Versatile Anonymous Authentication with Cloak

Simon Beyzerov and Eli Yablons
Mentor: Sacha Servan-Schreiber
Motivation

- Online authentication is ubiquitous
  - Typically makes no considerations for user privacy

- Metadata is powerful
  - Contextualizes surface-level data
  - Can be used to draw powerful inferences when cross-referenced with more concrete information

- Metadata is often leaked in typical authentication mechanisms
Metadata

[secret conversation]
Motivation

- Online authentication is ubiquitous
  - Typically makes no considerations for user privacy

- **Metadata** is powerful
  - Contextualizes surface-level data
  - Can be used to draw powerful inferences when cross-referenced with more concrete information

- Metadata is often leaked in typical authentication mechanisms
Example

Username: calvin
Password: ball

User: calvin
Password: ball
Time: 14:05:19 UTC

Authorized user! ✓
Anonymous Authentication

- Ability to **anonymize this exchange**
  - Prevent server from learning **which user** is authenticating
    - Only that **someone** is authenticating
  - Can other information about the user be hidden?

- Limit **metadata** leakage
  - Collected metadata can allow complex relationships to be drawn about users

- What about **Multi-Factor Authentication**?
  - Using another medium to verify your identity after the initial authentication step
  - Leaks data to third parties
Example

Username: calvin
Password: ball

User: calvin
Password: ball
Time: 14:05:19 UTC

Authorized user! ✔

Requested Content
Example

Authorized user!

Username: *******
Password: **********
Time: 14:05:19 UTC
Example

Username: *******
Password: ************
Time: 14:05:19 UTC
IPv6: Masked with VPN or Tor

Authorized user!

Requested Content
Existing Solutions

- Anonymous Credentials
- Multi-Party Computation
- Cryptographic Accumulators
Anonymous Credentials

- **Anonymous Credentials**
  - Requires storing and managing keys on the client
    - Credentials are like “tokens” that can be issued and spent, but must be stored
  - Do not have efficient revocation of credentials
  - Do not integrate well with current username-password systems
  - Unclear how to extend to more complex applications such as authenticated retrieval
Our Goal

Allow *Calvin* to authenticate to *MIT* without revealing who is logging in.
Design, Threat Model, Assumptions

- We consider a setting with two **non-colluding** authentication servers
  - For example: **MIT & Duo**:
    - **MIT** handles password authentication
    - **Duo** is normally responsible for **Two-Factor Authentication**
    - Independent parties, so non-collusion is a reasonable assumption

- In Cloak, both servers are responsible for password authentication and second-factor authentication, but remain independent and non-colluding
Design, Threat Model, Assumptions

- Assume both servers, individually, are fully malicious
  - MIT and Duo try and identify Calvin when he is authenticating.
  - Remain non-colluding, so they don’t maliciously interact with each other

- Users are assumed to be malicious by default
  - Malicious users want to authenticate, regardless of whether they have an account
Overview: Design, Threat Model, Assumptions
Overview: Design, Threat Model, Assumptions

1. Question mark
2. Key
3. Authenticated?
4. Database
5. Database
6. Duo
7. MIT
Technical overview

1. Use secret-sharing to obliviously “select” the account (username + password)
   ○ Neither server learns which account was selected
   ○ Achieved using Distributed Point Functions which are evaluated by the servers

2. Prove knowledge of the password without revealing any information
   ○ Performed using a new technique for proving knowledge over secret-shares
Background: Secret Sharing

- Distribute shares of a secret value among multiple parties
- Secret can only be revealed by combining shares
  - Nothing is learned without all parties coming together
- Toy example:
  - Masking a secret in a finite field: \((x - r)\) and \((r)\) form secret shares of \(x\)

\[
x - r + r = x
\]

- Notation: we use \([x]\) to denote a secret-share of \(x\)
Step 1: privately selecting the account

<table>
<thead>
<tr>
<th>Username</th>
<th>Password</th>
</tr>
</thead>
<tbody>
<tr>
<td>spaceman</td>
<td>******</td>
</tr>
<tr>
<td>hobbs</td>
<td>******</td>
</tr>
<tr>
<td>calvin</td>
<td>******</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Derkins</td>
<td>******</td>
</tr>
</tbody>
</table>

Query_A → A

Query_B → B

[calvin]_A[******]_A

[calvin]_B[******]_B
Step 1: privately selecting the account

<table>
<thead>
<tr>
<th>Username</th>
<th>Password</th>
</tr>
</thead>
<tbody>
<tr>
<td>spaceman</td>
<td>*****</td>
</tr>
<tr>
<td>hobbs</td>
<td>*****</td>
</tr>
<tr>
<td>calvin</td>
<td>*****</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Derkins</td>
<td>*****</td>
</tr>
</tbody>
</table>

Query_A: spaceman

Query_B: calvin

Secret Shares: [calvin]_A[******]_A

Internal: [calvin]_B[******]_B
Tool: Distributed Point Functions [NI’14]

One-hot vector:

\[ \nu = \begin{bmatrix} 0 & 0 & 1 & \ldots & 0 \end{bmatrix} \]

\[ \begin{bmatrix} 1 & 0 & 0 & \ldots & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 0 & 1 & \ldots & 1 \end{bmatrix} \]

\[ \begin{array}{c}
\begin{array}{c}
\text{A}
\end{array}
\end{array} \quad \begin{array}{c}
\begin{array}{c}
\text{B}
\end{array}
\end{array} \]

\[ \begin{array}{c}
\begin{array}{c}
+ \quad +
\end{array}
\end{array} \quad \begin{array}{c}
\begin{array}{c}
\text{A}
\end{array}
\end{array} + \text{B} \quad \begin{array}{c}
\begin{array}{c}
\text{0}
\end{array}
\end{array}
\]

\[ \begin{array}{c}
\begin{array}{c}
\text{0}
\end{array}
\end{array} \quad \begin{array}{c}
\begin{array}{c}
\text{0}
\end{array}
\end{array} \quad \begin{array}{c}
\begin{array}{c}
\text{1}
\end{array}
\end{array} \quad \begin{array}{c}
\begin{array}{c}
\ldots
\end{array}
\end{array} \quad \begin{array}{c}
\begin{array}{c}
\text{0}
\end{array}
\end{array} \]

21
Account selection with the DPF

<table>
<thead>
<tr>
<th>Username</th>
<th>Password</th>
</tr>
</thead>
<tbody>
<tr>
<td>spaceman</td>
<td>****</td>
</tr>
<tr>
<td>hobbs</td>
<td>****</td>
</tr>
<tr>
<td>calvin</td>
<td>****</td>
</tr>
<tr>
<td>:</td>
<td>:</td>
</tr>
<tr>
<td>Derkins</td>
<td>****</td>
</tr>
</tbody>
</table>

\[
\begin{bmatrix}
0 & 0 & 1 & \cdots & 0
\end{bmatrix}
\cdot
\begin{bmatrix}
\text{calvin}
\end{bmatrix}
= 
\begin{bmatrix}
****
\end{bmatrix}
\]
Account selection with the DPF

<table>
<thead>
<tr>
<th>Username</th>
<th>Password</th>
</tr>
</thead>
<tbody>
<tr>
<td>spaceman</td>
<td>$g^{x_1}$</td>
</tr>
<tr>
<td>hobbs</td>
<td>$g^{x_2}$</td>
</tr>
<tr>
<td>calvin</td>
<td>$g^{x_3}$</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Derkins</td>
<td>$g^{x_n}$</td>
</tr>
</tbody>
</table>

\[
\begin{bmatrix}
0 & 0 & 1 & \ldots & 0
\end{bmatrix}
\begin{bmatrix}
calvin \\
g^{x_3}
\end{bmatrix}
\]
Schnorr Proof [S’98]

Fix values, $G$, $g$, $g^x$ where $G$ is a group and $g$ is a generator of $G$.

**Goal:** efficiently prove to a verifier that you know $x$.

Must satisfy **zero-knowledge**: the verifier learns nothing beyond that the prover knows $x$. 

I’ll show you I know $x_3$ without you learning what $x_3$ is!
There are some issues!

A Schnorr Proof is not quite enough:

- We do **not** want servers to know **who** is verifying
  - A server that learns $g^{x_3}$ also learns that Calvin is the one authenticating.

- Servers in our design hold **shares** of $g^{x_3}$; hiding the user
  - Can we modify Schnorr’s proof to work over $[g^{x_3}]$ instead of $g^{x_3}$?
New tool: Schnorr Proof over Secret Shares (SPoSS)

Our contribution: SPoSS

Fix values, $\mathbb{G}$, $g$, $g^x$ where $\mathbb{G}$ is a prime order group and $g$ is a generator of $\mathbb{G}$.

Goal: efficiently prove to a verifier that you know $x$.

Must satisfy zero-knowledge: the verifier learns nothing beyond that the prover knows $x$.

We design a Schnorr proof for a secret-shared element $g^x$ with multiple verifiers:

- No verifier learns anything about $g^x$, but proof still passes if and only if the prover knows $x$.
- Each verifier has $[g^x]$ and must be convinced that the prover knows $x$. 
New tool: Schnorr Proof over Secret Shares (SPoSS)

We design a Schnorr proof for a secret-shared element $g^x$ with multiple verifiers:

- No verifier learns anything about $g^x$, but proof still passes if and only if the prover knows $x$.
- Each verifier has $[g^x]$ and must be convinced that the prover knows $x$.

I’ll show you I know $x_3$ without you learning what $x_3$ is or what $g^{x_3}$ is.
I’ll show you I know $x_3$ without you learning what $x_3$ is or what $g^{x_3}$ is

$I’ll$ show you I know my “password” without you learning what my “password” is or what my “username” is
The Cloak Protocol

1. **Prove:** use a DPF to obliviously select the account and make a SPoSS proof-of-knowledge for the corresponding password.

2. **Audit:** servers individually check the SPoSS proof over the secret-shares of the selected account to verify the password.

3. **Verify:** servers confirm with each other whether or not the user is authenticated.
Evaluation (work in progress)

- Implemented in **Go v1.14**
- **Massively parallelizable**: Auth with **1 billion** accounts takes **5 seconds** with 600 cores

- Evaluated on one core:

\[
2^{17} \approx 100,000 \text{ users} \Rightarrow 300 \text{ milliseconds}
\]

\[
2^{20} \approx 1,000,000 \text{ users} \Rightarrow 3 \text{ seconds}
\]

*Evaluated w/ 1 server @ 1-core*
Evaluation (work in progress)

- A standard 32-core server can support ~1 sec for **10 million users**
  and ~100 sec authentication with **1 billion users**
- Parallelization allows support for large-scale services

---

**Facebook**
- 2021 breach of **509,458,528** accounts

**MySpace**
- 2008 breach of **359,420,698** accounts

**Adobe**
- 2013 breach of **152,445,165** accounts

**DropBox**
- 2021 breach of **68,648,009** accounts

https://haveibeenpwned.com/
Acknowledgements

We would like to thank...

- Our mentor Sacha Servan-Schreiber,
- the MIT PRIMES program,
- our parents.

Questions?
References

