# Versatile Anonymous Authentication with Cloak

**Simon Beyzerov** and **Eli Yablon**
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

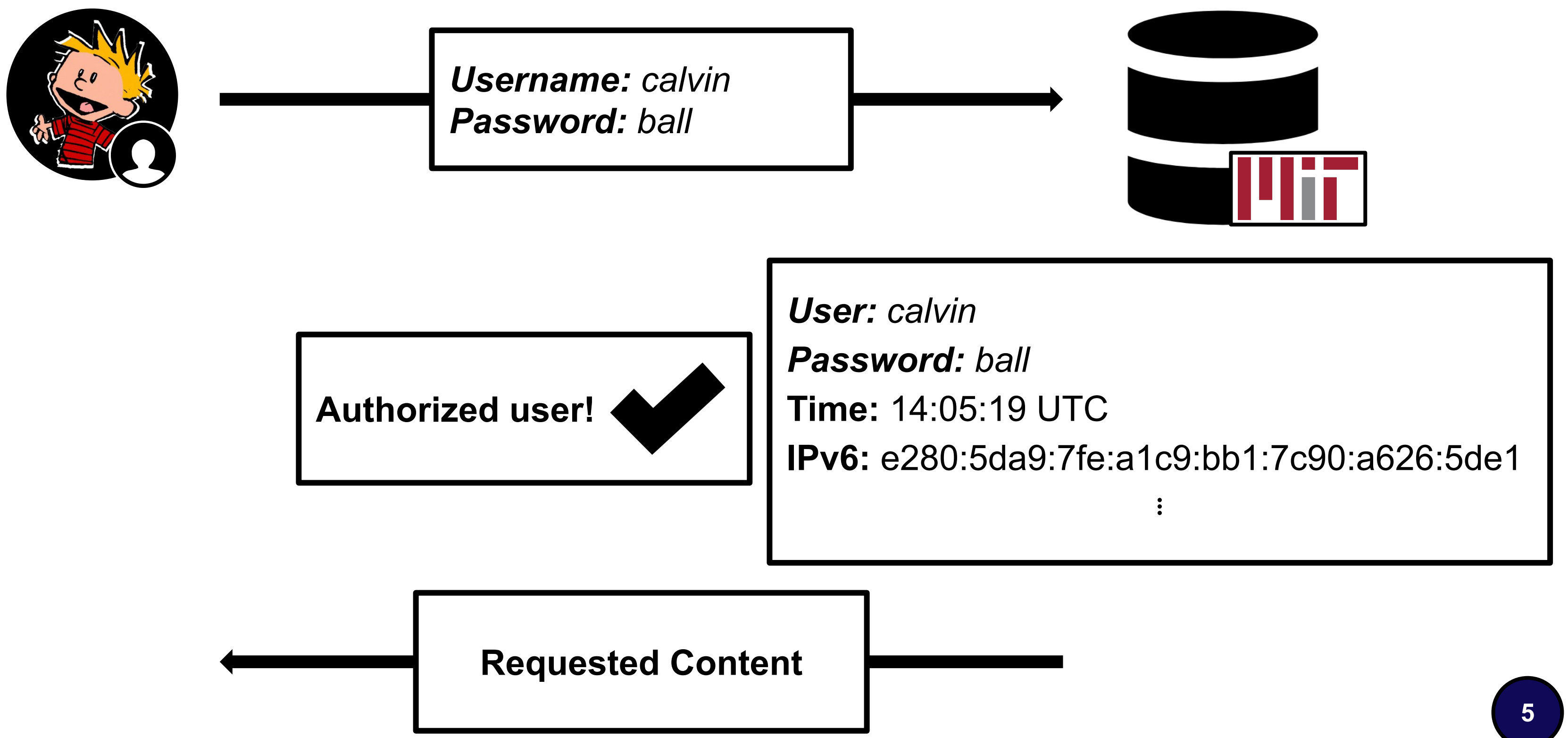
- Metadata is often leaked in typical authentication mechanisms

# Metadata

# 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
**IPv6:** e280:5da9:7fe:a1c9:bb1:7c90:a626:5de1
⋮
**Authorized user!**
**Requested Content**

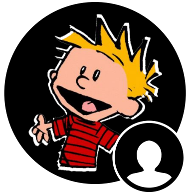
# Anonymous Authentication

- 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



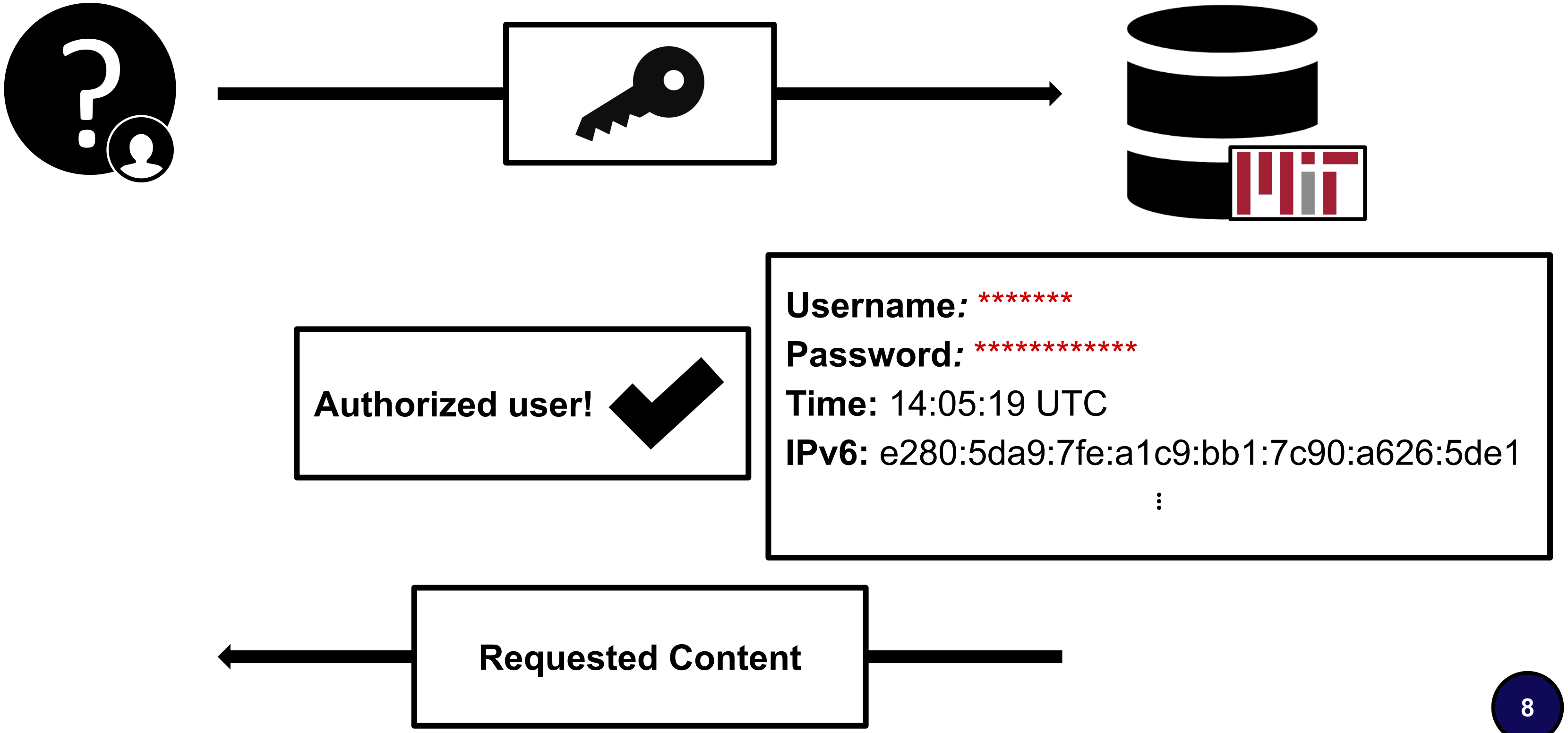
***Username:*** *calvin*
***Password:*** *ball*

***User:*** *calvin*
***Password:*** *ball*
**Time:** 14:05:19 UTC
**IPv6:** e280:5da9:7fe:a1c9:bb1:7c90:a626:5de1
⋮

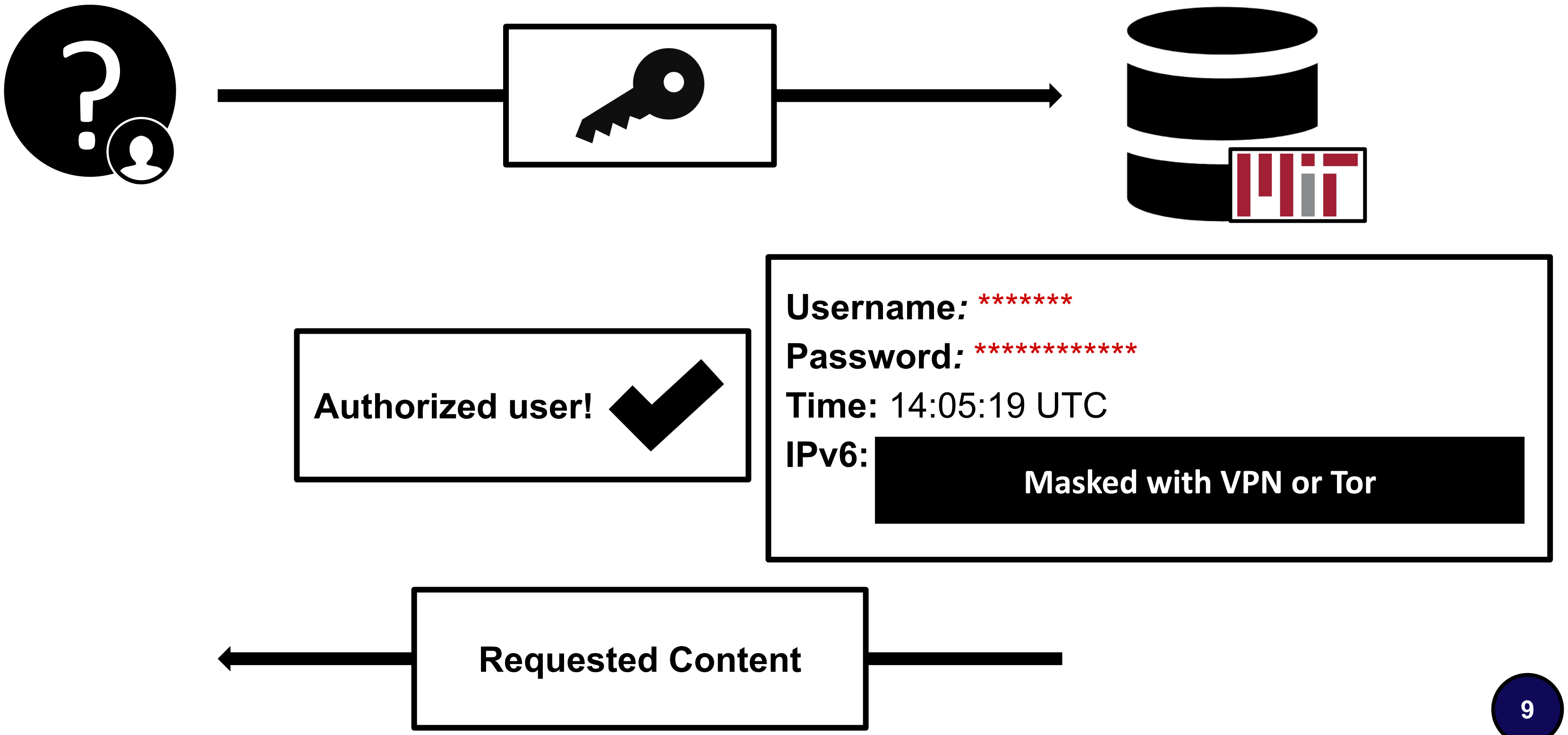
**Authorized user!** ✔

**Requested Content**

# Example


Authorized user!
Username: *******
Password: ************
Time: 14:05:19 UTC
IPv6: e280:5da9:7fe:a1c9:bb1:7c90:a626:5de1
Requested Content

# 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

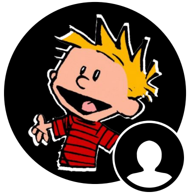
- In Cloak, both servers are responsible for password authentication and second-factor authentication, but remain independent and non-colluding

# Design, Threat Model, Assumptions

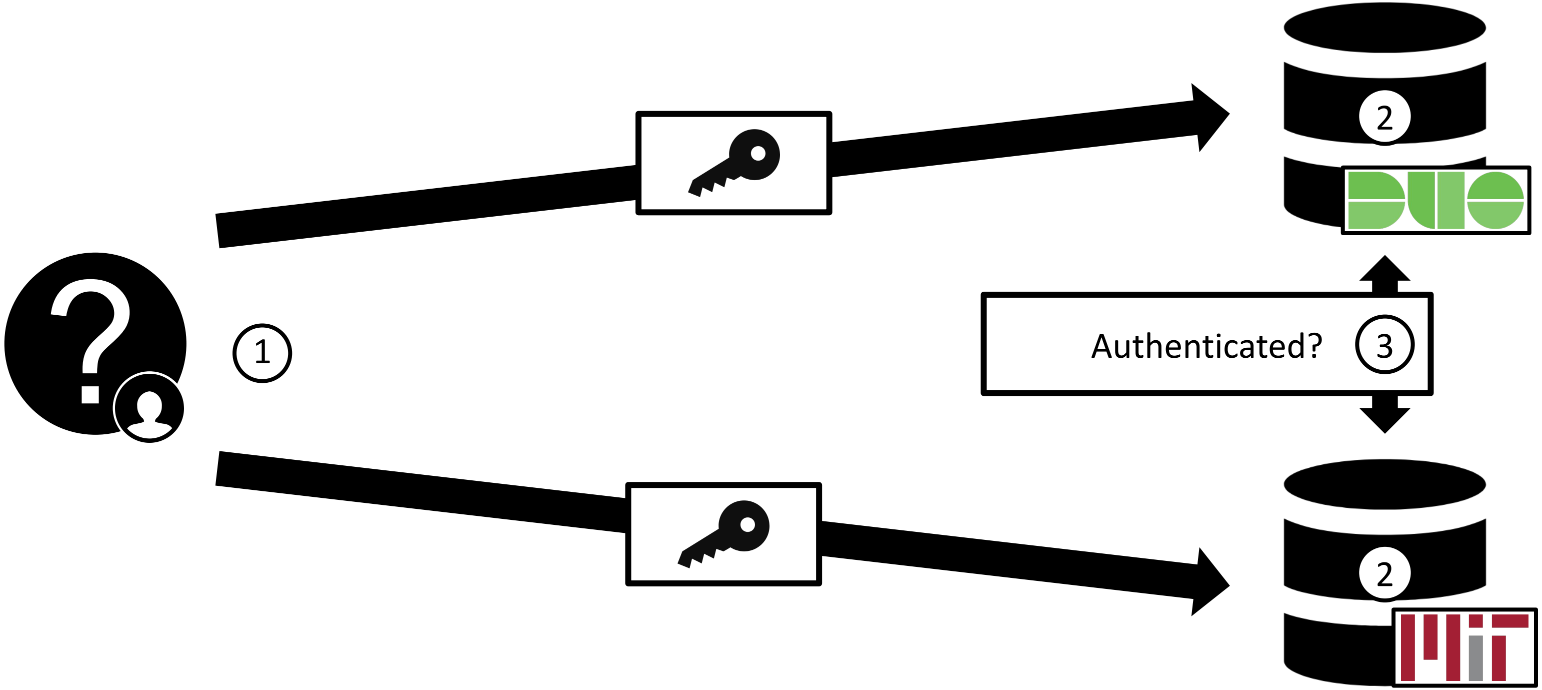
- Assume both servers, individually, are fully malicious
  - **MIT** and **Duo** try and identify Calvin when he is authenticating.
  - Remain non-colluding, so they don't maliciously interact with each other

- 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

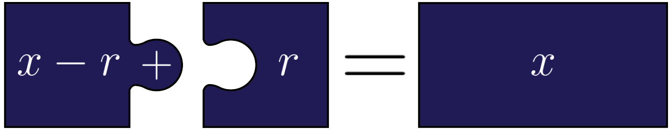


# Technical overview

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

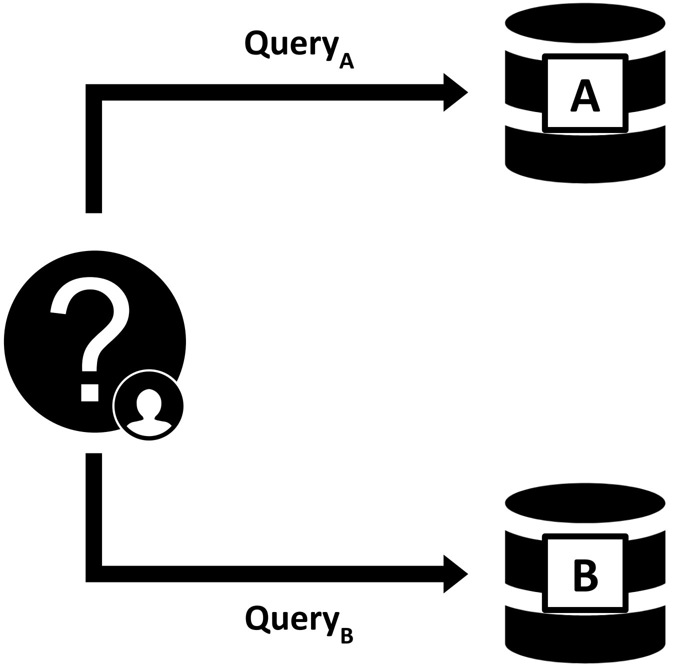
# Background: Secret Sharing

- 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




- Notation: we use [x] to denote a secret-share of x

# Step 1: privately selecting the account

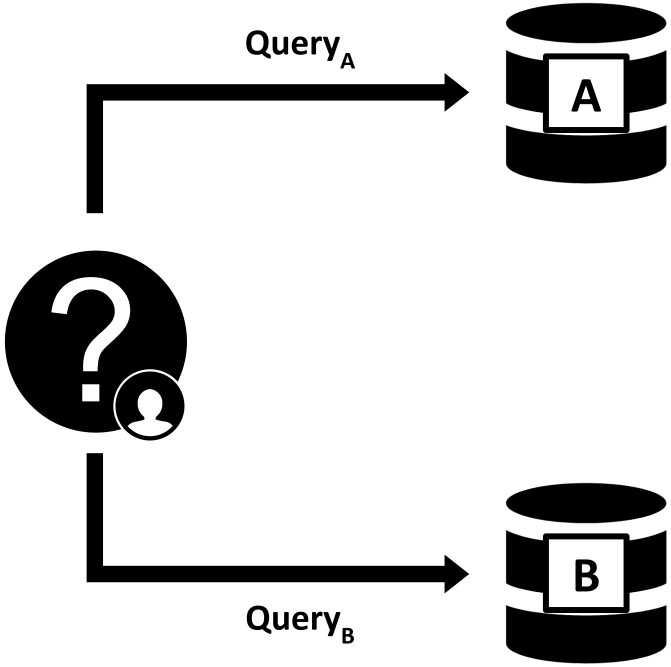



| Username | Password |
|---|---|
| spaceman | ****** |
| hobbs | ****** |
| calvin | ****** |
| ⋮ | ⋮ |
| Derkins | ****** |

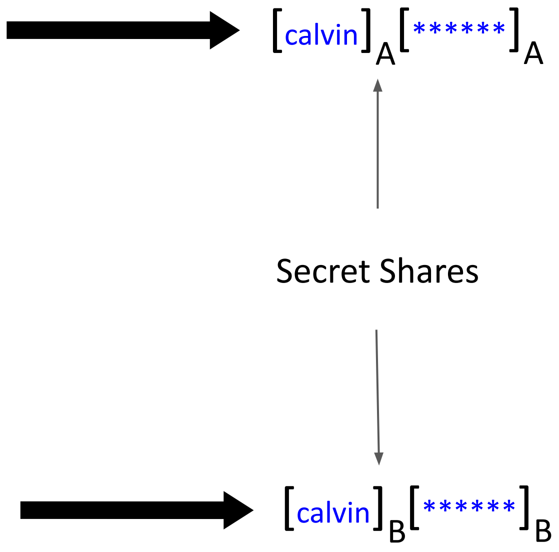
$\rightarrow$ $[\text{calvin}]_A[\text{******}]_A$

$\rightarrow$ $[\text{calvin}]_B[\text{******}]_B$

# Step 1: privately selecting the account




| Username | Password |
|---|---|
| spaceman | ****** |
| hobbs | ****** |
| calvin | ****** |
| ⋮ | ⋮ |
| Derkins | ****** |



# Tool: Distributed Point Functions [NI'14]

One-hot vector:

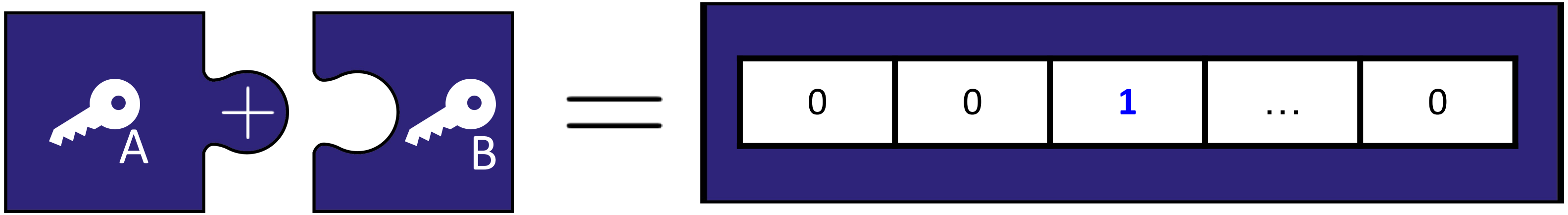
$$v = \begin{array}{|c|c|c|c|c|} \hline 0 & 0 & \color{blue}{1} & \dots & 0 \\ \hline \end{array}$$

| | | | | |
|---|---|---|---|---|
| **1** | **0** | **0** | **…** | **1** |

A

| | | | | |
|---|---|---|---|---|
| **1** | **0** | **1** | **…** | **1** |

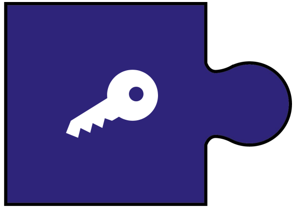
B



# Account selection with the DPF

| Username | Password |
|---|---|
| spaceman | **** |
| hobbs | **** |
| calvin | **** |
| ⋮ | ⋮ |
| Derkins | **** |

$$\bullet \begin{bmatrix} \boxed{0} & \boxed{0} & \boxed{1} & \boxed{\cdots} & \boxed{0} \end{bmatrix} = \begin{bmatrix} \text{calvin} \end{bmatrix} \begin{bmatrix} **** \end{bmatrix}$$



# Account selection with the DPF

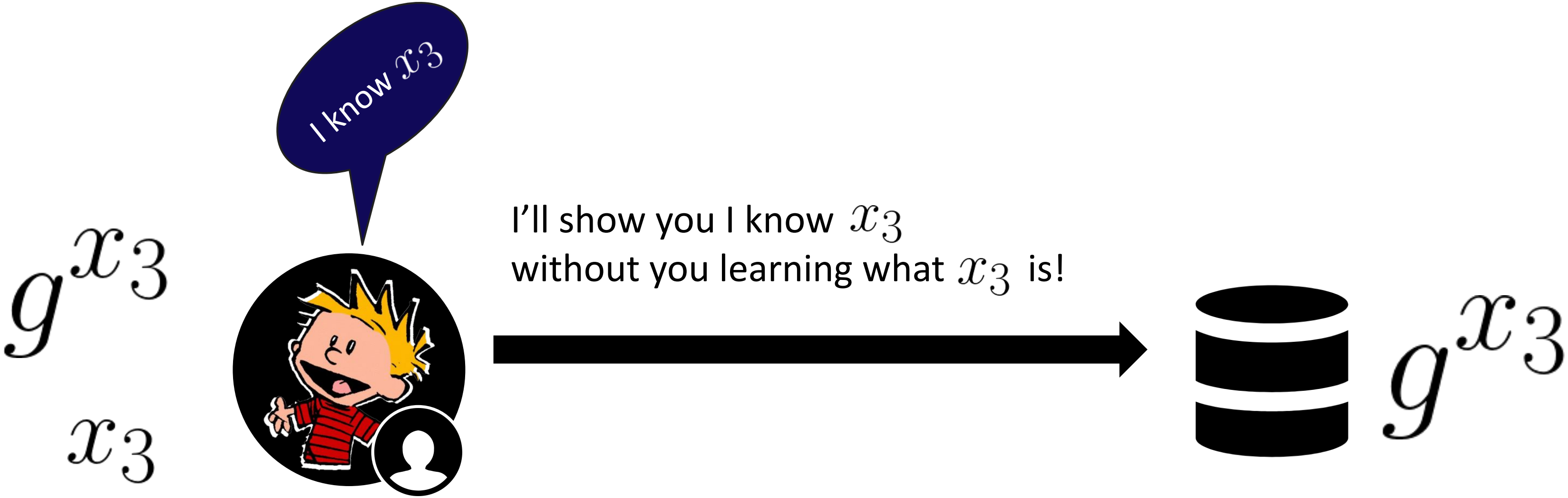
| Username | Password |
|---|---|
| spaceman | $g^{x_1}$ |
| hobbs | $g^{x_2}$ |
| calvin | $g^{x_3}$ |
| $\vdots$ | $\vdots$ |
| Derkins | $g^{x_n}$ |

$$\bullet \begin{bmatrix} 0 & 0 & 1 & \cdots & 0 \end{bmatrix} = \begin{bmatrix} \text{calvin} \end{bmatrix} \begin{bmatrix} g^{x_3} \end{bmatrix}$$

## **Schnorr Proof** [S'98]

Fix values, $\mathbb{G}$, $g$ , $g^x$ where $\mathbb{G}$ is a 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 $x$ .



# 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 $x$ .

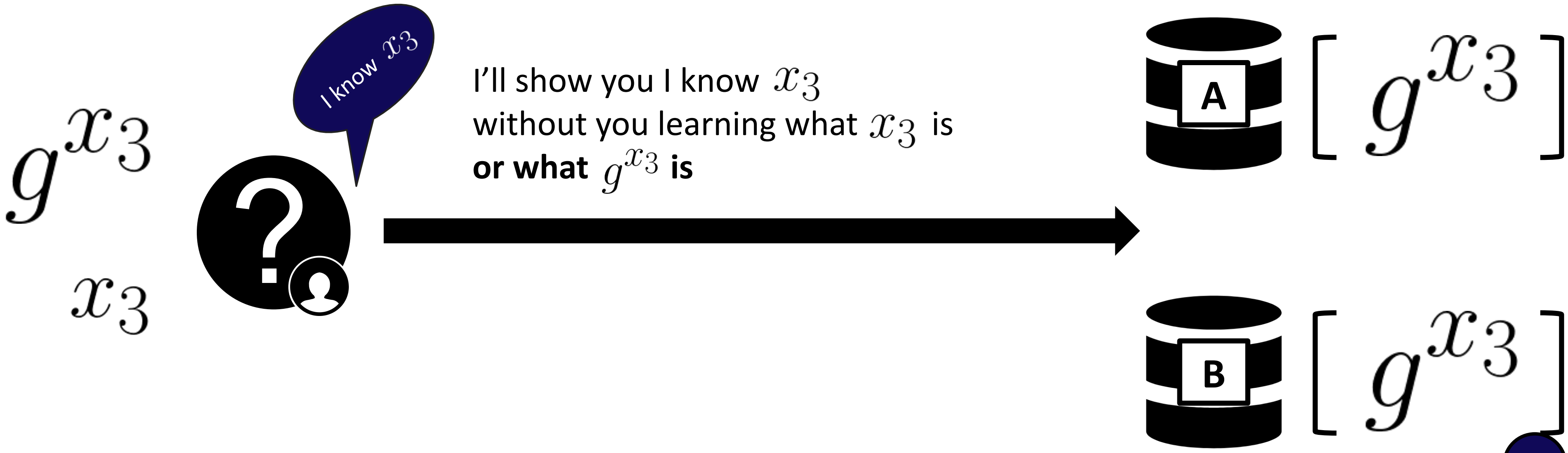
**We design a Schnorr proof for a secret-shared element $g^x$ with multiple verifiers:**

- **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**

$\approx$

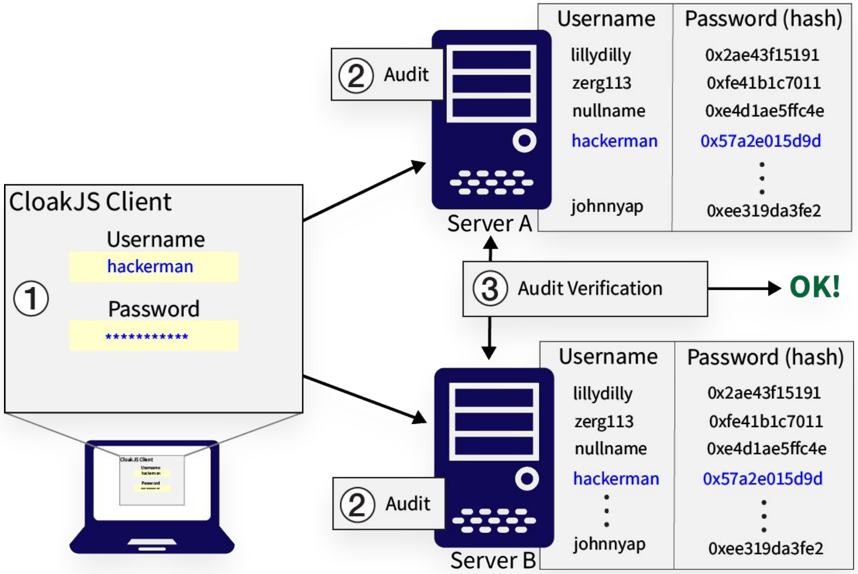
I’ll show you I know my “password”
without you learning what my
“password” is **or what** my “username” is

# The Cloak Protocol

① **Prove:** use a DPF to obliviously select the account and a 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.

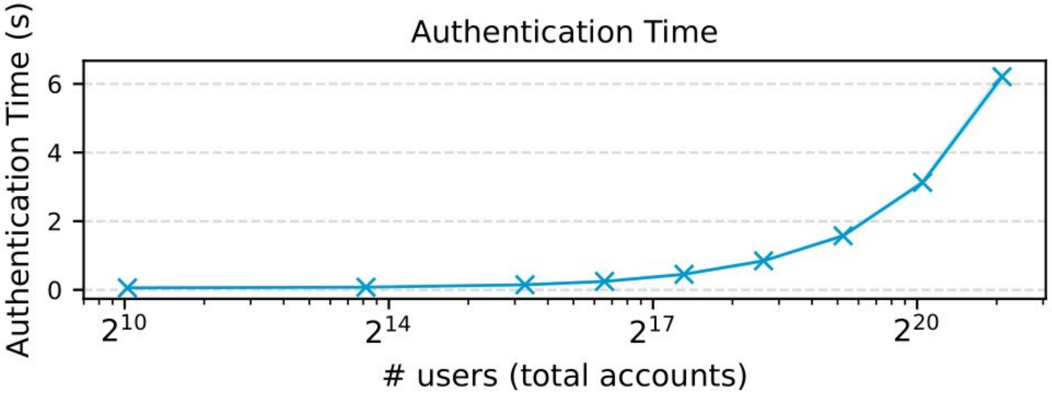
③ **Verify:** servers confirm with each other whether or not the user is authenticated.



# 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:

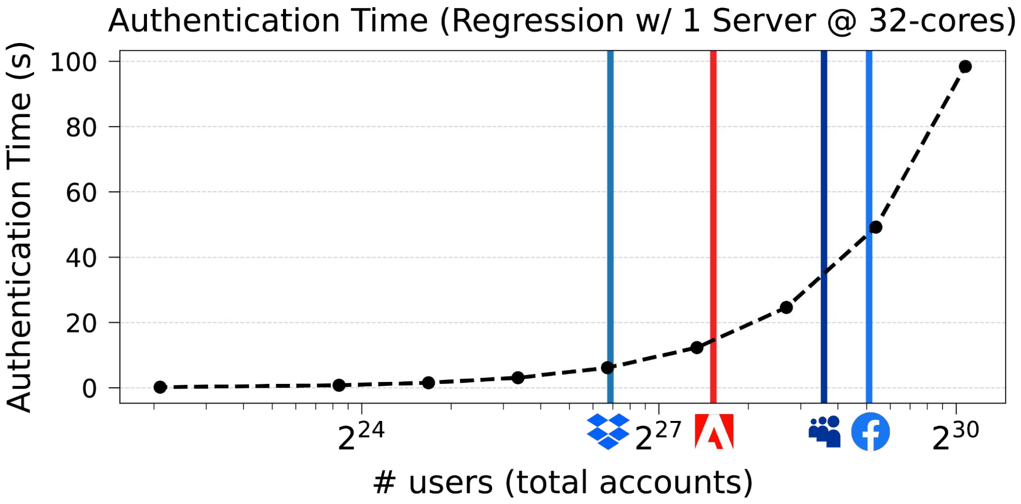



*Evaluated w/ 1 server @ 1-core*

$2^{17} \approx$ 100,000 users $\Rightarrow$ **300 milliseconds**

$2^{20} \approx$ 1,000,000 users $\Rightarrow$ **3 seconds**

# Evaluation (work in progress)




**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/

- 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

# Acknowledgements

We would like to thank...

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

**Questions?**

# References

[BGI’15]: Boyle, Elette, Niv Gilboa, and Yuval Ishai. "Function secret sharing." *Annual international conference on the theory and applications of cryptographic techniques*. Springer, Berlin, Heidelberg, 2015.

[S’98]: Schnorr, Claus-Peter. "Efficient identification and signatures for smart cards." *Conference on the Theory and Application of Cryptology*. Springer, New York, NY, 1989.

[NI’14]: Gilboa, Niv, and Yuval Ishai. "Distributed point functions and their applications." *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, Berlin, Heidelberg, 2014.