Protecting accounts from credential stuffing with password breach alerting

Kurt Thomas, Jennifer Pullman, Kevin Yeo, Ananth Raghunathan, Patrick Gage Kelley, Luca Invernizzi, Borbala Benko, Tadek Pietraszek, Sarvar Patel, Dan Boneh, Elie Bursztein

Google
Stanford University

A Little about Myself

- Lead team with goal of utilizing advanced cryptographic techniques to improve user privacy
- Research Interests: Oblivious RAM, Encrypted Search, Private Information Retrieval

Outline

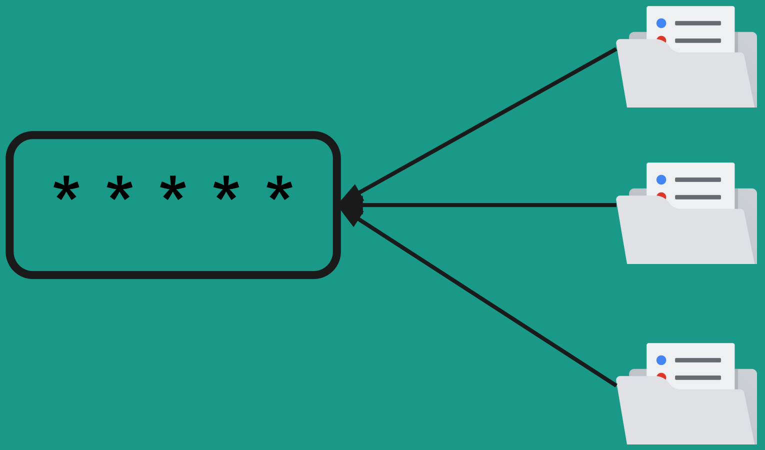
Problem

- Credential stuffing
- Password breach alerting

Solution

- Protocol for password breach alerting
- Privacy guarantees
- Deployment at Google

Credential Stuffing



Hackers Accumulate Breached Credentials


WIRED | Hackers Are Passing Around a Megaleak of 2.2 Billion

ANDY GREENBERG SECURITY 01.30.19 05:31 PM

HACKERS ARE PASSING AROUND A MEGALEAK OF 2.2 BILLION RECORDS


SECURITY

Old hacks strike again: Data from 2.2B accounts lands on the dark web

More than 600 gigabytes of hacked accounts from years ago have been compiled and are free to download.

BY ALFRED NG | JANUARY 31, 2019 9:05 AM PST

What do Breached Credentials look like?

Username	Salt	Hashed Password
UserA	E1F53135E559C253	H(PasswordA + E1F53135E559C253)
UserB	84B03D034B409D4E	H(PasswordB + 84B03D034B409D4E)
…	…	…

Reversing Hashes

Input.

Salt: E1F53135E559C253
Hashed password: 72AE25495A7981C40622D49

Goal.

Find password P such that H(P + E1F53135E559C253) = 72AE25495A7981C40622D49.

Reversing Hashes

Why is this possible?

- Utilize specialized hardware built to compute hashes efficiently
- Can try weaker passwords first
- Passwords have low entropy

Preventions

- Use “expensive-to-compute” hashes such as Scrypt or Argon2

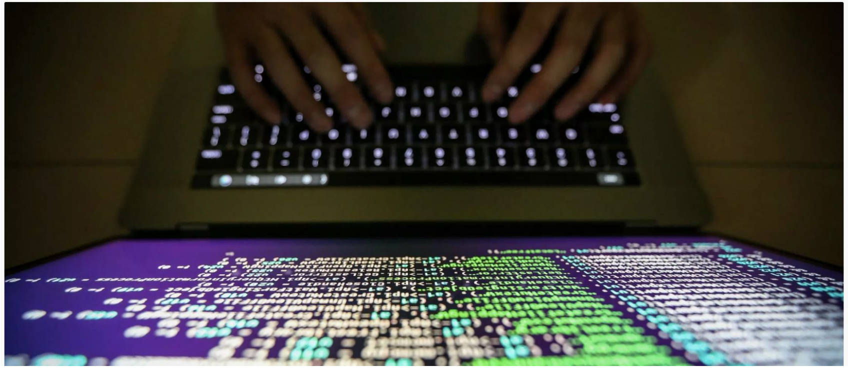
Hijacking Accounts using Breached Credentials

QUARTZ

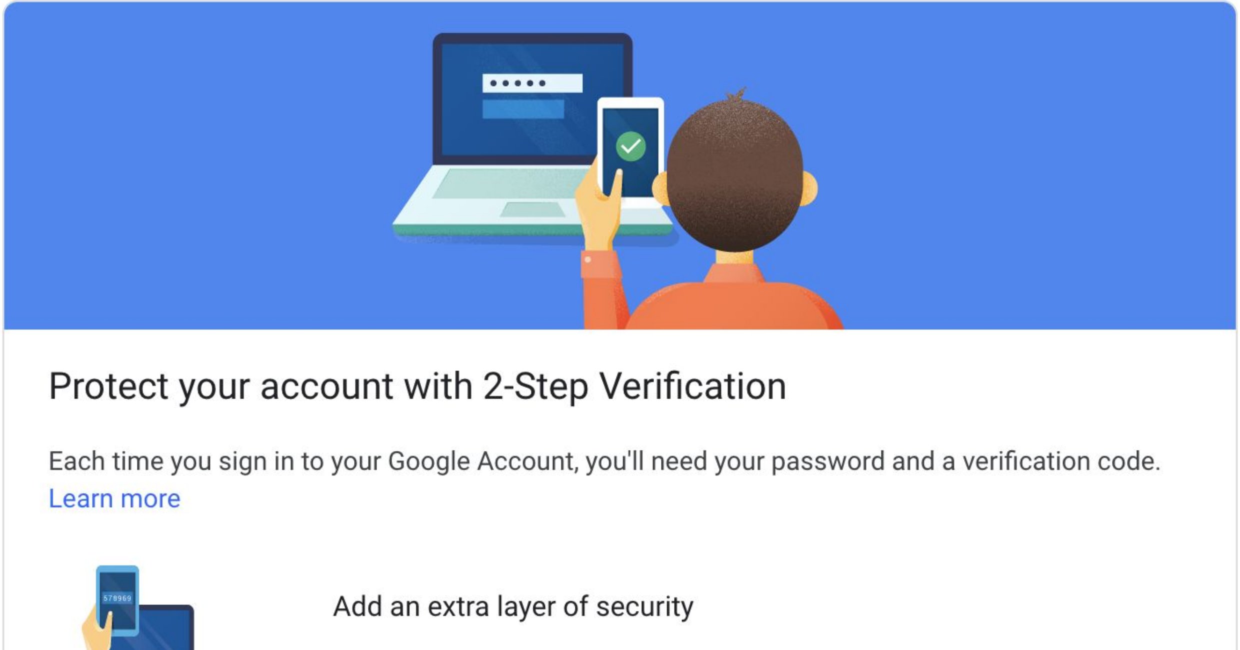
WHO MOVED MY CHEESE?

Hackers account for 90% of login attempts at online retailers

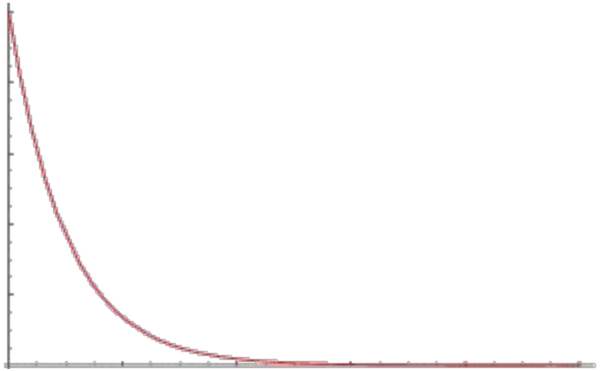
By John Detrixhe · July 18, 2018



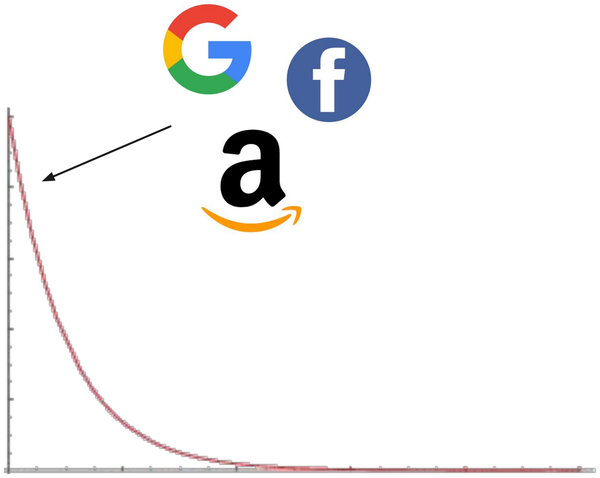
Defending against Breached Credentials



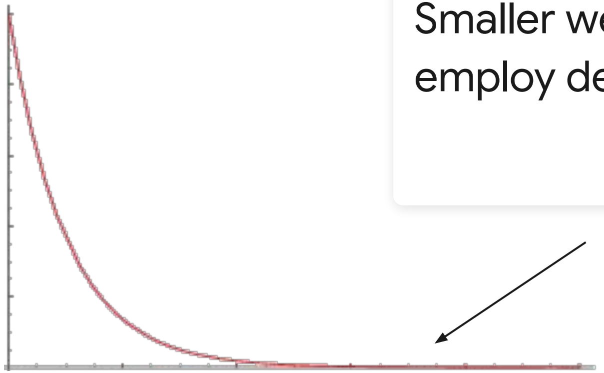
Small Tail of Websites



Small Tail of Websites

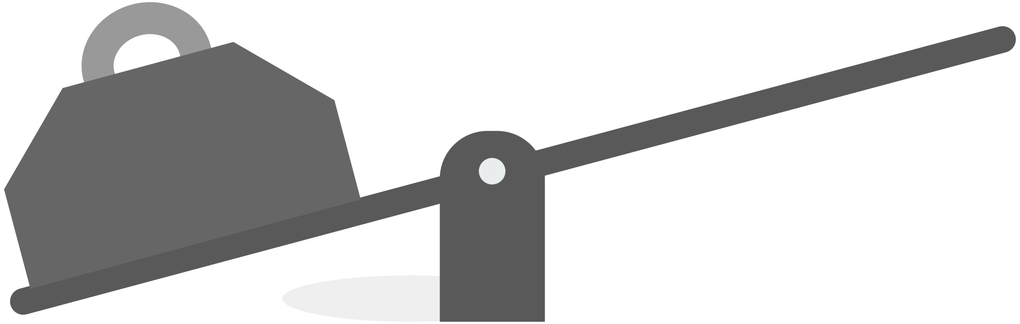


Small Tail of Websites



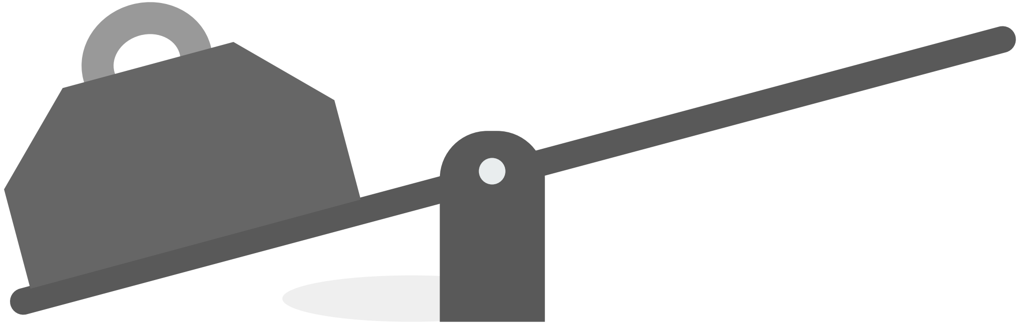
Smaller websites may not employ defence in-depth.

Imbalance of Knowledge



Imbalance of Knowledge

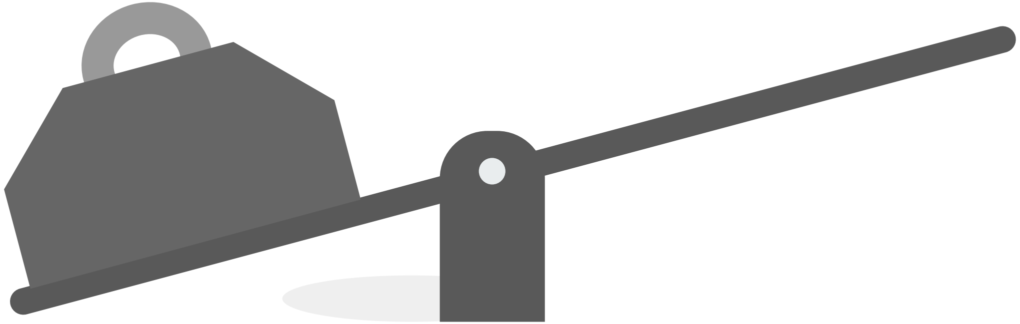
Attackers have wide-scale access to billions of stolen usernames and passwords.



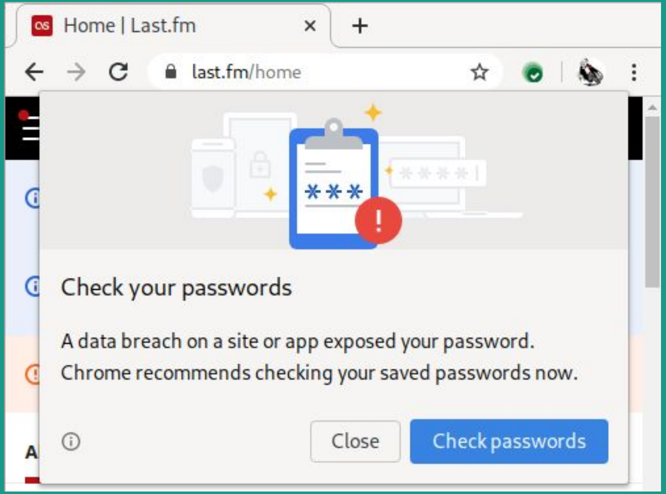
Imbalance of Knowledge

Attackers have wide-scale access to billions of stolen usernames and passwords.

Users and identity providers remain in the dark about which accounts to resecure.



Password Breach Alerting



Password Breach Alerting

Password Breach Alerting

$(\text{username}_1, \text{password}_1)$

$(\text{username}_2, \text{password}_2)$

...

$(\text{username}_n, \text{password}_n)$

Password Breach Alerting

$(username_1, password_1)$

$(username_2, password_2)$

...

$(username_n, password_n)$

Password Breach Alerting

(username, password)

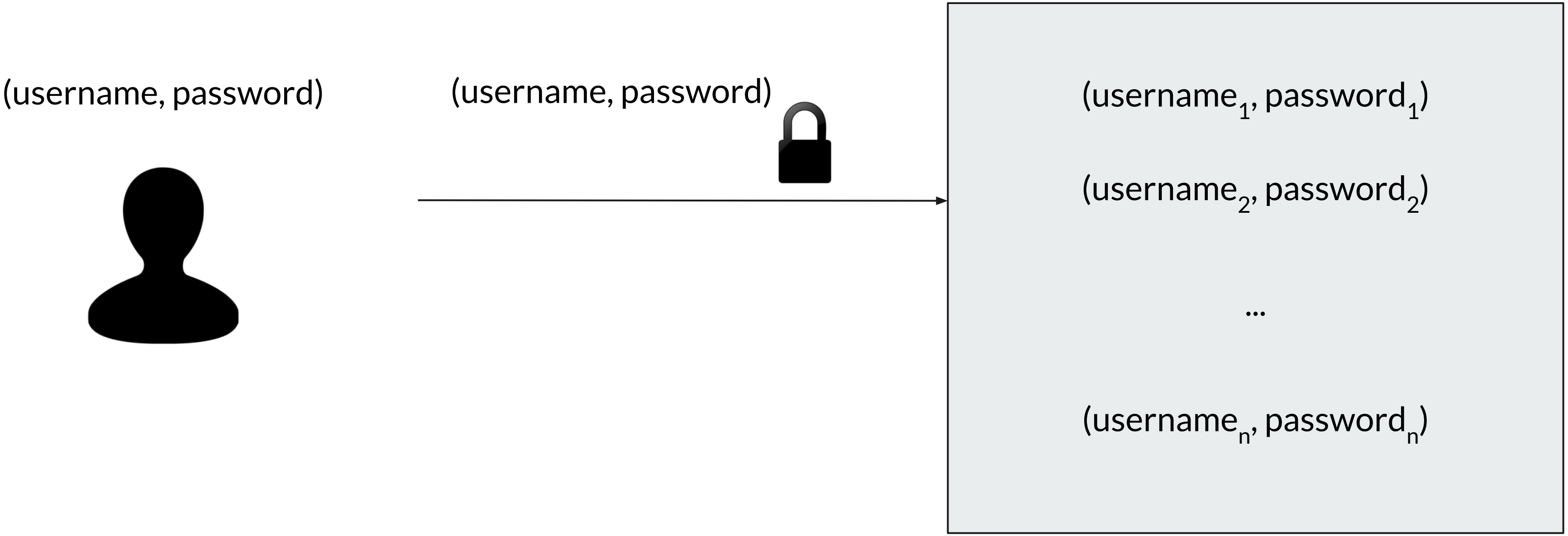
(username$_1$, password$_1$)

(username$_2$, password$_2$)

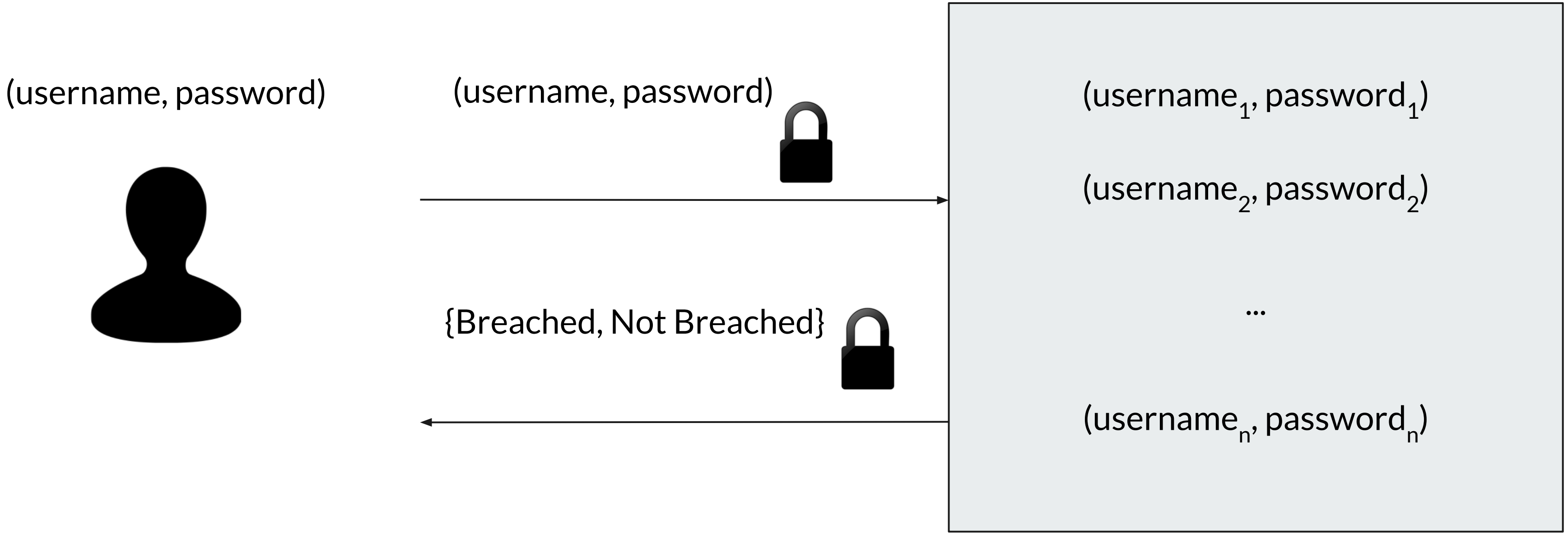
...

(username$_n$, password$_n$)

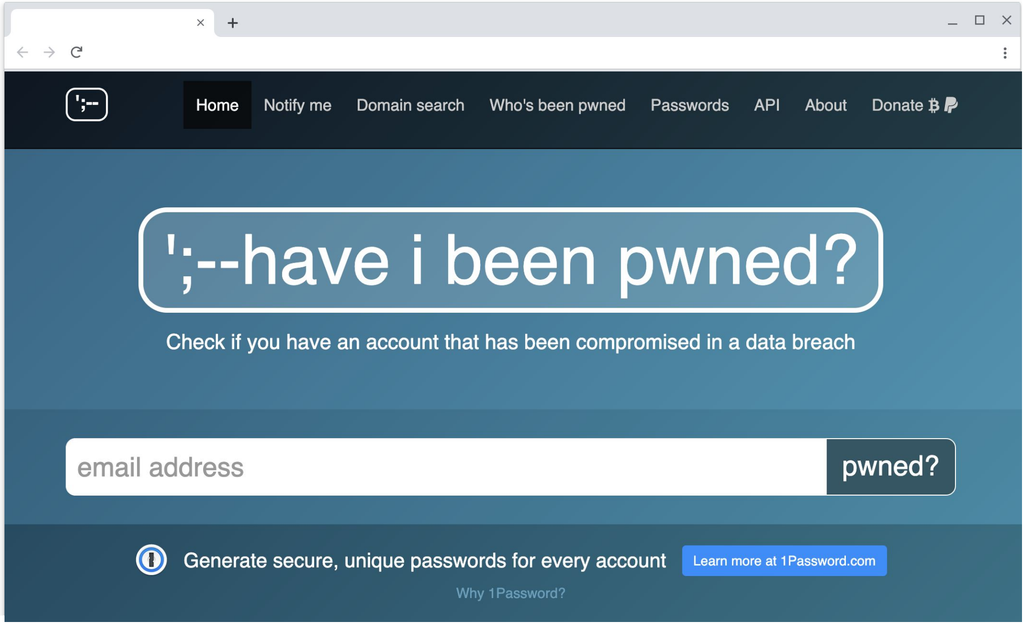
Password Breach Alerting



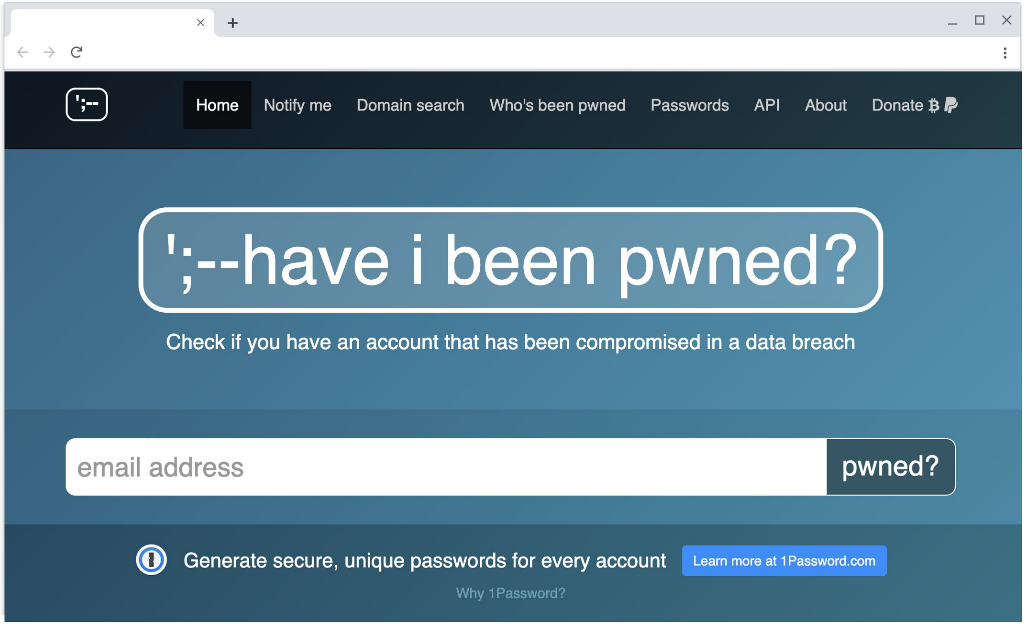
Password Breach Alerting



Prior Solutions

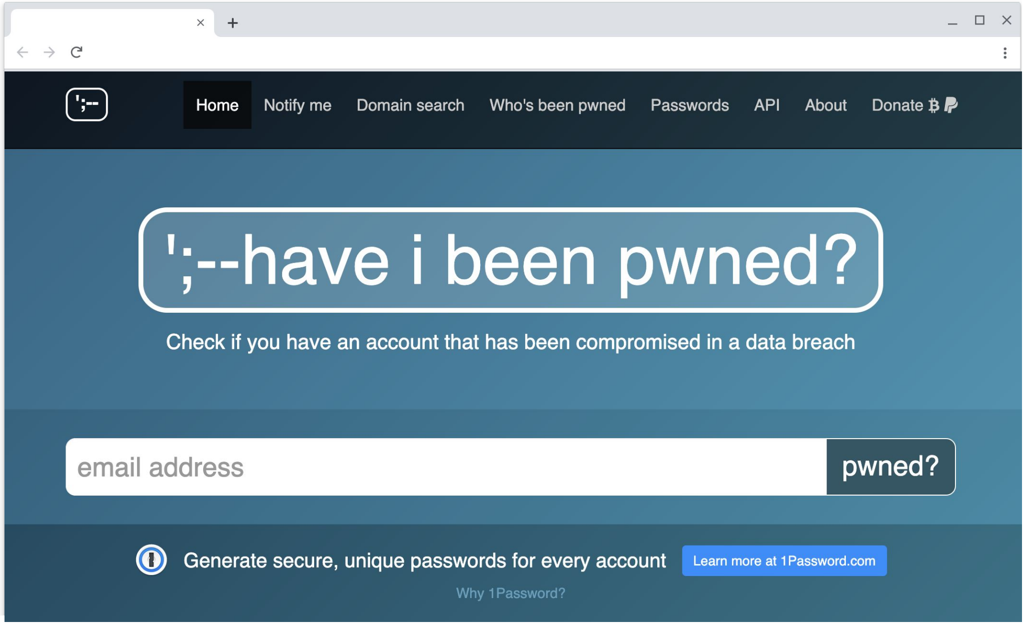


Prior Solutions



Risk of inaccurate advice due to username-only, or password-only lookup.

Prior Solutions



Risk of inaccurate advice due to username-only, or password-only lookup.

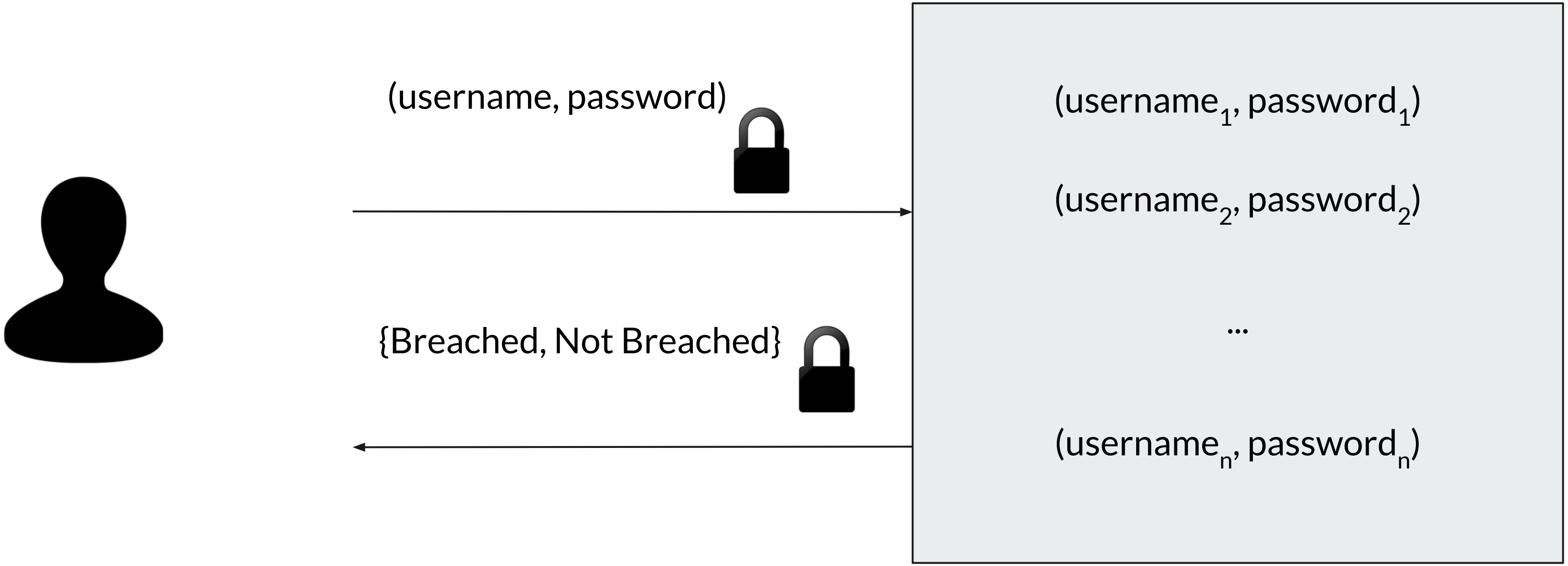
Privacy risk of what data you share with the breach alerting service.

Design Principles

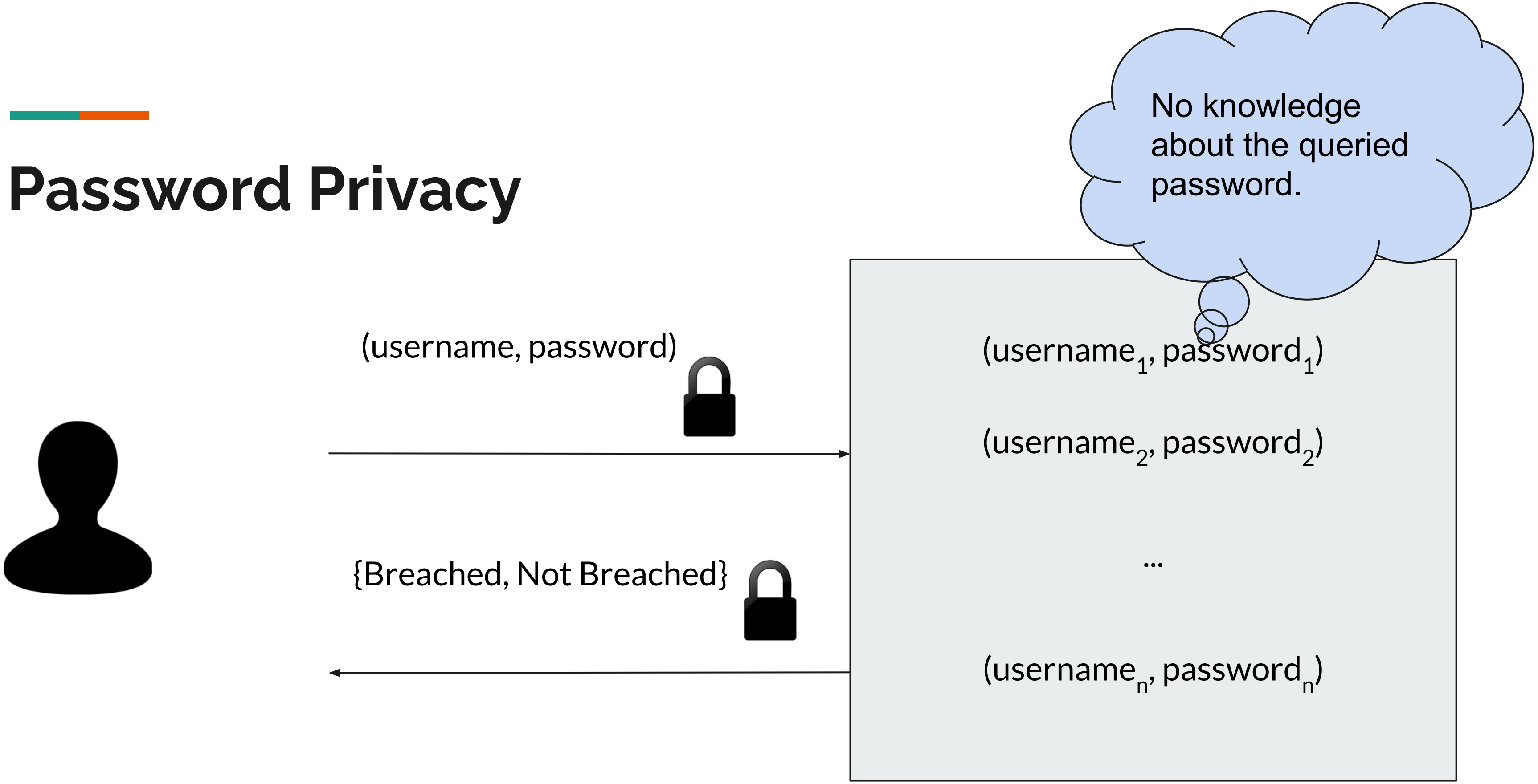
Design Principles

Password Privacy

Password Privacy



Password Privacy



Design Principles

Password Privacy

Design Principles

Password Privacy

Mitigate Abuse Risk

Mitigate Abuse Risk

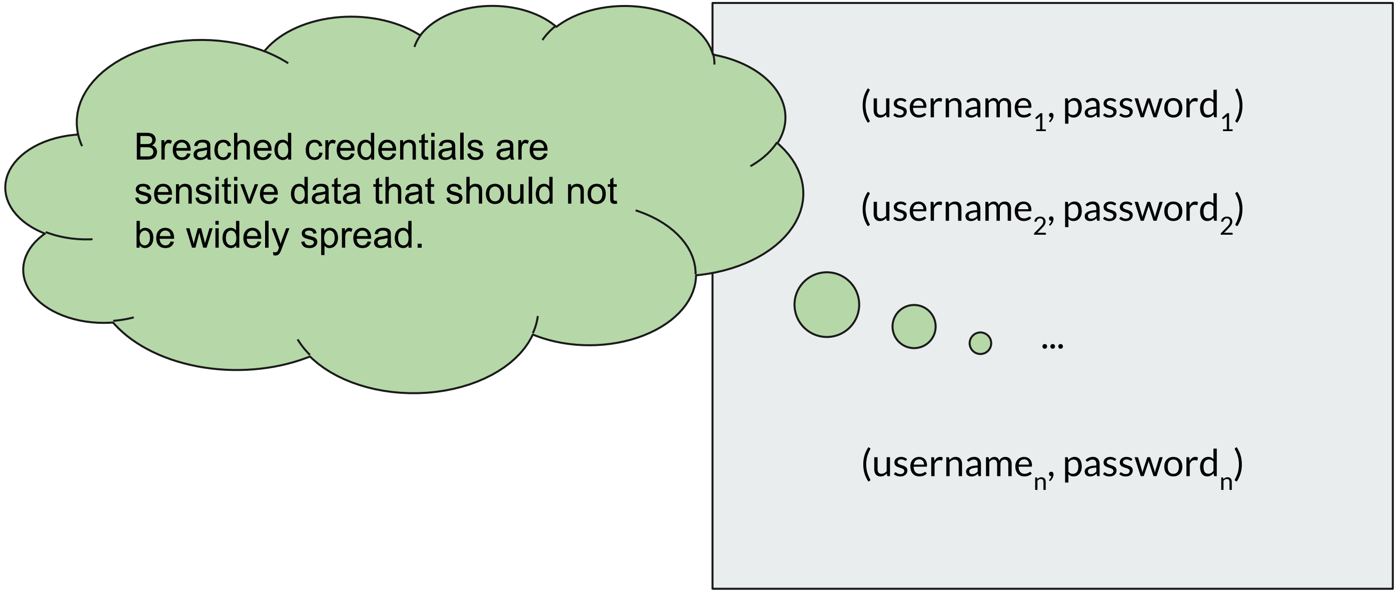
($username_1$, $password_1$)

($username_2$, $password_2$)

...

($username_n$, $password_n$)

Mitigate Abuse Risk



Mitigate Abuse Risk

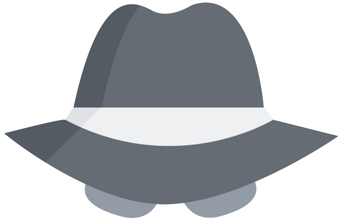
($username_1$, $password_1$)

($username_2$, $password_2$)

...

($username_n$, $password_n$)

Mitigate Abuse Risk



$(username_1, password_1)$

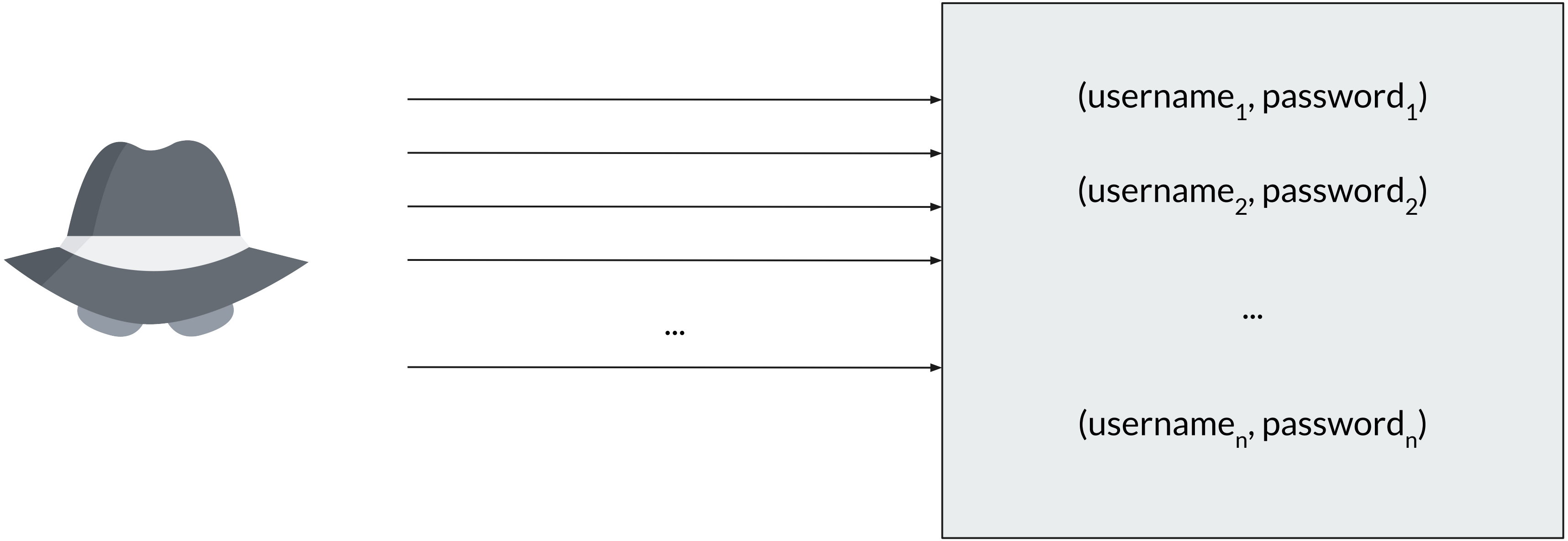
$(username_2, password_2)$

...

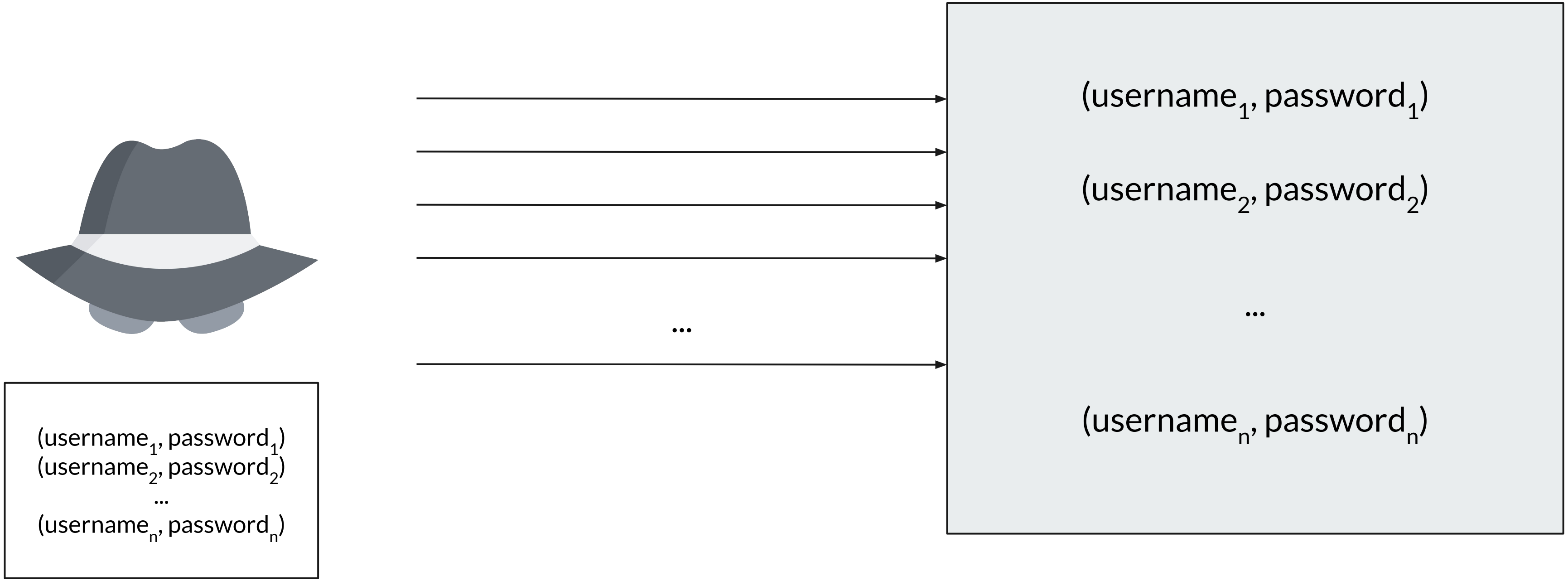
$(username_n, password_n)$

Mitigate Abuse Risk

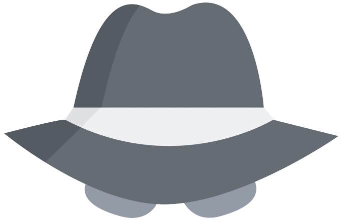
Mitigate Abuse Risk



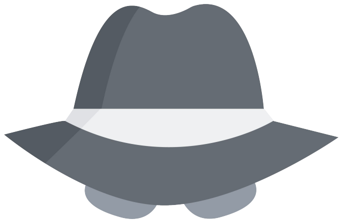
Mitigate Abuse Risk



Mitigate Abuse Risk



Mitigate Abuse Risk



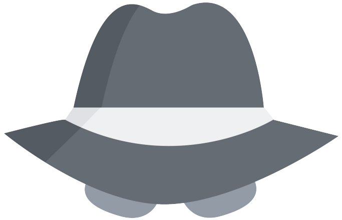
($username_1$, $password_1$)

($username_2$, $password_2$)

...

($username_n$, $password_n$)

Mitigate Abuse Risk



($username_1$, $password_1$)

($username_2$, $password_2$)

...

($username_n$, $password_n$)

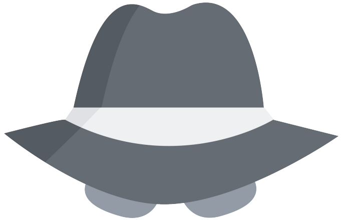
Mitigate Abuse Risk

(username$_1$, password$_1$)

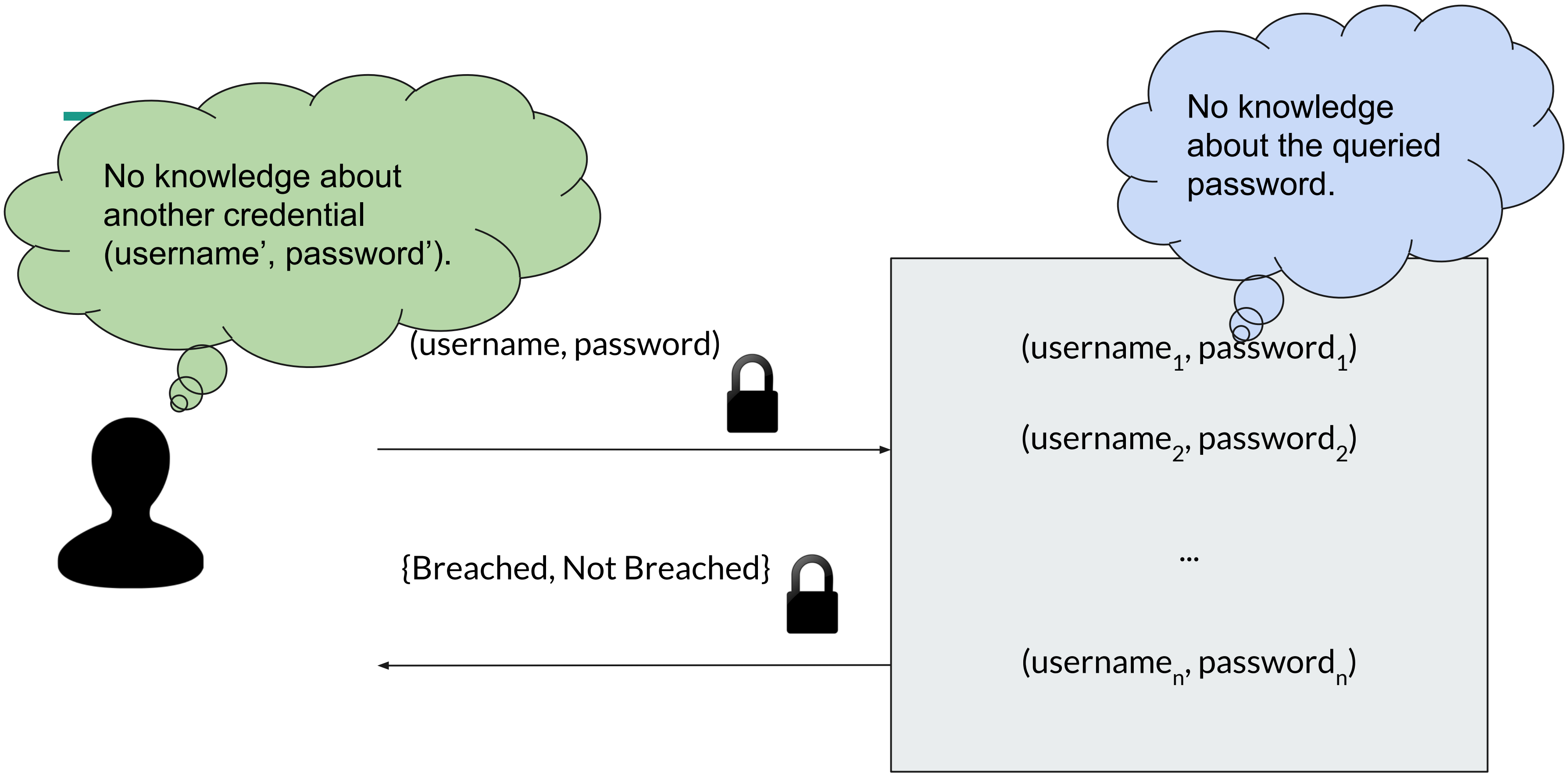
(username$_2$, password$_2$)

...

(username$_n$, password$_n$)



Cost(Abusing API) > Cost(Dark Web)


No knowledge about another credential (username', password').
(username, password)
{Breached, Not Breached}
No knowledge about the queried password.
(username$_1$, password$_1$)
(username$_2$, password$_2$)
...
(username$_n$, password$_n$)

Design Principles

Password Privacy

Mitigate Abuse Risk

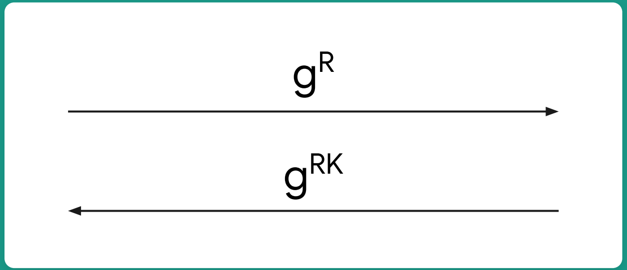
Design Principles

Password Privacy

Mitigate Abuse Risk

Simple but Effective Crypto

Protocol for Password Breach Alerting

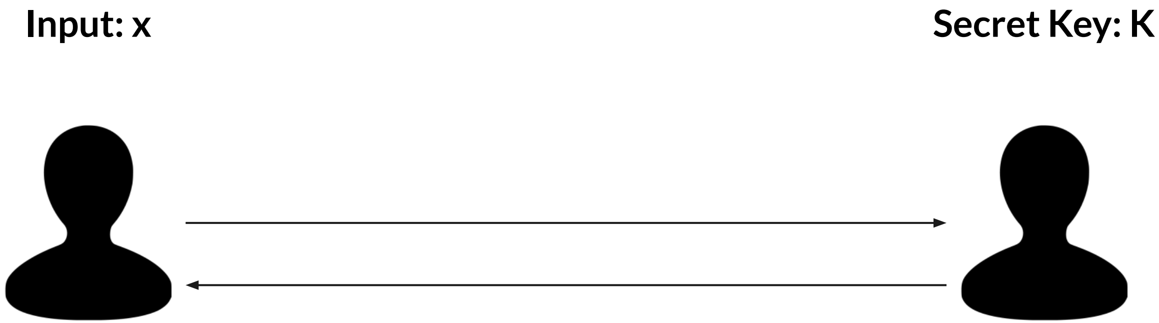


Tools: Oblivious PRF

Input: x

Secret Key: K

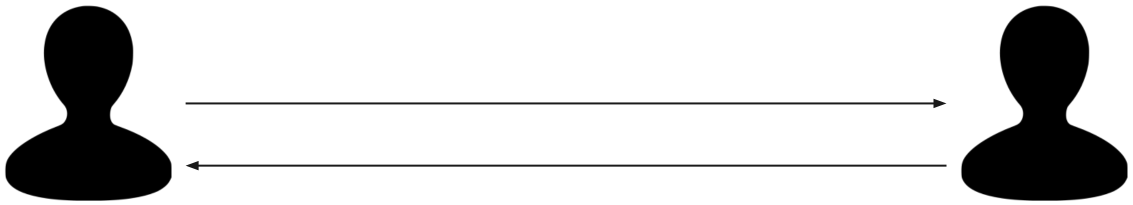
Tools: Oblivious PRF



Tools: Oblivious PRF

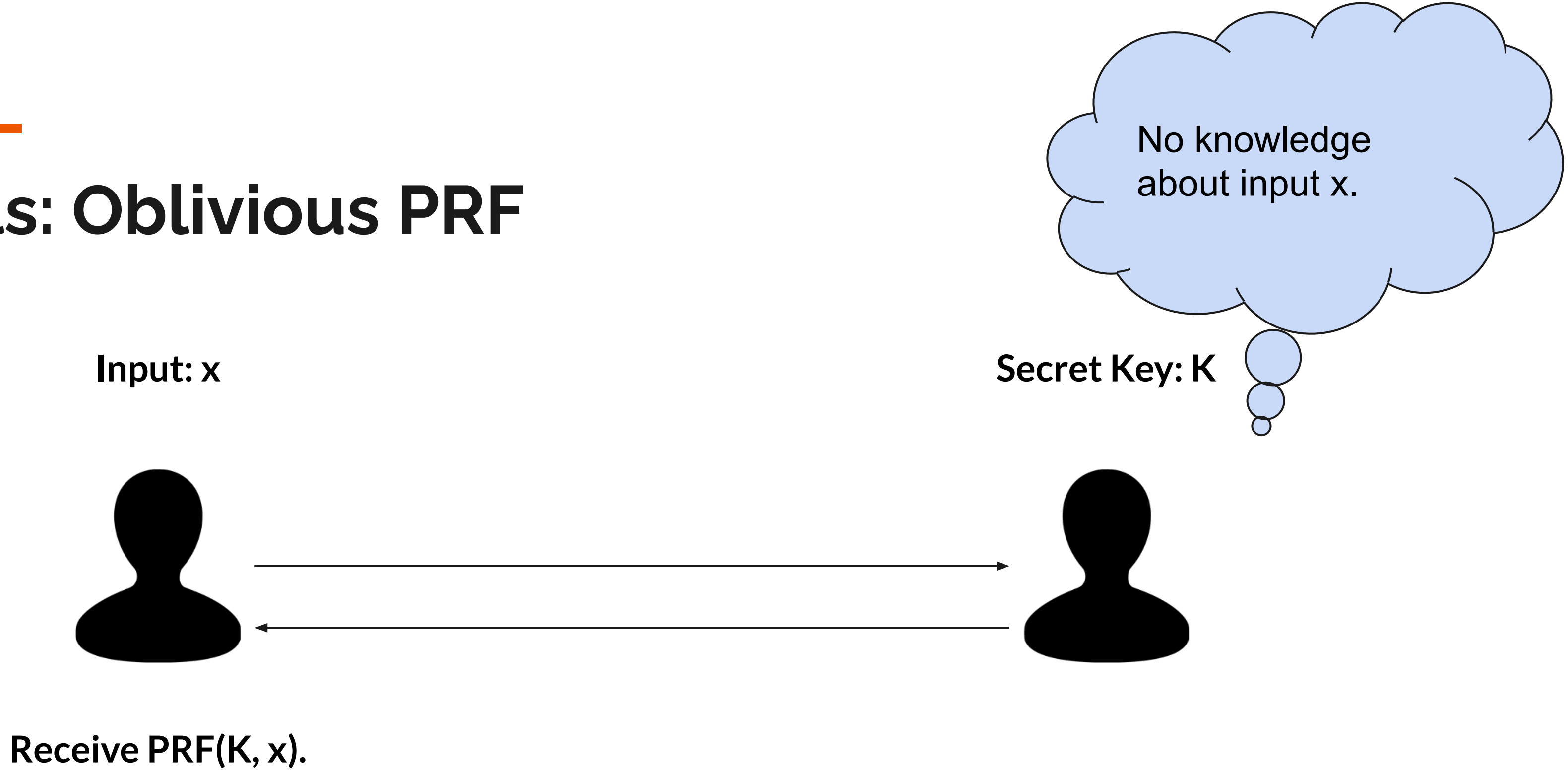
Input: x

Secret Key: K

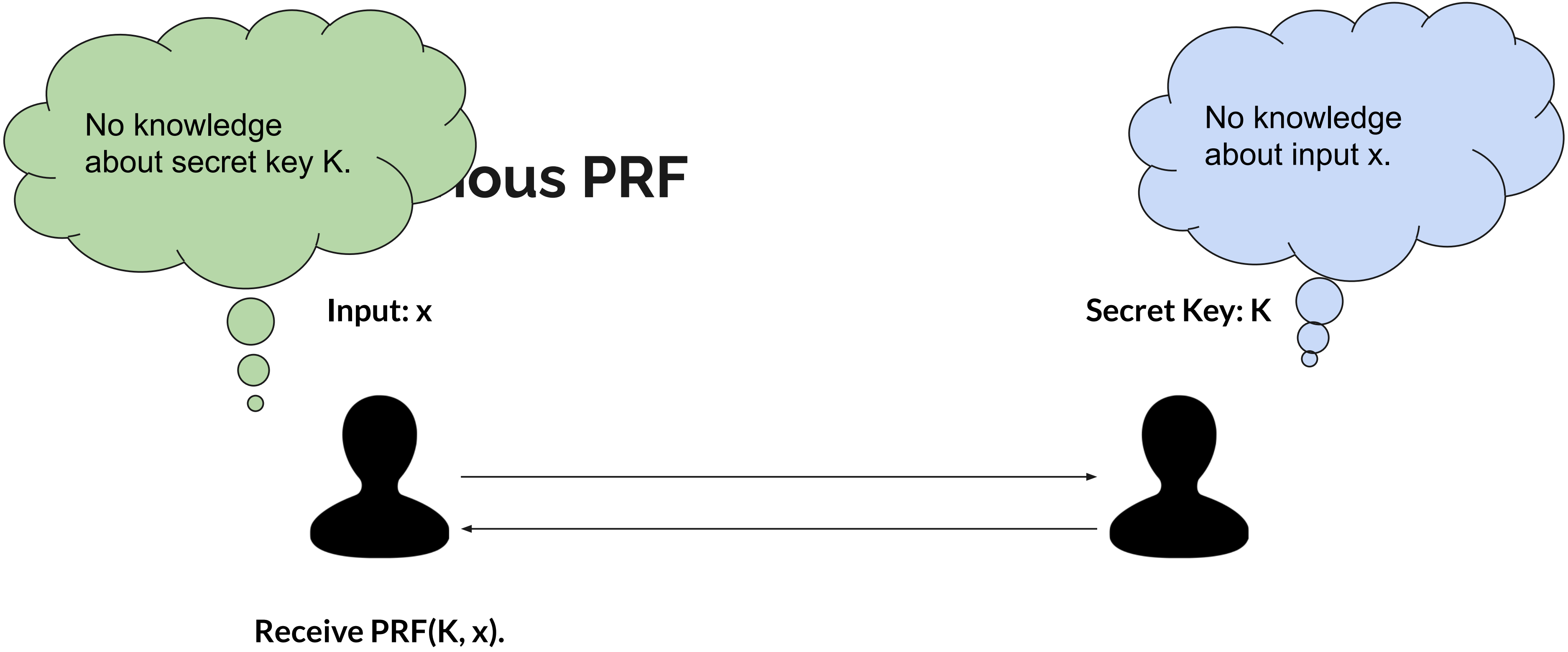


Receive PRF(K, x).

Tools: Oblivious PRF



ious PRF



No knowledge
about secret key K.
Input: x
No knowledge
about input x.
Secret Key: K
Receive PRF(K, x).

Tools: Oblivious PRF Protocol

Input: x

Secret Key: K

H hashes to an Elliptic Curve

Tools: Oblivious PRF Protocol

Input: x

Secret Key: K

H hashes to an Elliptic Curve

Tools: Oblivious PRF Protocol

Input: x

Secret Key: K

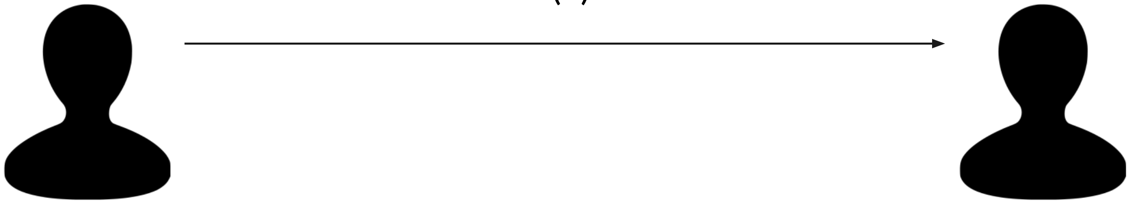
Generate ephemeral key R.

H hashes to an Elliptic Curve

Tools: Oblivious PRF Protocol

Input: x **Secret Key: K**

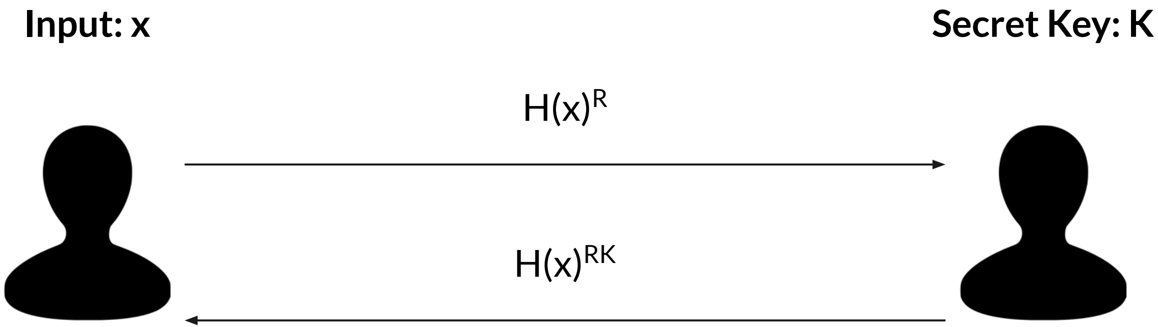
$H(x)^R$



Generate ephemeral key R.

H hashes to an Elliptic Curve

Tools: Oblivious PRF Protocol

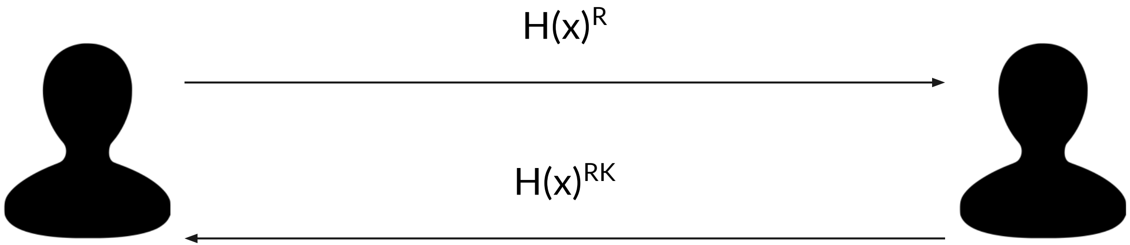


Generate ephemeral key R.

H hashes to an Elliptic Curve

Tools: Oblivious PRF Protocol

Input: x **Secret Key: K**



Generate ephemeral key R.

$OPRF(K, x) = (H(x)^{RK})^{1/R} = H(x)^K$

Password Breach Protocol

Password Breach Protocol

u_1, p_1

u_2, p_2

...

u_n, p_n

Step 1: Preprocess (u, p) pairs

Step 1: Preprocess (u, p) pairs

Secret Server Key: K

Step 1: Preprocess (u, p) pairs

Secret Server Key: K

u_1, OPRF(K, $u_1 \| p_1$)

u_2, OPRF(K, $u_2 \| p_2$)

...

u_n, OPRF(K, $u_n \| p_n$)

H' hashes to an Elliptic Curve

Step 1: Preprocess (u, p) pairs

H’ hashes to an Elliptic Curve

Step 1: Preprocess (u, p) pairs

u_1, H’(u_1, p_1)

u_2, H’(u_2, p_2)

...

u_n, H’(u_n, p_n)

H' hashes to an Elliptic Curve

Step 1: Preprocess (u, p) pairs

Secret Server Key: K

$u_1, H'(u_1, p_1)$

$u_2, H'(u_2, p_2)$

...

$u_n, H'(u_n, p_n)$

H' hashes to an Elliptic Curve

Step 1: Preprocess (u, p) pairs

Secret Server Key: K

u_1, $H'(u_1, p_1)^K$

u_2, $H'(u_2, p_2)^K$

...

u_n, $H'(u_n, p_n)^K$

Step 2: Preprocess u

Secret Server Key: K

u_1, $H'(u_1, p_1)^K$

u_2, $H'(u_2, p_2)^K$

...

u_n, $H'(u_n, p_n)^K$

Step 2: Preprocess u

H hashes to a string

Secret Server Key: K

$u_1, H'(u_1, p_1)^K$

$u_2, H'(u_2, p_2)^K$

…

$u_n, H'(u_n, p_n)^K$

Step 2: Preprocess u

H hashes to a string

Secret Server Key: K

$\mathbf{H(u_1)}$, $H'(u_1, p_1)^K$

$\mathbf{H(u_2)}$, $H'(u_2, p_2)^K$

…

$\mathbf{H(u_n)}$, $H'(u_n, p_n)^K$

Step 2: Preprocess u

Secret Server Key: K

$\mathbf{H(u_1)_{0...B-1}}$, $H'(u_1, p_1)^K$

$\mathbf{H(u_2)_{0...B-1}}$, $H'(u_2, p_2)^K$

. . .

$\mathbf{H(u_n)_{0...B-1}}$, $H'(u_n, p_n)^K$

Step 2: Preprocess u

Secret Server Key: K

110...1, $H'(u_1, p_1)^K$

010...0, $H'(u_2, p_2)^K$

...

110...1, $H'(u_n, p_n)^K$

Step 2: Preprocess u

Secret Server Key: K

110...1, $H'(u_1, p_1)^K$

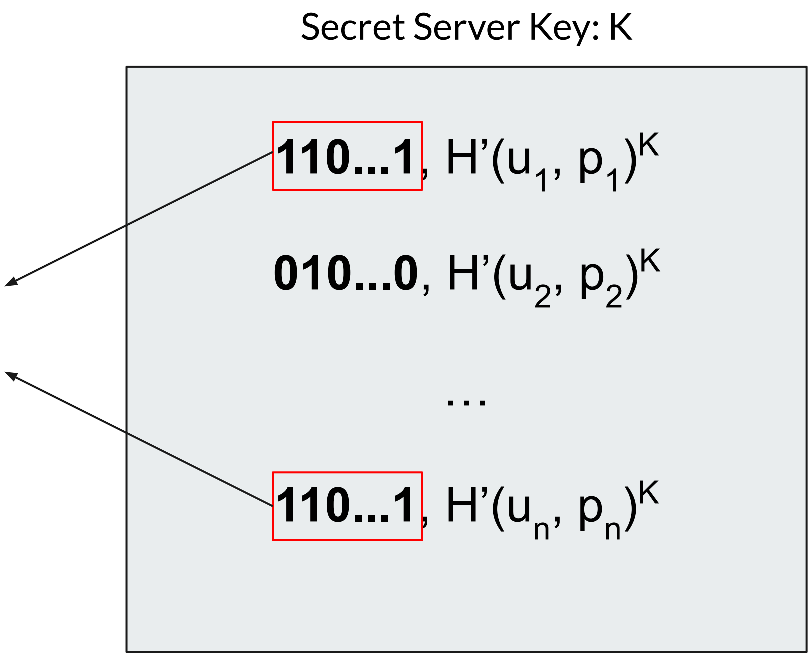
010...0, $H'(u_2, p_2)^K$

...

110...1, $H'(u_n, p_n)^K$

Step 2: Preprocess u

Bucketize all OPRF evaluations according to username hash prefix.



Step 2: Preprocess u

Secret Server Key: K

000...0: $H'(u_3, p_3)^K$, $H'(u_{16}, p_{16})^K$, ...

000...1: $H'(u_7, p_7)^K$, $H'(u_{13}, p_{13})^K$, ...

...

111...1: $H'(u_4, p_4)^K$, $H'(u_{19}, p_{19})^K$, ...

Query Algorithm

Secret Server Key: K

$000...0: H'(u_3, p_3)^K, H(u_{16}, p_{16})^K, ...$

$000...1: H'(u_7, p_7)^K, H(u_{13}, p_{13})^K, ...$

...

$111...1: H'(u_4, p_4)^K, H(u_{19}, p_{19})^K, ...$

Query Algorithm

Secret Server Key: K

000...0: $H'(u_3, p_3)^K$, $H(u_{16}, p_{16})^K$, ...

000...1: $H'(u_7, p_7)^K$, $H(u_{13}, p_{13})^K$, ...

...

111...1: $H'(u_4, p_4)^K$, $H(u_{19}, p_{19})^K$, ...

Query Algorithm

(u, p)

Secret Server Key: K

000...0: $H'(u_3, p_3)^K$, $H(u_{16}, p_{16})^K$, ...

000...1: $H'(u_7, p_7)^K$, $H(u_{13}, p_{13})^K$, ...

...

111...1: $H'(u_4, p_4)^K$, $H(u_{19}, p_{19})^K$, ...

Query Algorithm

(u, p)

User Secret Key: R

Secret Server Key: K

000...0: $H'(u_3, p_3)^K$, $H(u_{16}, p_{16})^K$, ...

000...1: $H'(u_7, p_7)^K$, $H(u_{13}, p_{13})^K$, ...

...

111...1: $H'(u_4, p_4)^K$, $H(u_{19}, p_{19})^K$, ...

Query Algorithm

(u, p)

User Secret Key: R

$H(u)_{0...B-1}$ →

Secret Server Key: K

000...0: $H'(u_3, p_3)^K$, $H(u_{16}, p_{16})^K$, ...

000...1: $H'(u_7, p_7)^K$, $H(u_{13}, p_{13})^K$, ...

...

111...1: $H'(u_4, p_4)^K$, $H(u_{19}, p_{19})^K$, ...

Query Algorithm

Secret Server Key: K

(u, p)

User Secret Key: R

$H(u)_{0...B-1}$

OPRF(u, p) Request →

$000...0: H'(u_3, p_3)^K, H(u_{16}, p_{16})^K, ...$

$000...1: H'(u_7, p_7)^K, H(u_{13}, p_{13})^K, ...$

...

$111...1: H'(u_4, p_4)^K, H(u_{19}, p_{19})^K, ...$

Query Algorithm

(u, p)

User Secret Key: R

$H(u)_{0 \ldots B-1}$

$H'(u, p)^R$

Secret Server Key: K

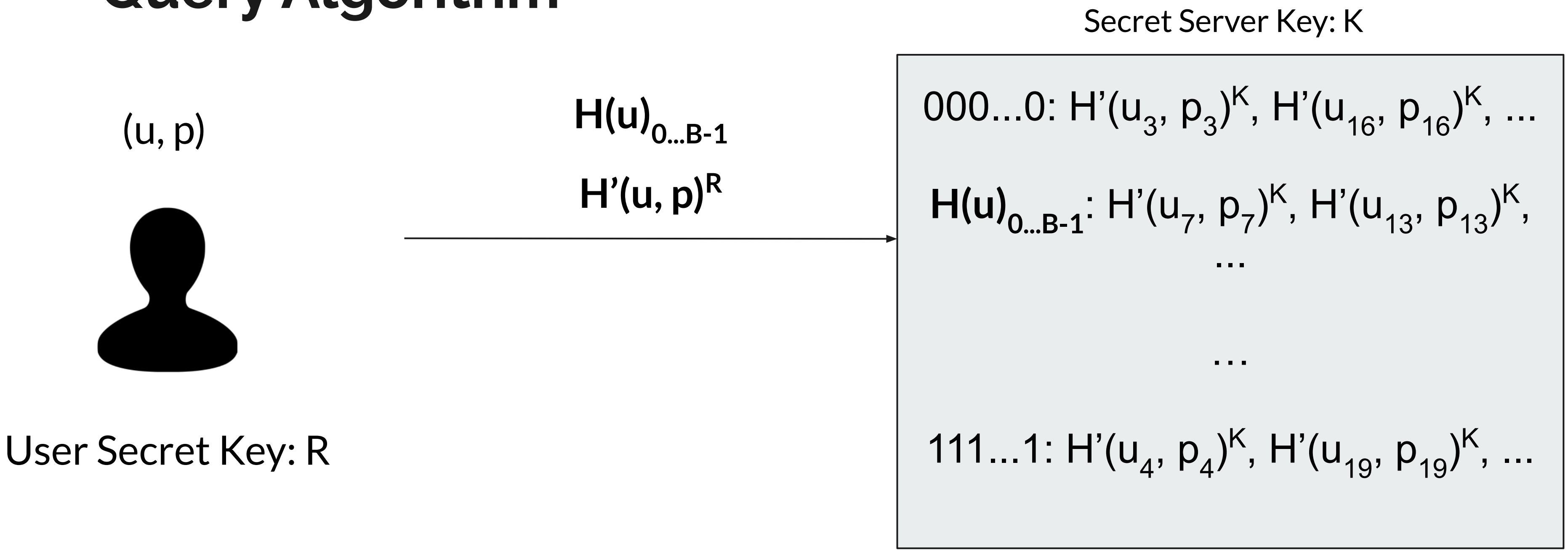
$000 \ldots 0$: $H'(u_3, p_3)^K$, $H(u_{16}, p_{16})^K$, ...

$000 \ldots 1$: $H'(u_7, p_7)^K$, $H(u_{13}, p_{13})^K$, ...

...

$111 \ldots 1$: $H'(u_4, p_4)^K$, $H(u_{19}, p_{19})^K$, ...

Query Algorithm



Query Algorithm

(u, p)

User Secret Key: R

$\mathbf{H(u)_{0...B-1}}$

$\mathbf{H'(u, p)^R}$

→

$\mathbf{\{H'(u_7, p_7)^K, H'(u_{13}, p_{13})^K, \ldots\}}$

←

Secret Server Key: K

$000...0: H'(u_3, p_3)^K, H'(u_{16}, p_{16})^K, \ldots$

$\mathbf{H(u)_{0...B-1}}: H'(u_7, p_7)^K, H'(u_{13}, p_{13})^K, \ldots$

…

$111...1: H'(u_4, p_4)^K, H'(u_{19}, p_{19})^K, \ldots$

Query Algorithm

(u, p)

User Secret Key: R

$\mathbf{H(u)_{0...B-1}}$

$\mathbf{H'(u, p)^R}$

$\mathbf{\{H'(u_7, p_7)^K, H'(u_{13}, p_{13})^K, \ldots\}}$

OPRF(K, u, p) Response

Secret Server Key: K

000...0: $H'(u_3, p_3)^K$, $H'(u_{16}, p_{16})^K$, ...

$\mathbf{H(u)_{0...B-1}}$: $H'(u_7, p_7)^K$, $H'(u_{13}, p_{13})^K$,

...

...

111...1: $H'(u_4, p_4)^K$, $H'(u_{19}, p_{19})^K$, ...

Query Algorithm

Secret Server Key: K

(u, p)

User Secret Key: R

$\mathbf{H(u)_{0 \ldots B-1}}$

$\mathbf{H'(u, p)^R}$

→

$\mathbf{\{H'(u_7, p_7)^K, H'(u_{13}, p_{13})^K, \ldots\}}$

$\mathbf{H'(u, p)^{RK}}$

←

$000\ldots0$: $H'(u_3, p_3)^K, H'(u_{16}, p_{16})^K, \ldots$

$\mathbf{H(u)_{0 \ldots B-1}}$: $H'(u_7, p_7)^K, H'(u_{13}, p_{13})^K, \ldots$

…

$111\ldots1$: $H'(u_4, p_4)^K, H'(u_{19}, p_{19})^K, \ldots$

Query Algorithm

(u, p)

User Secret Key: R

$\{H'(u_7, p_7)^K, H'(u_{13}, p_{13})^K, \ldots\}$
$H'(u, p)^{RK}$

←

Query Algorithm

(u, p)

User Secret Key: R

1. Compute
OPRF(K, u || p) = H'(u, p)^K.

$\{H'(u_7, p_7)^K, H'(u_{13}, p_{13})^K, \ldots\}$
$H'(u, p)^{RK}$

←

Query Algorithm

(u, p)

User Secret Key: R

1. Compute
$OPRF(K, u \,||\, p) = H'(u, p)^K$.

2. (u, p) is breached iff
$H'(u, p)^K \in \{H'(u_7, p_7)^K, \ldots\}$.

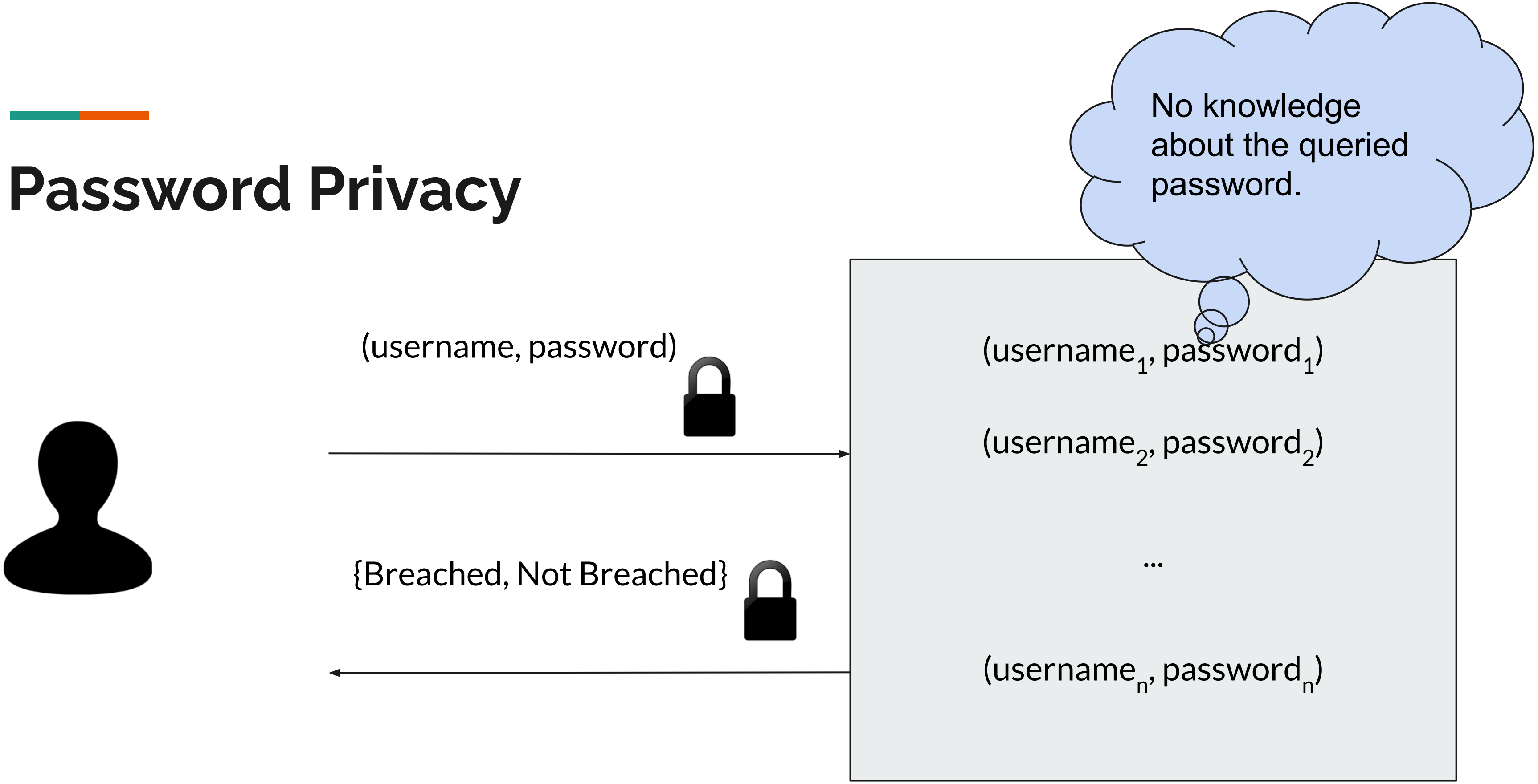
$\{H'(u_7, p_7)^K, H'(u_{13}, p_{13})^K, \ldots\}$
$H'(u, p)^{RK}$

Privacy Guarantees

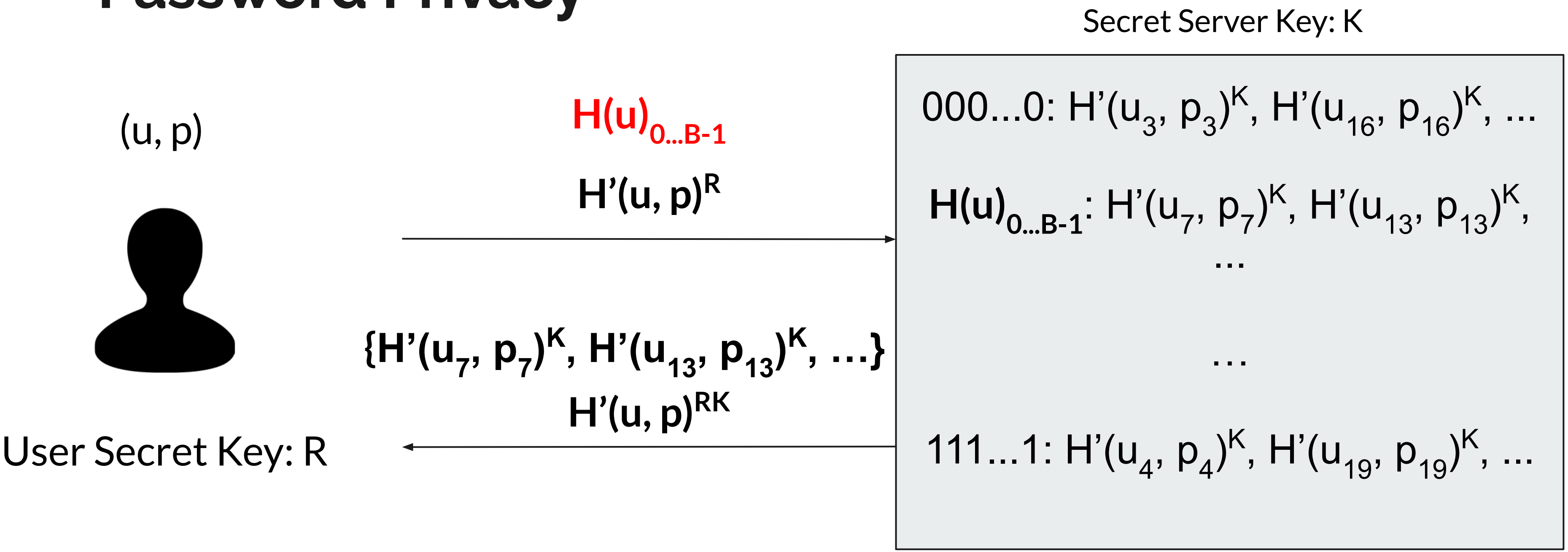
Privacy Guarantees

1. **Full privacy for passwords**
2. K-anonymity for usernames
3. Abuse mitigations

Password Privacy



Password Privacy



Password Privacy

Secret Server Key: K

(u, p)

User Secret Key: R

$\mathbf{H(u)_{0...B-1}}$

$H'(u, p)^R$

$\{H'(u_7, p_7)^K, H'(u_{13}, p_{13})^K, \ldots\}$

$H'(u, p)^{RK}$

$000...0: H'(u_3, p_3)^K, H'(u_{16}, p_{16})^K, \ldots$

$\mathbf{H(u)_{0...B-1}}: H'(u_7, p_7)^K, H'(u_{13}, p_{13})^K, \ldots$

…

$111...1: H'(u_4, p_4)^K, H'(u_{19}, p_{19})^K, \ldots$

Privacy Guarantees

1. Full privacy for passwords
2. **K-anonymity for usernames**
3. Abuse mitigations

Username Privacy

Secret Server Key: K

(u, p)

User Secret Key: R

$H(u)_{0...B-1}$

$H'(u, p)^R$

$\{H'(u_7, p_7)^K, H'(u_{13}, p_{13})^K, ...\}$

$H'(u, p)^{RK}$

$000...0: H'(u_3, p_3)^K, H'(u_{16}, p_{16})^K, ...$

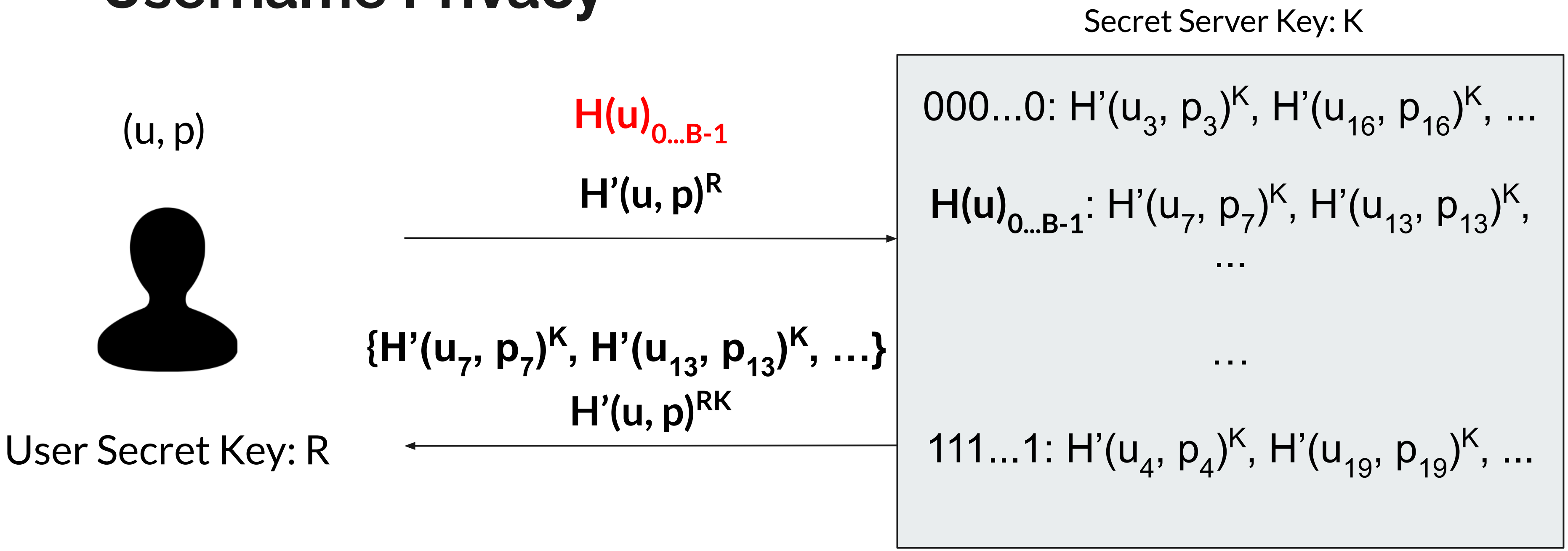
$H(u)_{0...B-1}: H'(u_7, p_7)^K, H'(u_{13}, p_{13})^K,$

...

...

$111...1: H'(u_4, p_4)^K, H'(u_{19}, p_{19})^K, ...$

Username Privacy



K-anonymity of usernames

For any username u', the server can use $H(u)_{0...B-1}$ to check:

- $H(u)_{0...B-1} \neq H(u')_{0...B-1} \rightarrow$ u' was not queried
- $H(u)_{0...B-1} = H(u')_{0...B-1} \rightarrow$ u' may have been queried

The set $\{u' \mid H(u)_{0...B-1} = H(u')_{0...B-1}\}$ is the anonymity set. All usernames in anonymity set were possible.

The parameter K refers to the size of the anonymity set.

K-anonymity of usernames

How big is the anonymity set (i.e., how large is K)?

- **Local View:** Consider only known usernames (such as only those in the breached database).
- **Universal View**: Consider all possible usernames (i.e., all possible strings of some length).

How to choose hash prefix length (B)?

Smaller B implies:

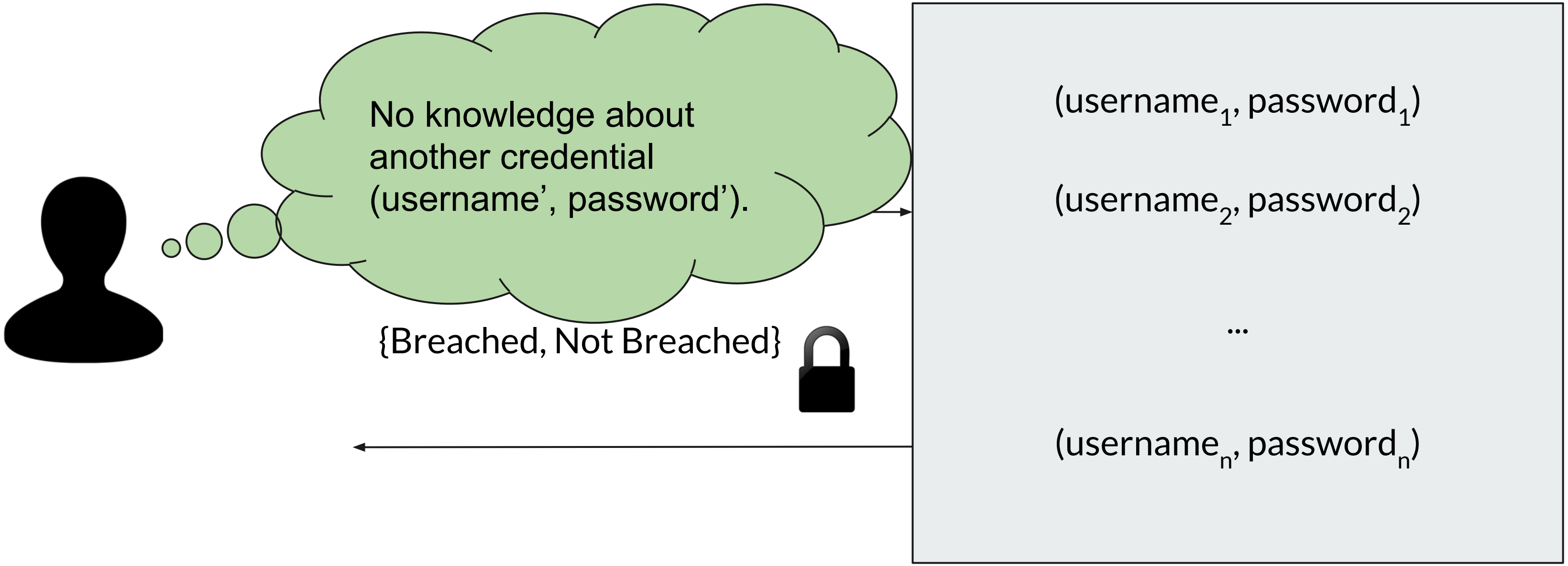
- Larger anonymity sets
- Larger communication

Choose B as small as possible while satisfying efficiency requirements.

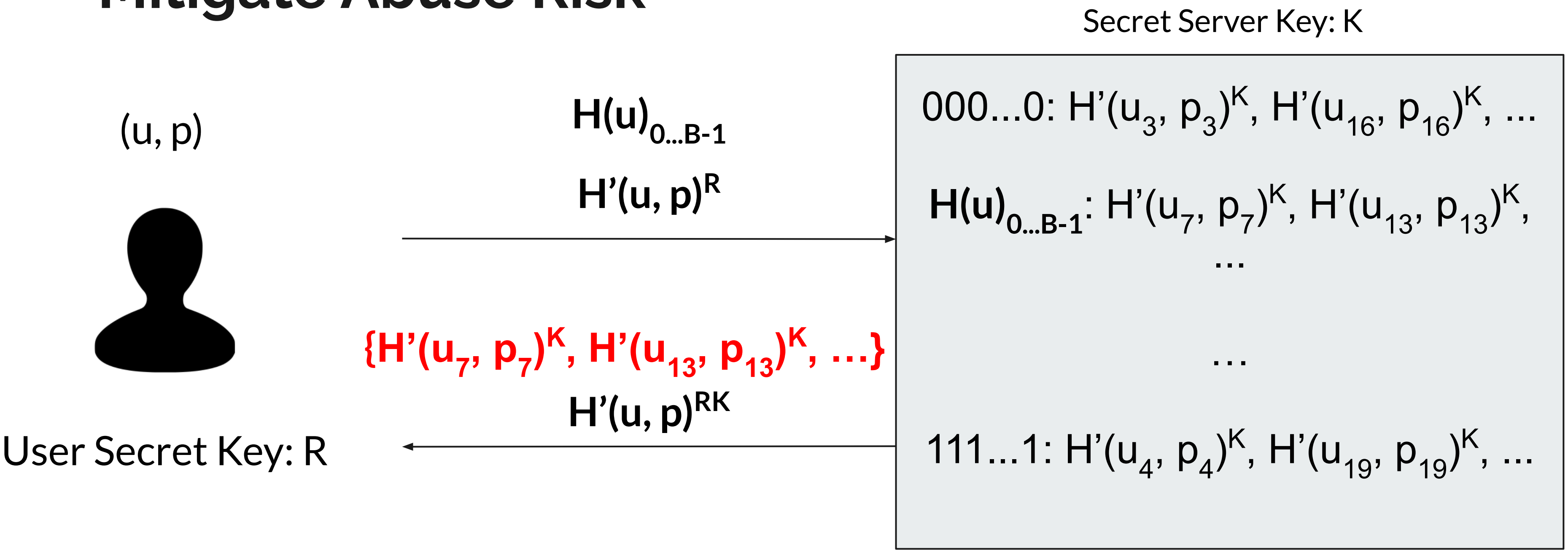
Privacy Guarantees

1. Full privacy for passwords
2. K-anonymity for usernames
3. **Abuse mitigations**

Mitigate Abuse Risk



Mitigate Abuse Risk



Mitigate Abuse Risk

(u, p)

User Secret Key: R

$H(u)_{0...B-1}$

$H'(u, p)^R$

→

$\{H'(u_7, p_7)^K, H'(u_{13}, p_{13})^K, ...\}$

$H'(u, p)^{RK}$

←

Secret Server Key: K

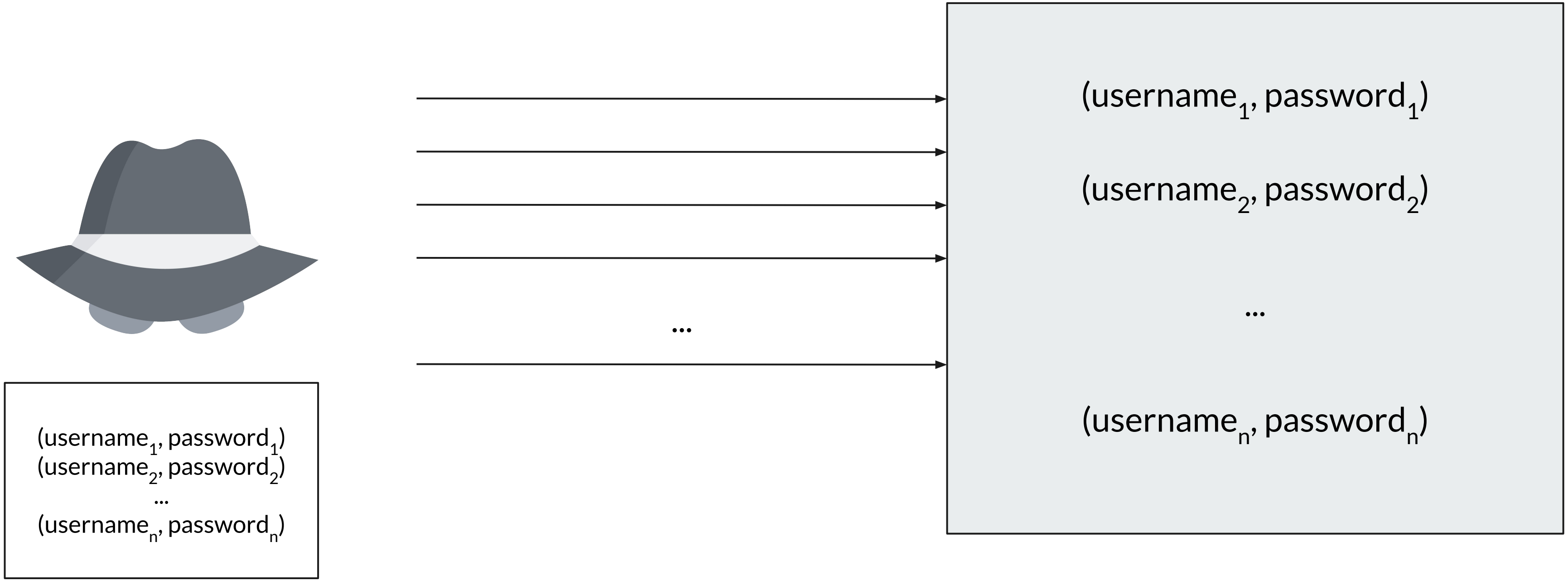
$000...0: H'(u_3, p_3)^K, H'(u_{16}, p_{16})^K, ...$

$H(u)_{0...B-1}: H'(u_7, p_7)^K, H'(u_{13}, p_{13})^K, ...$

...

$111...1: H'(u_4, p_4)^K, H'(u_{19}, p_{19})^K, ...$

Brute Force Queries

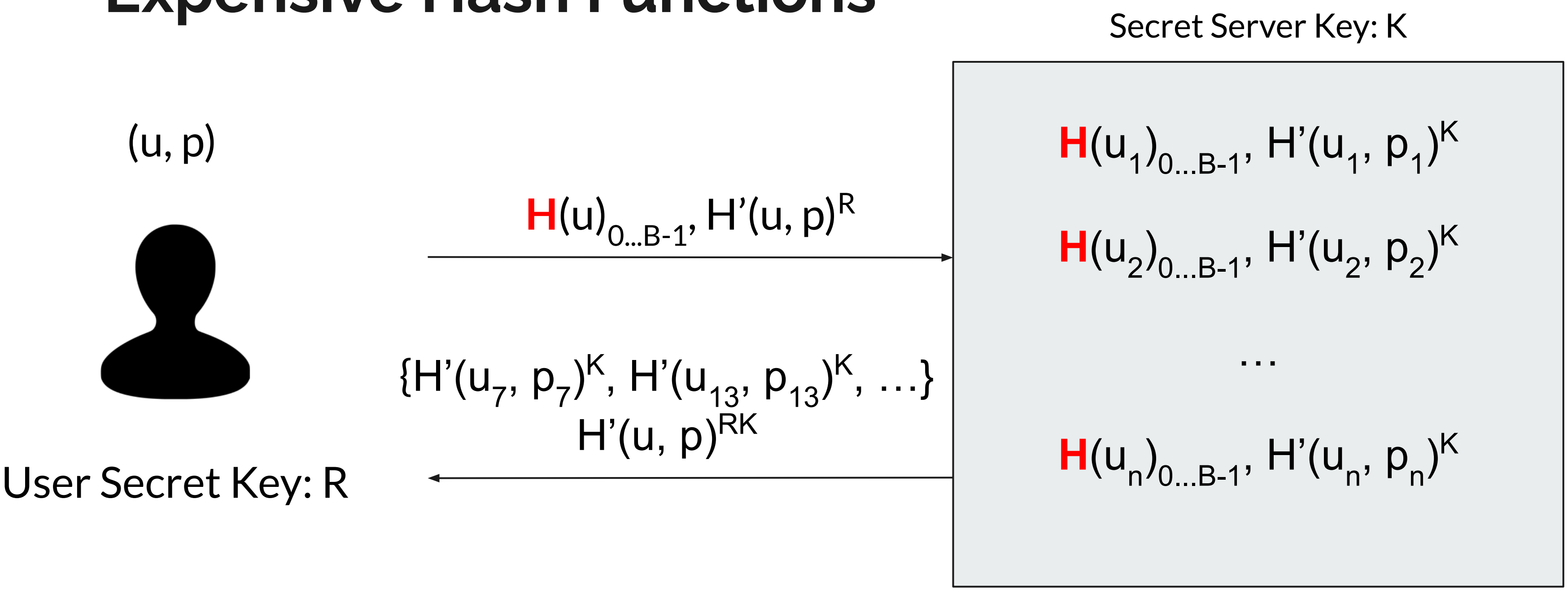


Brute Force Prevention

1. **Expensive hash functions**
2. Rate-limiting

H is Argon2

Expensive Hash Functions



Brute Force Prevention

1. Expensive hash functions
2. **Rate-limiting**

Rate-Limiting

- Limit to X queries/hour using some criteria such as:
  - Signed-in account
  - Username Bucket
  - IP Address
  - ...
- For example, using signed-in accounts reduces problem to creating accounts.

Brute Force Prevention

1. Expensive hash functions
   a. Requires large computation on user devices
2. Rate-limiting
   a. No additional computation on user device
   b. Better for older or lower-end devices

Deployment at Google

Third-party breach corpus

4 billion

Unique usernames and passwords from breaches

Thomas et al. Data breaches, phishing, or malware? ... CCS'17.

Protecting users across Google products

1. Chrome extension (sunset)
2. Chrome browser
3. Google password manager
4. Android

Chrome Browser Extension

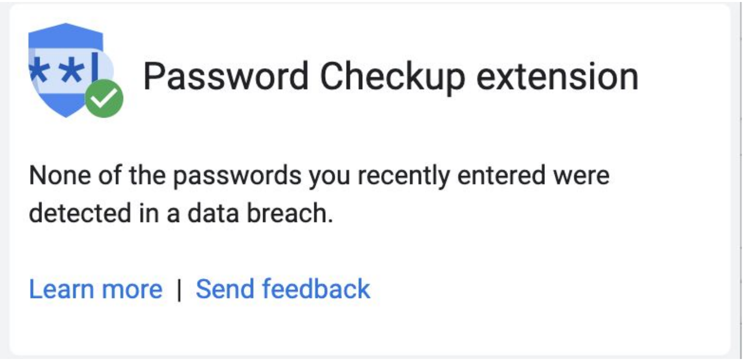
Password Checkup extension

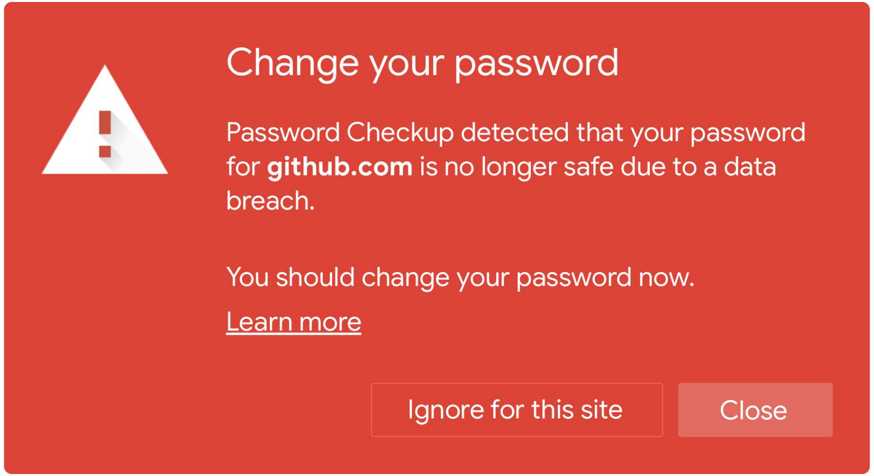
Offered by: google.com

★★★★☆ 269 | Productivity | 1,003,965 users

By Google

Chrome Browser Extension





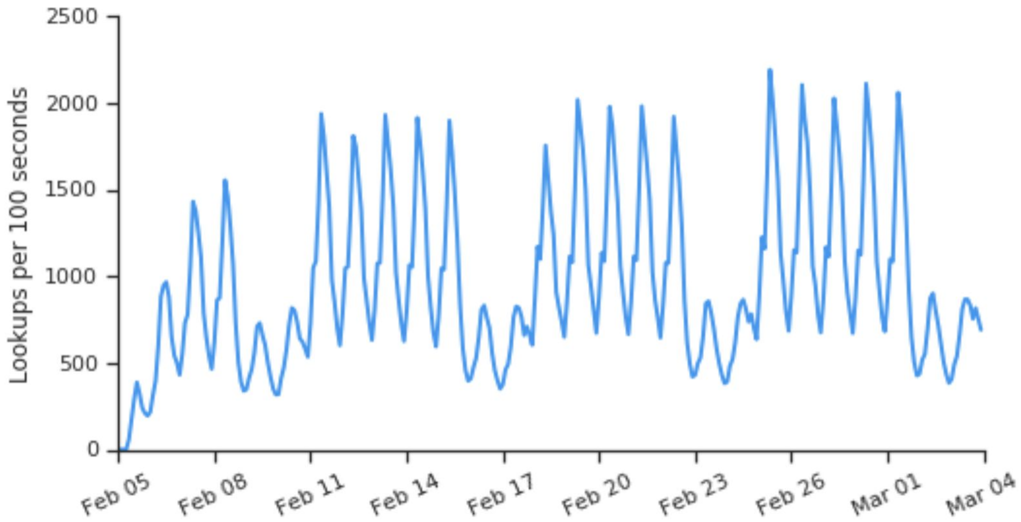
Anonymous telemetry reported

On login: domain, timestamp, breach status, hashing performance

On ignore: domain, timestamp

On password change: domain, timestamp, original strength, new strength (zxcvbn)

Performance Metrics of Extension



20 QPS

At peak usage

8.5s

Per query,
half spent on proof of work

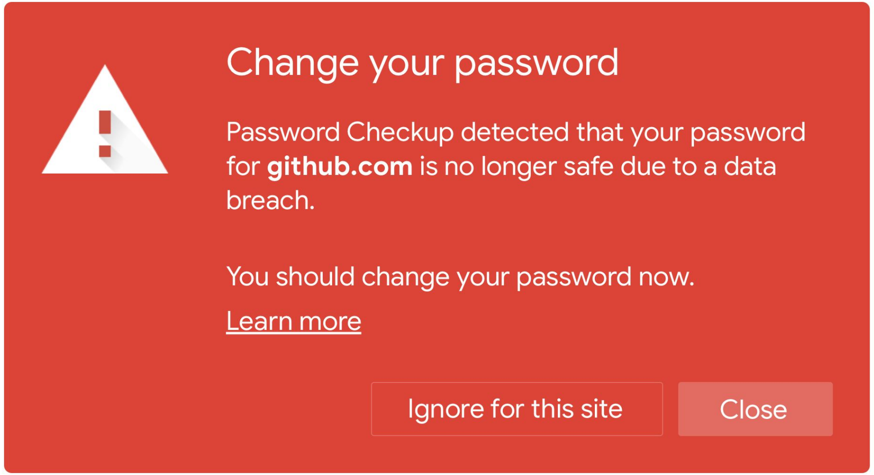
Frequency of Warnings by Extension

21 million
Logins scanned in first month

316,000
Breached credentials detected

1.5%
Logins on the web are breached

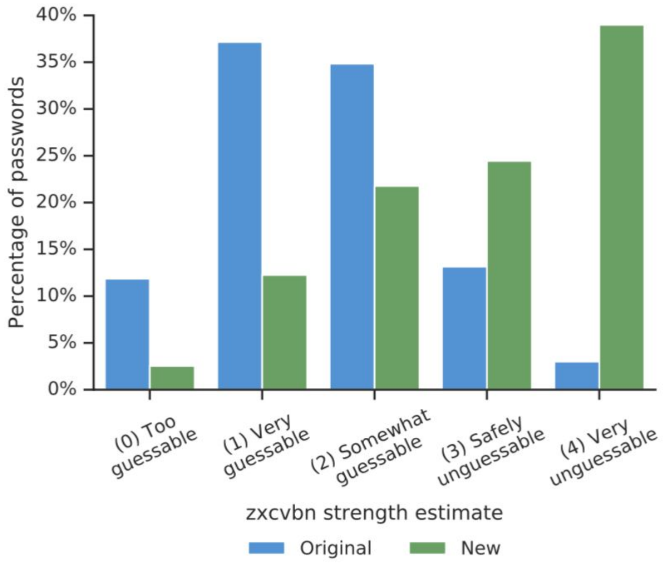
User Response to Warnings by Extension



26%

Breached credentials reset and resecured

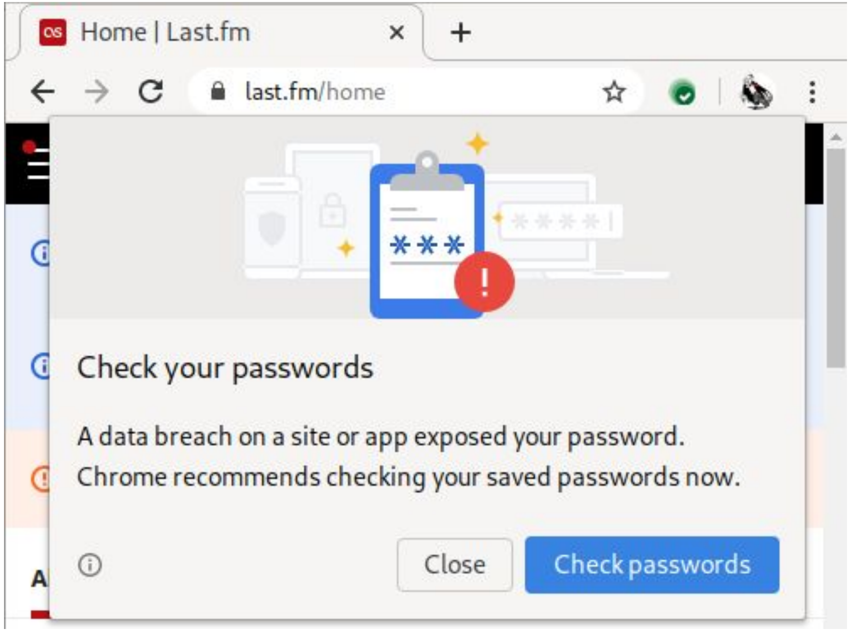
Passwords Improve After Warning and Update



94%

New passwords as strong or stronger than original

Chrome Browser



Google Password Manager

Google Account

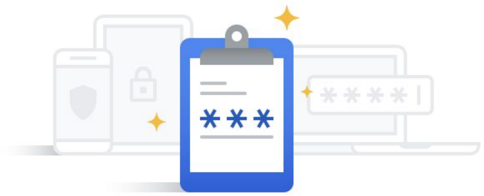
K

Password Manager

See, change, or remove passwords you saved in your Google Account. Learn more

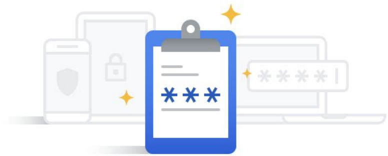
Password Checkup

Check your saved passwords to strengthen your security.



Check passwords

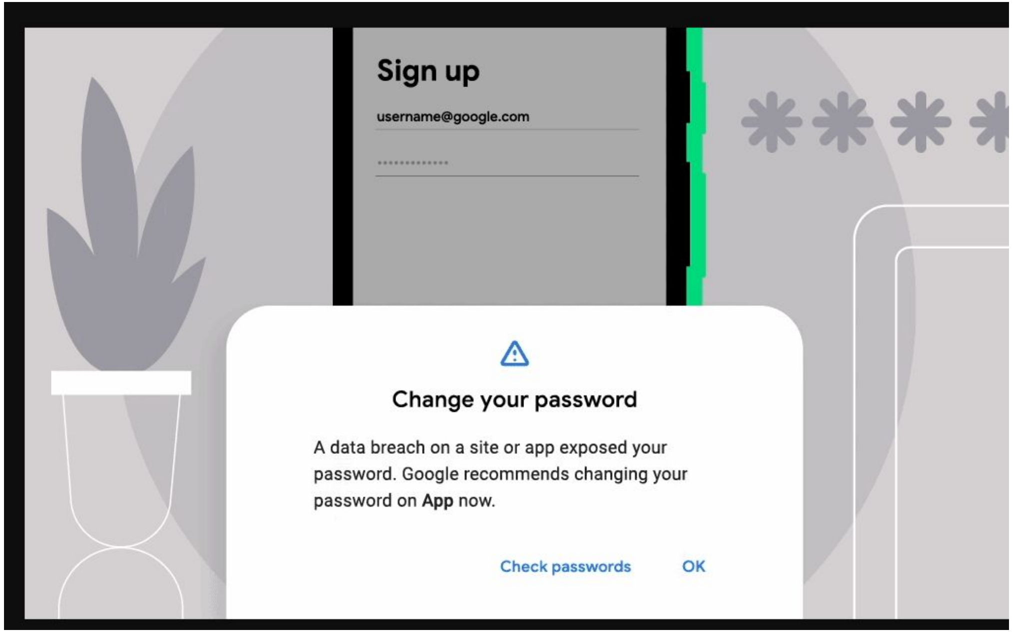
Google Password Manager



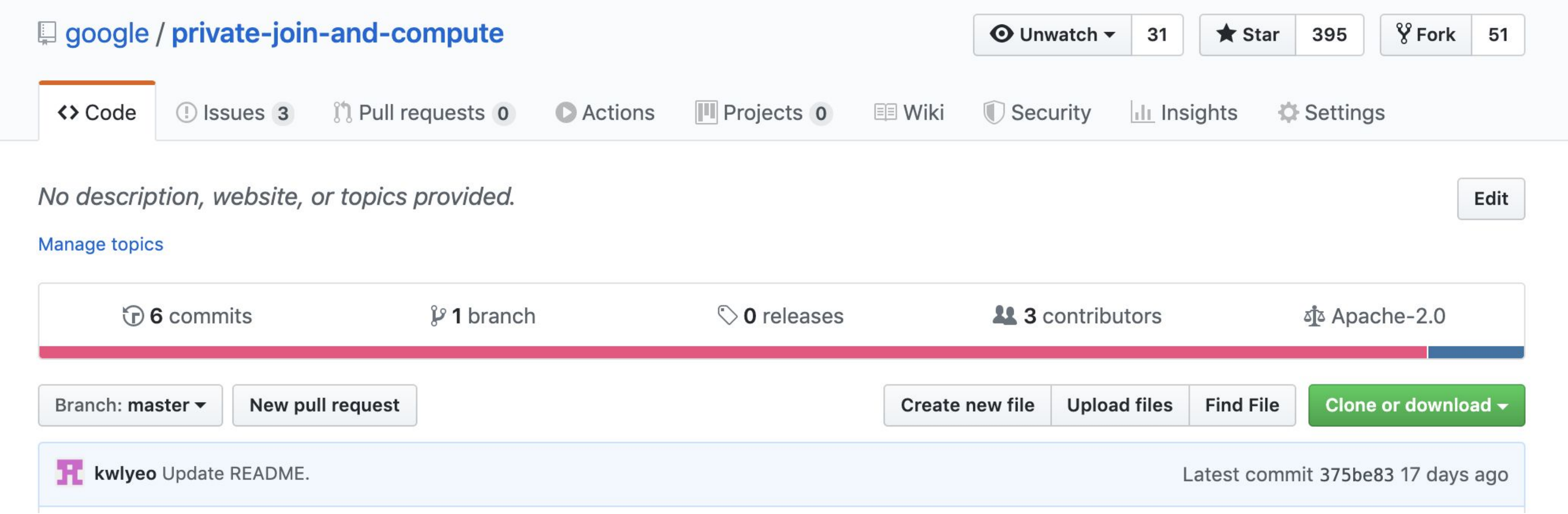
We analyzed your saved passwords and found the following issues

- No compromised passwords
- 2 reused passwords
  Create unique passwords
- 2 accounts using a weak password
  Create strong passwords

Android Autofill



Open Source


google / private-join-and-compute
Unwatch 31
Star 395
Fork 51
Code
Issues 3
Pull requests 0
Actions
Projects 0
Wiki
Security
Insights
Settings
No description, website, or topics provided.
Edit
Manage topics
6 commits
1 branch
0 releases
3 contributors
Apache-2.0
Branch: master
New pull request
Create new file
Upload files
Find File
Clone or download
kwlyeo Update README.
Latest commit 375be83 17 days ago

Open Source

chromium / chromium / src / refs/heads/master / . / components / password_manager / core / browser /
leak_detection

```
tree: 56cb379e8c52895bf7ea7892d39e71a04f06fcb9 [path history] [tgz]
```

- BUILD.gn
- DEPS
- authenticated_leak_check.cc
- authenticated_leak_check.h
- authenticated_leak_check_unittest.cc
- encryption_utils.cc
- encryption_utils.h
- encryption_utils_unittest.cc
- fuzzer/
- leak_detection_api.proto
- leak_detection_check.h
- leak_detection_check_factory.h
- leak_detection_check_factory_impl.cc
- leak_detection_check_factory_impl.h
- leak_detection_check_factory_impl_unittest.cc
- leak_detection_delegate_interface.h
- leak_detection_request.cc

Conclusions

Privacy-preserving password checkup is protecting billions of users from account hijacking.

Thanks! Questions?