Lecture 14:
Authentication
(Pt I: Passwords-based Protocols)

CPSC 4440/5440, 2026 Spring

Instructor: Fan Zhang

Yale

Computer Science

The story so far…

- TLS allows Alice to securely talk with bank.com
- But how does bank.com verify that Alice is Alice?
- This is the task of **authentication**.
- Critical for *both* Alice and bank.

Don’t we have AKEs?

- But AKE is rarely used to authenticate users. Why?
- One reason: Key management is **terribly difficult** for everyone users.

Key management dilemma

Recoverability: Ability to regain access

Security: Protection against theft

Need to back up keys (e.g., in the cloud)

How to authenticate to the backup system? (Perhaps another key?)

Attacker now has two places to steal keys from.

THE LEDGER • BITCOIN

Exclusive: Nearly 4 Million Bitcoins Lost Forever, New Study Says

BY **JEFF JOHN ROBERTS** AND **NICOLAS RAPP**

November 25, 2017 8:30 AM EST

General goals

- **Security** – only allow authorized users to access (certain systems, resources, etc)
- **Correctness/liveness** — authorized users can always access
- **Recoverability** — users can lose everything (her phones, laptop, keys to her safe) and she can still recover access.
- **Replaceability** — users can change or revoke credentials.
- **Intrusion detection** — users can detect unauthorized access
- The list goes on

Four mechanisms for user authentication

- Something you know/remember
  - Passwords, PINs, answers to security questions, last 4 digits of SSN
- Something you have
  - Keys, hardware tokens, mobile phones, smartcards
- Something you are
  - Fingerprint, voiceprint, iris code
- Somebody you know
  - “Social recovery”: Alice choose Bob and Charlie to be her “guardians”; They can access her account in case Alice can’t.

Chapter of
Identity and Credentials

- Today: **Password**-based auth protocols
- **Federated** authentication (e.g., SSO) & security and privacy problems therein
- **Anonymous** credentials
- Integration with modern systems (after the break):
  - Decentralized Identities in decentralized social media (BlueSky)
  - Self-credentials (permissionless account creation)
  - Key transparency logs in E2EE messaging
- Presentations of recent papers (second lecture after the break; list will be finalized by Monday.)

Passwords:

Can’t live with it, can’t live without it.

Passwords

- The only **recoverable** and **replaceable** auth method
- However, security pitfalls throughout the pipeline
  - ***Generation***: Humans are bad Random Number Generators
  - ***Storage*****:** servers need to store password hashes; they may get hacked.
  - ***Transmission*****:** passwords from user to server can be insecure (What kind of attacks?)
    - Common examples: phishing

Problems on **user** end

- Ideally, users should Use strong passwords and different passwords on different sites
  - Strong = hard-to-guess; Long != strong
- Password managers aim to make this easier, but it carries its won risks.
- How do human choose passwords?

Problems on user end

- "Social gaming" site RockYou hacked in December 2009
- Disclosed 32 million user passwords; posted to internet
- Passwords were in clear (not hashed or encrypted)
- Main source today of research / knowledge about user password composition

Rank	Password	Number of Users with Password (Absolute)
1	123456	290731
2	12345	79078
3	123456789	76790
4	Password	61958
5	iloveyou	51622
6	princess	35231
7	rockyou	22588
8	1234567	21726
9	12345678	20553
10	abc123	17542

Top 10 RockYou passwords

Problems on **server** end

- Anyone can try to guess the password
  - Weak pass come from a small dictionary.
  - Stolen passwords can help guessing.
- Server must store the passwords (in some form)
  - Servers get hacked on a regular basis
  - Hackers can steal password-related data
- Server must *see* the passwords when logging in.
  - That’s why phishing attacks can work

Major breaches in recent years

LinkedIn (2012, 2016)

- **Affected Users:** ~117 million (initial), ~700 million (later leak)
- **Cause:** Credentials stolen and sold on the dark web
- **Impact:** Massive credential stuffing attacks followed

Yahoo (2013, 2014)

- **Affected Users:** 3 billion (2013), 500 million (2014)
- **Cause:** State-sponsored hacking, stolen hashed passwords
- **Impact:** One of the largest breaches in history, affecting years of user data

Adobe (2013)

- **Affected Users:** ~153 million
- **Cause:** Poor password encryption, data leaked online
- **Impact:** Millions of weakly encrypted passwords exposed

Equifax (2017)

- **Affected Users:** 147 million
- **Cause:** Unpatched vulnerability in Apache Struts
- **Impact:** Stolen social security numbers, credit card details

Marriott/Starwood (2018)

- **Affected Users:** 500 million
- **Cause:** Unauthorized access via compromised credentials
- **Impact:** Sensitive guest data stolen, including passport numbers

Facebook (2019)

- **Affected Users:** 533 million
- **Cause:** Poor security practices, plaintext password storage
- **Impact:** Personal data leaked online

Twitter (2022)

- **Affected Users:** 5.4 million
- **Cause:** API vulnerability exploited by attackers
- **Impact:** Personal data, including emails and phone numbers, leaked

LastPass (2022)

- **Affected Users:** Undisclosed (potentially millions)
- **Cause:** Cloud storage breach exposing encrypted vault data
- **Impact:** Master password-based encryption put to the test

MOVEit (2023)

- **Affected Users:** Millions across multiple organizations
- **Cause:** Zero-day vulnerability exploited
- **Impact:** Ransomware group stole sensitive financial and personal data

';--have i been pwned?

Check if your email address is in a data breach

email address pwned?

Using Have I Been Pwned is subject to the terms of use

865	14,635,332,539	115,797	229,161,508
pwned websites	pwned accounts	pastes	paste accounts

Largest breaches

772,904,991	Collection #1 accounts
763,117,241	Verifications.io accounts
711,477,622	Onliner Spambot accounts
622,161,052	Data Enrichment Exposure From PDL Customer accounts
593,427,119	Exploit.In accounts

Recently added breaches

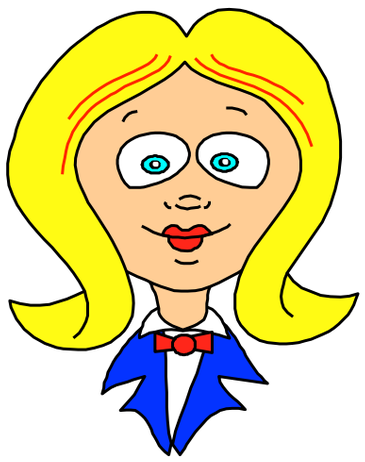
337,373	LandAirSea accounts
86,136	Adopt Me Trading Values accounts
937,912	Youthmanual accounts
3,123,439	Thermomix Recipe World Forum accounts
9,665	Hakko Corporation accounts

https://haveibeenpwned.com/

How should servers stored passwords to maximize security?

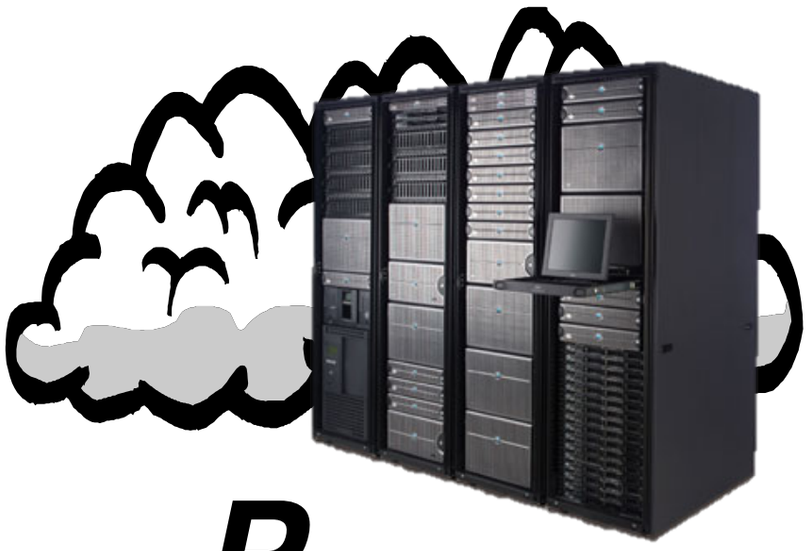
- Definitely not in plaintext
- Idea: store some **one-way representation of** passwords
  - How about the hashes?

Passwords are generally protected via hashing



Alice

P = “CatPajamas”

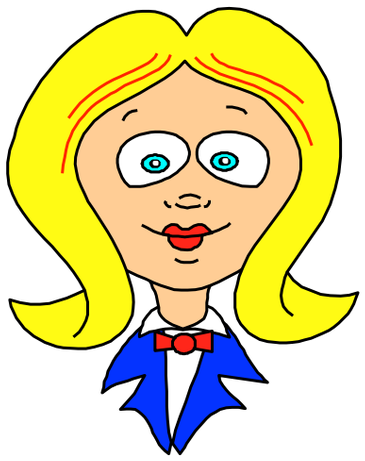


P

H

H(P)

To verify an incoming password…



Alice

P'

P'

H

H(P') ? = H(P)

Hashing alone is not enough

- **Pre-computation** attack
  - Attacker precomputes (pw, H(pw)) for popular passwords
  - Upon compromise, plaintext pw immediately recovered from H(pw) by looking up from pre-computed tables
  - Credential stuffing attack on other sites
- Personalized hash: per-user salt
  - Store (Alice, s, $H(p||s)$) for password p and salt s
  - Upon compromise, attack learns s, and can *eventually* recover pw
  - But this *slows down* attacks so users can change passwords on other sites.

What hash function to use?

- Ideal: fast when hashing a single password, but hard to scale up
- Not SHA-2: **too fast**
- The Open Worldwide Application Security Project (OWASP) recommends the following KDFs for password hashing
  - **Argon2id**
  - **scrypt** if Argon2id is unavailable
  - **bcrypt** for legacy systems
  - **PBKDF2** if FIPS-140 compliance is required
- They are purposefully slow on CPU/GPU and/or consume a lot of memory.

Nevertheless, server must see the passwords…

- Server must see the passwords *in plaintext* when users log in
- Risks
  - may be *eavesdropped* by trojan / rootkit
  - may be *accidentally* logged/cached

- Logging passwords? Who would do that??

TECH / TWITTER - X / MOBILE

Twitter advising all 330 million users to change passwords after bug exposed them in plain text / There's apparently no evidence of any breach or misuse, but you should change your password anyway

By Chaim Gartenberg

May 3, 2018, 4:21 PM EDT

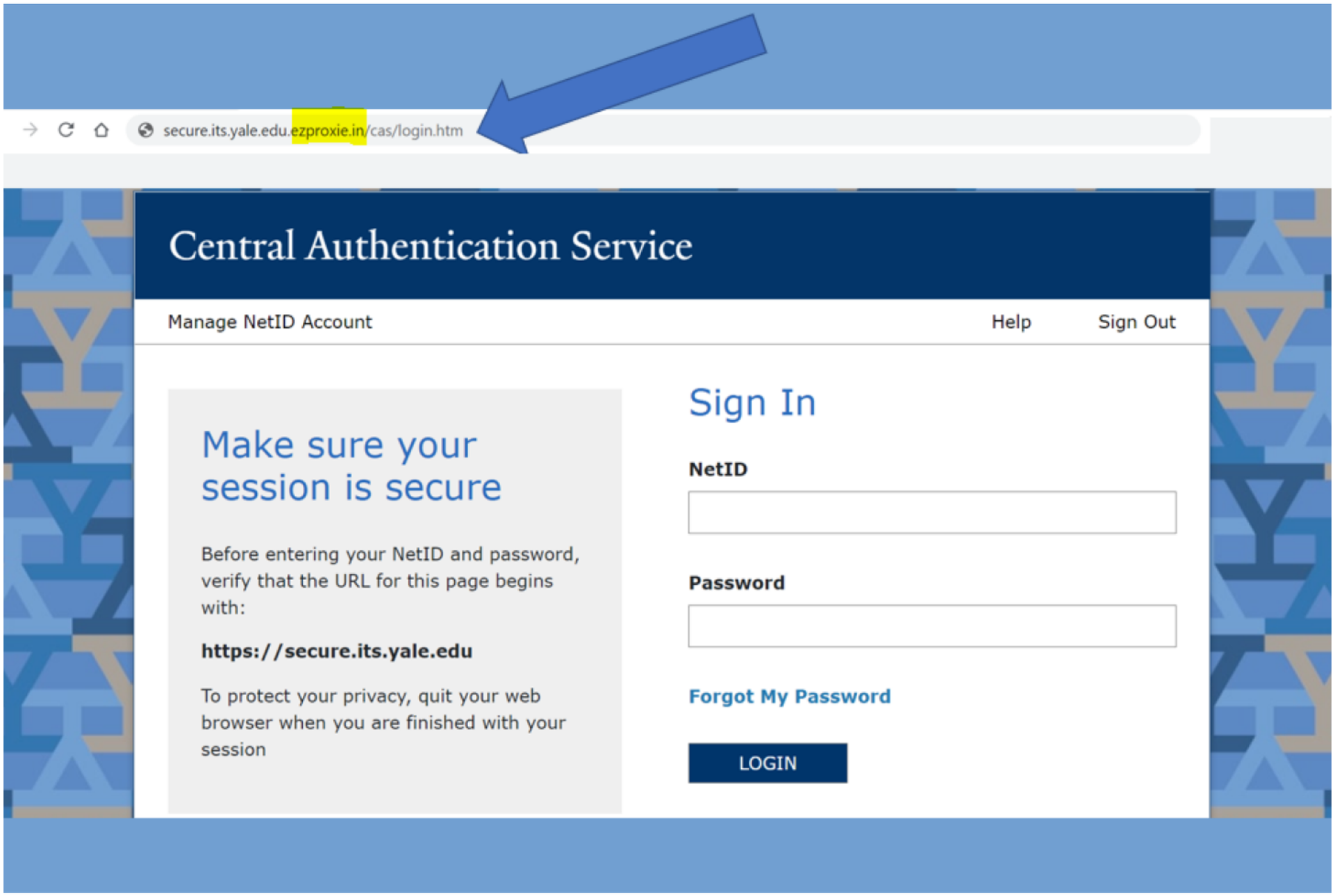
Security > Encryption

G Suite passwords stored in plain text for 14 years

News By Bobby Hellard published May 22, 2019

Google reveals a bug prevented its cryptography system encrypting enterprise users' login details since 2005

Another problem: Phishing



- If authentication doesn't reveal passwords to server, phishing won't be as problematic!

Limits of Password-based authentication

- Unavoidable attacks
  - guessing: mitigation via rate limiting, 2FA
  - dictionary attack upon server compromise
- Avoidable attacks
  - leaking passwords to the server when logging in
  - pre-computation attack upon server compromise
  - dictionary attack by network attackers

Password-over-TLS

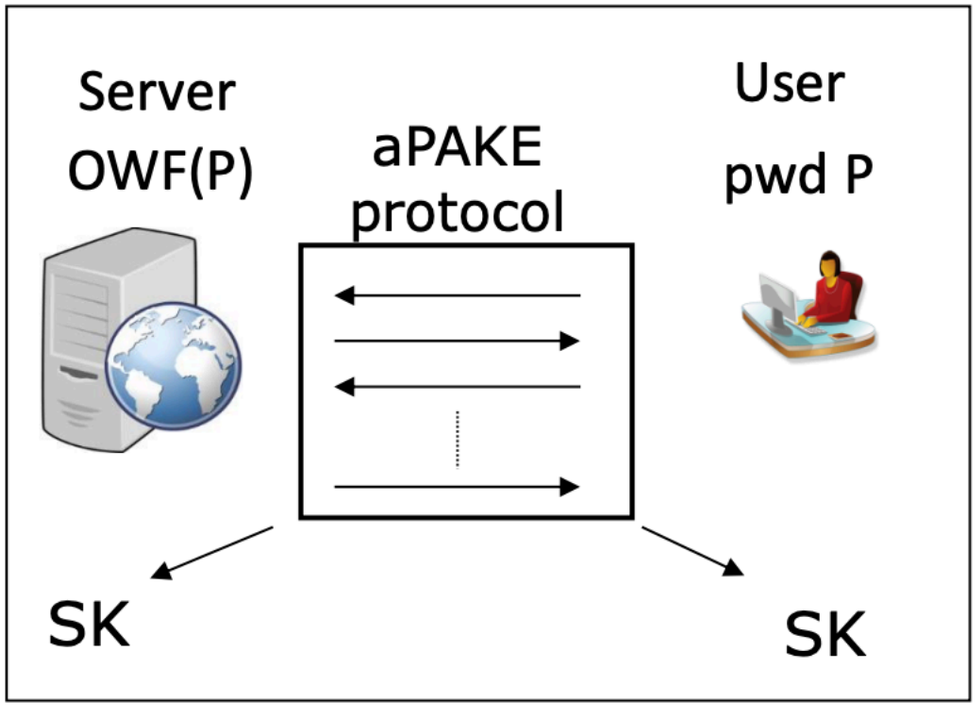
Some attempts

- Server stores (s, H(s, pw)); User sends H(s, pw) instead of pw.
- Challenge: how can user get salt from server?
- Option 1: Retrieve salt from server
  - Users are not yet authenticated
  - Allows the attacker to retrieve salt and mount pre-computation attack
- Option 2: Secure two-party computation
  - Inefficient
- We will see a **whole new solution**: strong aPAKE

Password-Authenticated Key Exchange (PAKE)

or “zkPassword”

Password-Authenticated Key Exchange (PAKE)



Don’t be get confused by KE, AKE, PAKE…

- Registration phase (**over a secure channel**)
  - Sever stores (username, password) securely
  - User stores her password
- Login phase (no PKI needed)
  - U and S output the same key if U has the correct password.
- Attacker: eavesdropping or actively interfering.
- Same goals as AKEs

(Recall) AKE: Basic Goals

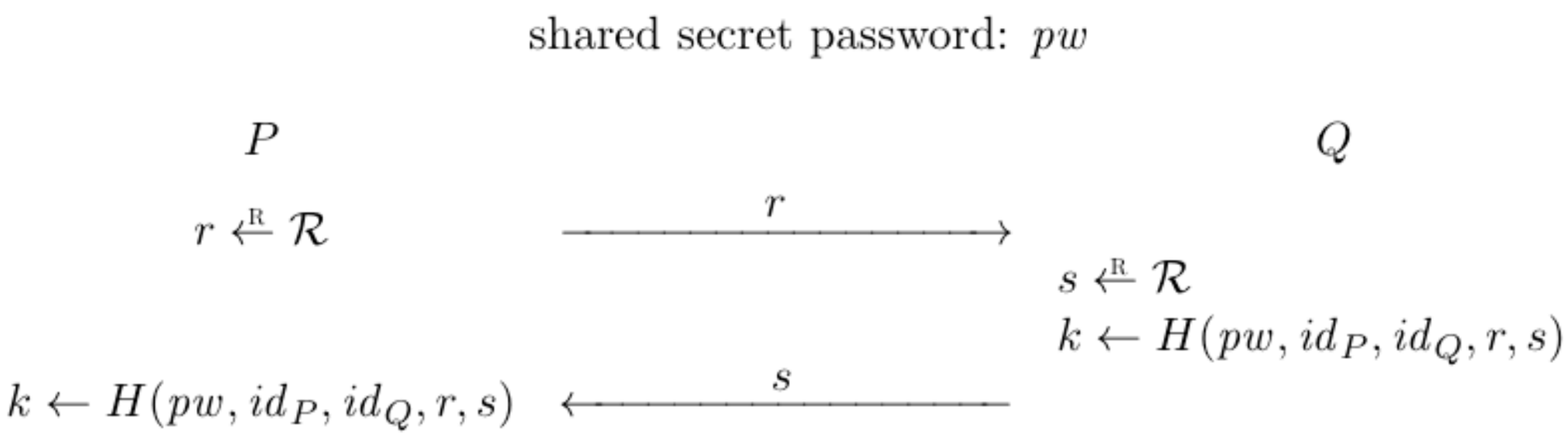
- **Authenticity**: If a party outputs $(K, \mathbf{id})$, then K is shared with only **id** — or nobody.
- **Secrecy**: adversary can't indistinguishable K from a random string by observing the communication
- e.g., ***Key re-use attack***: It breaks secrecy if the adversary can force re-use of an old key from a previous run.

PAKE0

- P = user, Q = server
- Assumption: During a registration phase, user and server share password **pw**. We will focus on login phase.

PAKE0

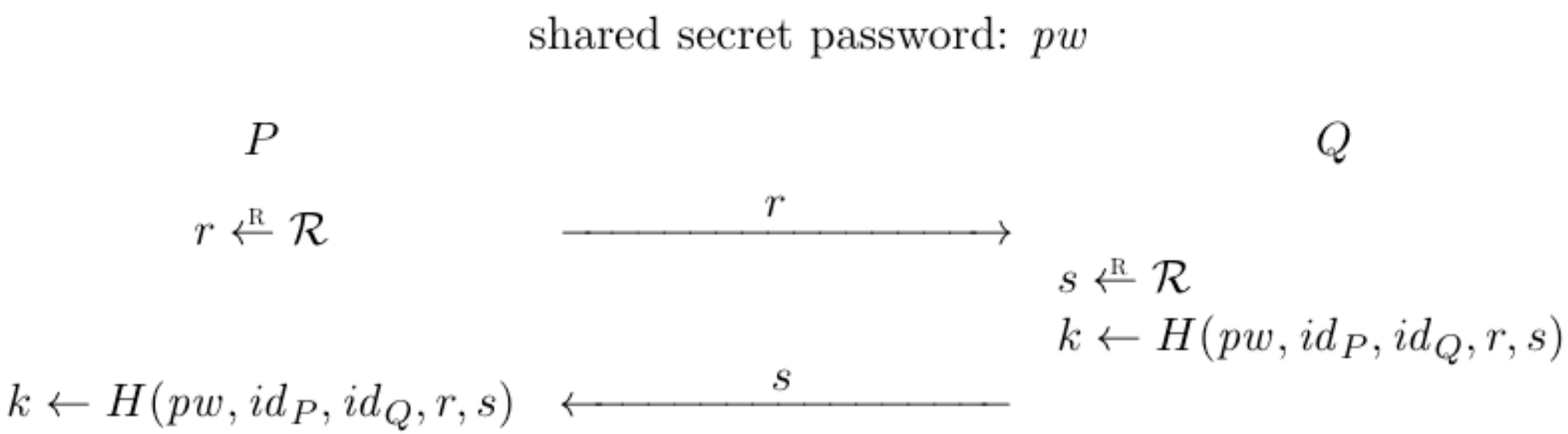
- P = user, Q = server
- user and server share password **pw**.



- **Security** assuming strong passwords
- **Authenticity**: if P establish a channel with someone, it’s with Q.

PAKE0

- P = user, Q = server
- User and server share password **pw**.



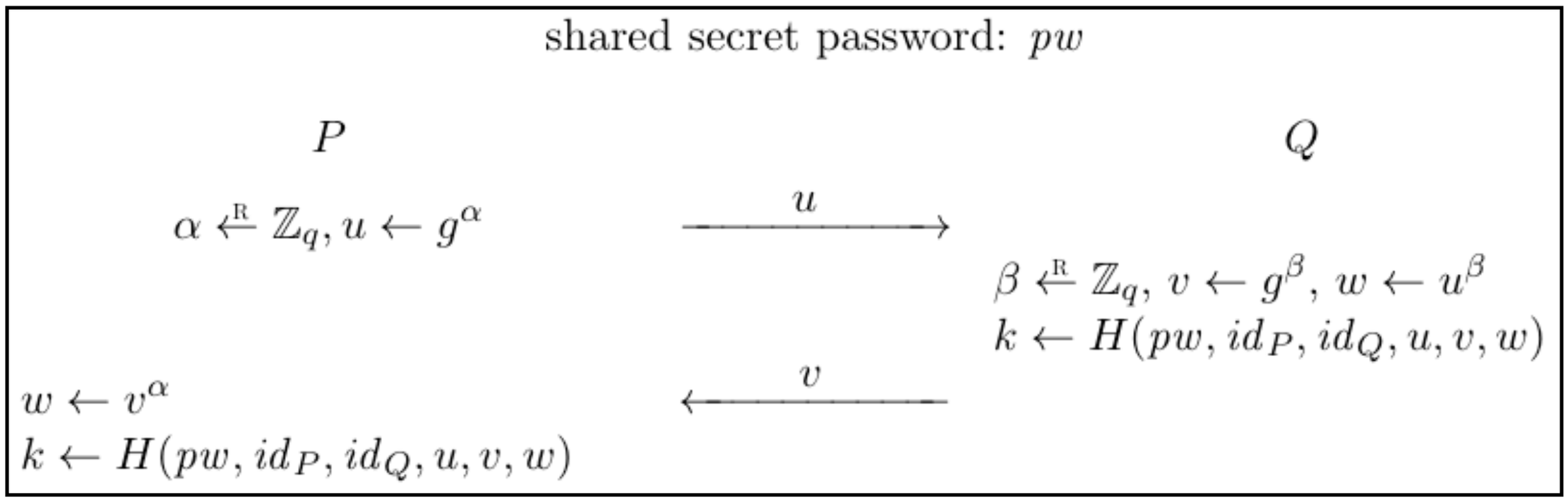
- Can **weak** passwords be brute-forced?
- Fine until a message is sent. Suppose P sends $c = \text{Enc}(k, m)$ for some publicly known m
- Offline dictionary attack possible: for pw in Dictionary, test if Dec($H(pw, id_P, id_Q, r, s)$, c) = m

PAKE1

- Same as assumption as before.

PAKE1

- Same as assumption as before.



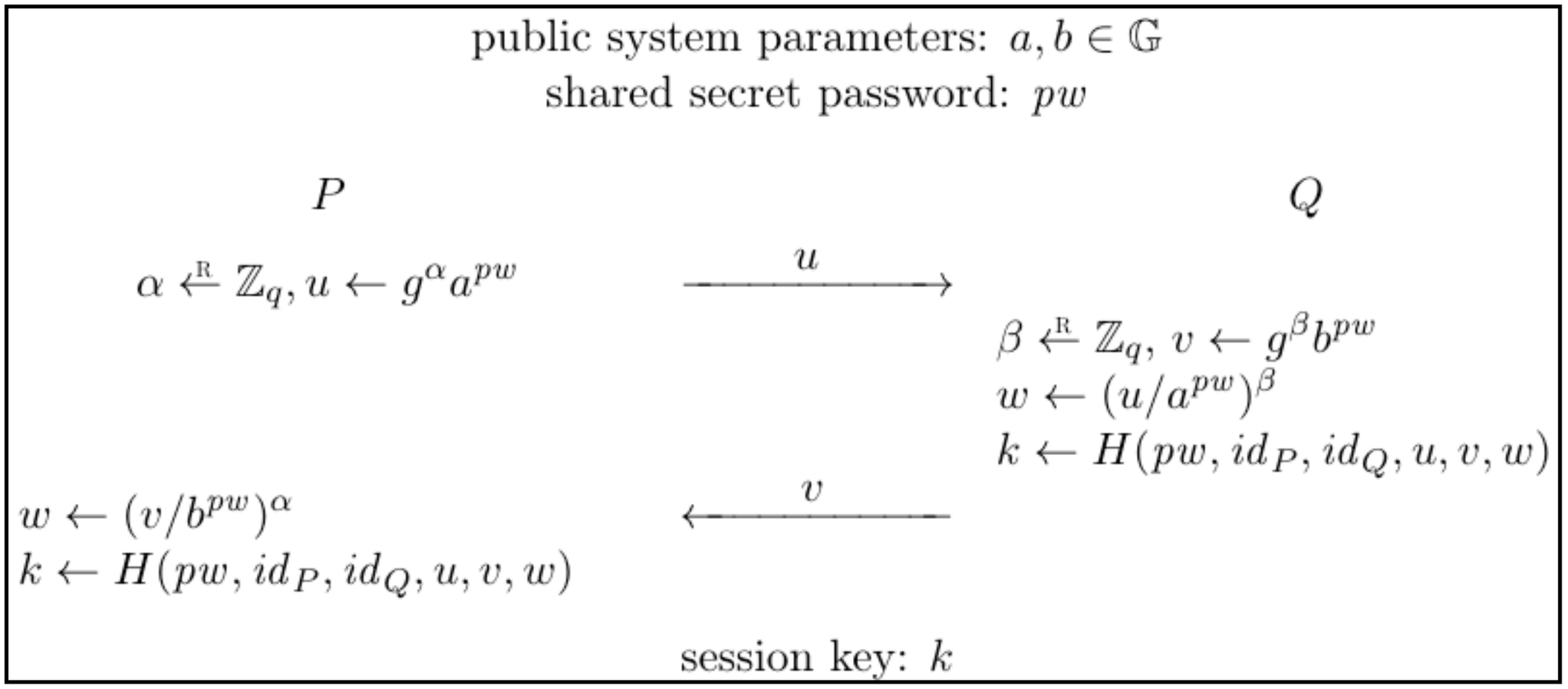
- For weak passwords, does previous attack work?
  - No. The attacker does know w (due to CDH).
- Dictionary attacks can be done by an *active* attacker.
  - Pretend to be Q, and wait for P to send a message like before.

PAKE2

- Idea: use password to “blind” u and v

PAKE2

- Idea: use password to "blind" u and v



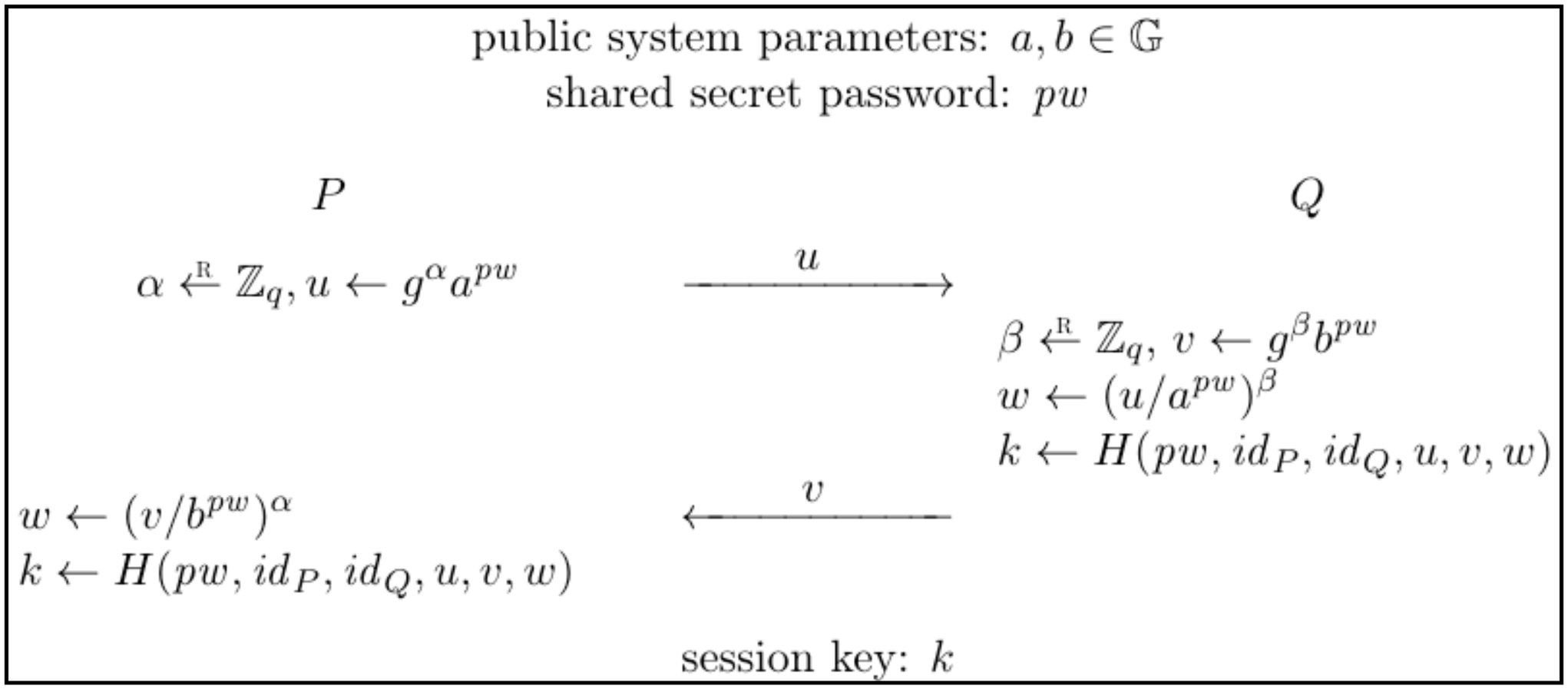
- An eavesdropper obtains (u, v)
- Dictionary attack involves trying different pw' in $H(pw', \mathsf{id}_P, \mathsf{id}_Q, u, v, \mathsf{DH}[u/a^{pw'}, v/b^{pw'}])$

PAKE2

- Main claim: for random $a, b, u, v \in \mathbb{G}$, it is hard to compute γ, w such that $w = [u/a^{\gamma}, v/b^{\gamma}]$
- Observe
  - $[xy, z] = [x, z][y, z]$
  - $[x^u, y^v] = [x, y]^{uv}$
  - Therefore, $[u/a^{\gamma}, v/\beta^{\gamma}] = [u, v][u, \beta^{-\gamma}][a^{\gamma}, v][a^{\gamma}, \beta^{-\gamma}]$
- Proof: if an attacker can compute γ, w as above, she can compute $[u, v]$, which breaks CDH.

PAKE2

- Idea: use password to “blind” u and v



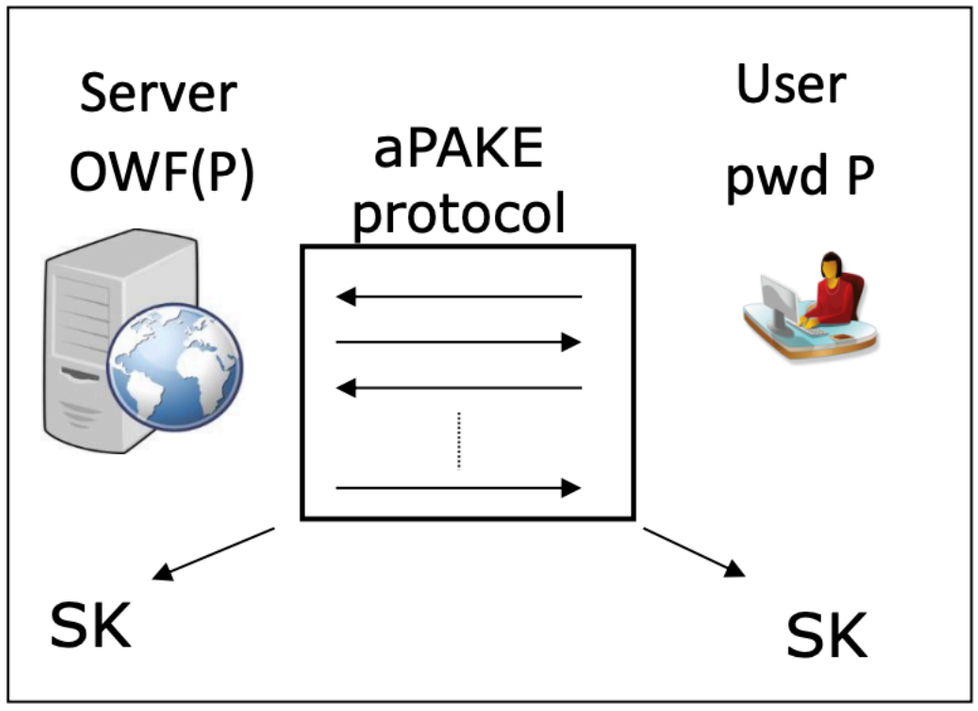
- Pros: password not revealed during log-in
- Cons: server must store passwords in plaintext.
  Can’t use salt to prevent pre-computation attacks

Limits of Password-based authentication

- Unavoidable attacks
  - guessing: mitigation via rate limiting, 2FA
  - dictionary attack upon server compromise
- Avoidable attacks

	Password-over-TLS	PAKE
leaking passwords to the server when logging in	✗	✓
pre-computation attack upon server compromise	✓	✗
dictionary attack by network attackers	✓	✓

Strong aPAKE



- What we saw are **aPAKE** (Asymmetric PAKE): server never sees password in plaintext.
- Strong aPAKE: pre-computation attacks are prevented.