], [GGPR13], [SBBVBPW13], [BCIOP13]

Rough idea:

• Take a circuit $\mathcal{C}$ computing the predicate $\text{VALID}$.

• Evaluate $\mathcal{C}$ on input $x$.

• Proof is an error-correcting encoding of the internal wire values in $\mathcal{C}(x)$.

• Three linear questions are enough to check proof.
Can construct fully linear PCPs from plain linear PCPs

Linear PCPs: [IKO07], [GGPR13], [SBBVBPW13], [BCIOP13]

Rough idea:

• Take a circuit $\mathcal{C}$ computing the predicate $\text{VALID}$.

• Evaluate $\mathcal{C}$ on input $x$.

• Proof is an error-correcting encoding of the internal wire values in $\mathcal{C}(x)$.

• Three linear questions are enough to check proof.
Our construction: Proof on secret-shared data from the Box Game

Client

Valid input $x$.

Servers

$[x]_A$

$[x]_B$
Our construction: Proof on secret-shared data from the Box Game

Client

\[ [x]_A \]

\[ [x]_B \]

Valid input \( x \).

\[ \pi \]
Our construction: Proof on secret-shared data from the Box Game

\[ \pi = [\pi]_A + [\pi]_B \]

Valid input \( x \).
Our construction: Proof on secret-shared data from the Box Game

Valid input $x$. 

Client

Servers

$[\pi]_A$  $[\pi]_B$

$[x]_A$

$[x]_B$
Our construction: Proof on secret-shared data from the Box Game

Valid input $x$. 

Client

Servers
Our construction: Proof on secret-shared data from the Box Game

Servers use shared randomness to choose each question.

Valid input $x$.

$\text{question } q$

 diversas
Our construction: Proof on secret-shared data from the Box Game

1. Servers jointly compute the Box’s answers to the three linear questions.
Our construction: Proof on secret-shared data from the Box Game

1. Servers jointly compute the Box’s answers to the three linear questions.

\[
\begin{bmatrix}
5 \\
1 \\
2 \\
7 \\
4
\end{bmatrix}_A
\Rightarrow
\begin{bmatrix}
5x_1 + x_2 + \ldots
\end{bmatrix}_A
\]

\[
\begin{bmatrix}
x_A \\
\pi_A
\end{bmatrix}
\]

\[
\begin{bmatrix}
x_B \\
\pi_B
\end{bmatrix}
\Rightarrow
\begin{bmatrix}
5x_1 + x_2 + \ldots
\end{bmatrix}_B
\]

\[
\begin{bmatrix}
x_A \\
\pi_A
\end{bmatrix}
\]

\[
\begin{bmatrix}
x_B \\
\pi_B
\end{bmatrix}
\]
Our construction: Proof on secret-shared data from the Box Game

1. Servers jointly compute the Box’s answers to the three linear questions.

Answer to question:

\[ 5x_1 + x_2 + \ldots \]
Our construction: Proof on secret-shared data from the Box Game

1. Servers jointly compute the Box’s answers to the three linear questions.

2. Servers each run the Box Game verifier on the answers.
Our construction: Proof on secret-shared data from the Box Game

1. Servers jointly compute the Box’s answers to the three linear questions.

2. Servers each run the Box Game verifier on the answers.

\[
\begin{align*}
[x]_A \quad [\pi]_A \\
[x]_B \quad [\pi]_B
\end{align*}
\]

\[a_1, a_2, a_3\]
Prio: Full system

- **Correctness.** Servers learn the sum of the $x_i$s
- **$f$-Privacy.** Attacker must compromise all servers to learn more than the sum of $x_i$s
- **Efficiency.** No heavy cryptographic operations
- **Disruption resistance.** Malicious clients have bounded influence

Using proofs on secret-shared data
Five-server cluster in five Amazon data centers.

Graph showing throughput (submissions/sec.) against submission length (values/submission). The graph compares Baseline (no privacy) with General zero knowledge. The Baseline shows a decreasing trend with increasing submission length, while General zero knowledge maintains a higher throughput with a slight increase at higher submission lengths.
Five-server cluster in five Amazon data centers.
Five-server cluster in five Amazon data centers.
Five-server cluster in five Amazon data centers.
Firefox deployment

Uses a C library I wrote (libprio) that implements Prio
- github.com/mozilla/libprio - 3.5k LoC

Challenges
- Our code runs in the main event loop
- Interacting across three teams
- Browser idiosyncrasies

Pilot phase, 11/2018–now
- Ships to all Firefox users, enabled by default in the “Nightly” build
- Mozilla runs all servers, collects only non-sensitive information

Next step: Move second server to external org.
Firefox deployment

$pk_A, pk_B$

$sk_A$

$sk_B$
Firefox deployment

36n + 160 bytes to collect n ints

~160 bytes
(AES key encrypted for server B)
<table>
<thead>
<tr>
<th>Preference Name</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>network.http.on_click_priority</td>
<td>boolean</td>
<td>true</td>
</tr>
<tr>
<td>network.http.rcwn.cache_queue_priority</td>
<td>integer</td>
<td>2</td>
</tr>
<tr>
<td>network.http.rendering-critical-requested</td>
<td>boolean</td>
<td>true</td>
</tr>
<tr>
<td>prio.publicKeyA</td>
<td>string</td>
<td>5D97ADED9B6EC40E9DD12...</td>
</tr>
<tr>
<td>prio.publicKeyB</td>
<td>string</td>
<td>1A3EE6A31E921A01B1F137...</td>
</tr>
<tr>
<td>privacy.trackingprotection.lower_numeric_policy</td>
<td>boolean</td>
<td>false</td>
</tr>
</tbody>
</table>
PrioEncoder.encode("demo-packet",
   {
      "booleans": [1, 0, 0, 1, 1, 0, 0, 1, 0, 1]
   })

Object { a: Uint8Array(461), b: Uint8Array(161) }
Testing Privacy-Preserving Telemetry with Prio

By Robert Helmer, Anthony Miyaguchi, Eric Rescorla

Posted on October 29, 2018 in Firefox and Privacy

Building a browser is hard; building a good browser inevitably requires gathering a lot of data to make sure that things that work in the lab work in the field. But as soon as you gather data, you have to make sure you protect user privacy. We’re always looking at ways to improve the security of our data collection, and lately we’ve been experimenting with a really cool technique called Prio.
Prio supports a range of aggregation functions

- **Average**
- **Variance** [PBBL11]
- **Most popular (approx.)** [MDD16]
- **Min and max (approx.)**
- **Quality of arbitrary regression model** \( (R^2) \)
- **Least-squares regression**
- **Gradient descent step** [BIKMMPRSS17]
Prio supports a range of aggregation functions

- Average
- Variance [PBBL11]
- **Most popular** (approx.) [MDD16]
- Min and max (approx.)
- Quality of arbitrary regression model ($R^2$)
- Least-squares regression
- Gradient descent step [BIKMMPRSS17]

Encode integer $x_i$ as $(x_i^2, x_i)$.

$$\text{Var}(X) = (\sum_i x_i^2) - (\sum_i x_i)^2$$
Prio applies more broadly...

Which PINs are most popular?

How congested is the Bay Bridge?

How much time does a user spend in each app? [DKY17]
Eliminate single points of privacy failure

**Applications**

Data-collection systems (NSDI ’17)
Shipping in the Firefox browser

**Libraries**

Messaging services (SOSP ’17),
(IEEE S&P ’15) (USENIX Sec. ’13) (OSDI ’12) (CCS ’10)
S&P distinguished paper, PET award

Cryptographic standards
(Eurocrypt ’18) (ECCC ’18)
Best young researcher paper

**OS**

Password storage (Asiacrypt ’16)

**Hardware**

Hardware components (IEEE S&P ’19) (CCS ’13)
Eliminate single points of privacy failure

- **Applications**
  - **Data-collection systems** (NSDI ’17)
    Shipping in the Firefox browser
  - **Messaging services** (SOSP ’17),
    (IEEE S&P ’15) (USENIX Sec. ’13) (OSDI ’12) (CCS ’10)
    S&P distinguished paper, PET award
  - **Cryptographic standards**
    (Eurocrypt ’18) (ECCC ’18)
    Best young researcher paper
  - **Password storage** (Asiacrypt ’16)

- **Libraries**

- **OS**

- **Hardware**
  - **Hardware components** (IEEE S&P ’19) (CCS ’13)
U2F hardware authentication tokens

- USB token that computes digital signatures
- To authenticate, provide password and signature
  - Protects against browser malware
U2F hardware authentication tokens

- USB token that computes digital signatures
- To authenticate, provide password and signature
  - Protects against browser malware
U2F hardware authentication tokens

- USB token that computes digital signatures
- To authenticate, provide password and signature
  - Protects against browser malware
U2F hardware authentication tokens

- USB token that computes digital signatures
- To authenticate, provide password and signature
  - Protects against browser malware
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.

Security Keys are inexpensive USB-based devices that offer an alternative approach to two-factor authentication (2FA), which requires the user to log in to a Web site using something they know (the password) and something they have (e.g., a mobile device).

A Google spokesperson said Security Keys now form the basis of all account access at Google.
Opaque crypto hardware is a threat

[S84], [D88], [YY97], [ESJRZ14], [BPR14], [SWSHE15], [YHDAS16], ...

• Faulty token can use weak keys or weak randomness
  ⇒ Inadvertently leak your secret keys to evil.com

• Manufacturer is a single point of failure...
Opaque crypto hardware is a threat

[S84], [D88], [YY97], [ESJRZ14], [BPR14], [SWSHE15], [YHDAS16], ...

• Faulty token can use weak keys or weak randomness
  ⇒ Inadvertently leak your secret keys to evil.com

• Manufacturer is a single point of failure...

Token

Flaw Found in an Online Encryption Method

SAN FRANCISCO — A team of European and American mathematicians and cryptographers have discovered an unexpected weakness in the encryption system widely used worldwide for online shopping, banking, e-mail and other Internet services intended to remain private and secure.

[HDWH12], [LHABKW12]
Opaque crypto hardware is a threat

[S84], [D88], [YY97], [ESJRZ14], [BPR14], [SWSHE15], [YHDAS16], ...

- Faulty token can use weak keys or weak randomness ⇒ Inadvertently leak your secret keys to evil.com
- Manufacturer is a single point of failure...

Flaw Found in an Online Encryption Method
By JOHN MARKOFF FEB. 14, 2012
SAN FRANCISCO — A team of European and American mathematicians and cryptographers have discovered an unexpected weakness in the encryption system widely used worldwide for online shopping, banking, e-mail and other Internet services intended to remain private and secure...

[HDWH12], [LHABKW12]

Protecting against randomness failures
C-G, Mu, Boneh, Ford (CCS 2013)
Opaque crypto hardware is a threat

[S84], [D88], [YY97], [ESJRZ14], [BPR14], [SWSHE15], [YHDAS16], ...

• Faulty token can use weak keys or weak randomness
⇒ Inadvertently leak your secret keys to evil.com

• Manufacturer is a single point of failure...

Token

**Flaw Found in an Identity Token**

By JOHN MARKOFF  FEB. 14, 2012

SAN FRANCISCO — A team of computer scientists and cryptographers have discovered a flaw in the RSA encryption system widely used for securing e-mail and other Internet services.

**Security Advisory 2017-10-16**

Weak RSA Key Generation

[NSSKM17]
Opaque crypto hardware is a threat

- Faulty token can use weak keys or weak randomness ⇒ Inadvertently leak your secret keys to evil.com
- Manufacturer is a single point of failure...

→ A chance to use cryptographic hardware the right way.
True2F: Protection against token faults


If token is correct:
    Malicious browser learns nothing about the token’s secret keys
⇒ Protects against browser compromise

If browser is correct:
    Malicious website (evil.com) cannot distinguish the real token from an ideal token
⇒ Protects against faulty token token
True2F: Protection against token faults

Idea: Split trust between browser and token.
Use new two-party protocols to implement each token operation.

[MS15, DMS16]
True2F: Design constraints

1. **Backwards-compatibility** with U2F-enabled sites
   
   **Idea:** Design new schemes for two-party key-generation and signing

2. **No changes** to U2F hardware (24 MHz chip, 512 KB flash)
   
   **Idea:** Token offloads compute and storage to browser, while checking browser’s work
Evaluation

On Google’s standard U2F hardware

Authentication time

Naïve implementation 446 ms
True2F (optimized) 57 ms
U2F 23 ms

Ongoing: Effort to standardize True2F via the U2F standards body (FIDO alliance)
Eliminate single points of privacy failure

- **Applications**
  - **Data-collection systems** (NSDI ’17)
    Shipping in the Firefox browser
  - **Messaging services** (SOSP ’17), (IEEE S&P ’15), (USENIX Sec. ’13), (OSDI ’12), (CCS ’10)
    S&P distinguished paper, PET award
  - **Cryptographic standards** (Eurocrypt ’18), (ECCC ’18)
    Best young researcher paper
  - **Password storage** (Asiacrypt ’16)
  - **Hardware components** (IEEE S&P ’19), (CCS ’13)

- **Libraries**
- **OS**
- **Hardware**
Future work

Protecting against crypto hardware bugs
In the data center (HSMs), in the phone, ...
Future work

Metadata-hiding messaging at Internet scale
Future work

Training sophisticated ML models on private data
Conclusion

To use a computer system today is to surrender control.
  – Of our most sensitive data
  – Of our hardware
  – Of our autonomy

Why do we accept this?
We have no other option.

Our systems should put the user back in charge.
It is feasible, e.g., by splitting trust.

We should expect more and get more.
Conclusion

To use a computer system today is to surrender control.
  – Of our most sensitive data
  – Of our hardware
  – Of our autonomy

Why do we accept this?
  We have no other option.

Our systems should put the user back in charge.
  It is feasible, e.g., by splitting trust.

We should expect more and get more.