Lecture 15:
Authentication
(Pt 2: Federated Auth)

CPSC 4440/5440, 2026 Spring

Instructor: Fan Zhang

Yale

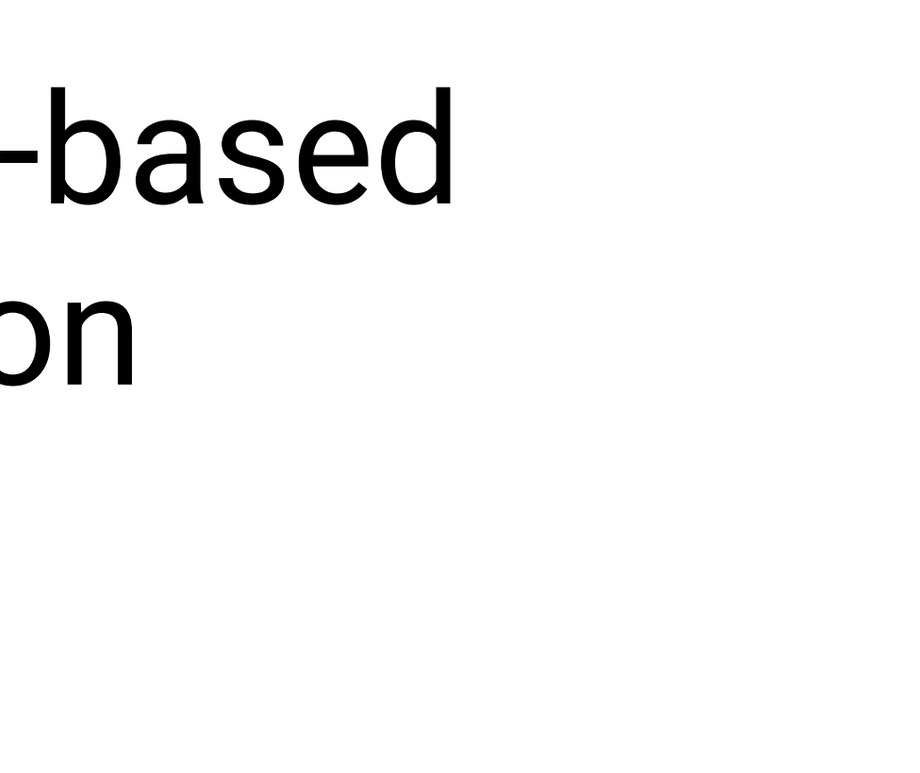
Computer Science

Updates

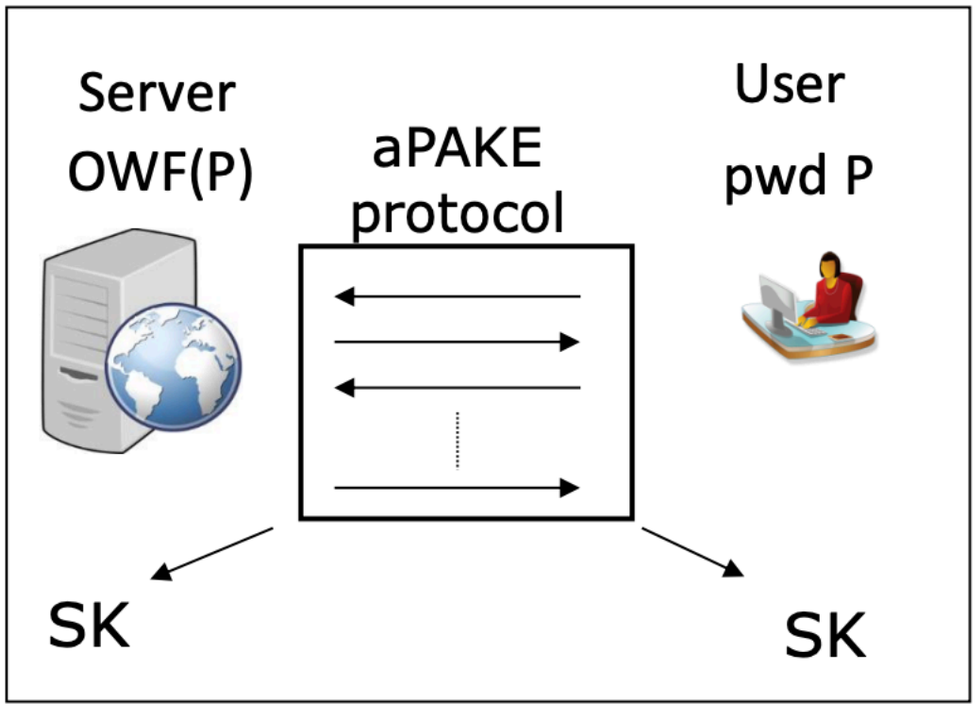
- HW4 deadline pushed after the break!
- March 25th, second presentation session by Ben Raykhman and Amin. Make your selection by Friday:
  - Unlinkable Inference as a User Privacy Architecture (blog post)
  - zkLogin: Privacy-Preserving Blockchain Authentication with Existing Credentials (CCS'24)
  - Securing Zoom Groups against Malicious Servers without New Security Elements (IEEE S&P'25)
  - Encrypted Access Logging for Online Accounts: Device Attributions without Device Tracking (USENIX Sec'25)
  - Search title to find PDFs or see links on Canvas or https://www.fanzhang.me/teaching/444/

Recall: Password-based Authentication

- *Unavoidable* attacks (w/o additional assumptions)
  - 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-Authenticated Key Exchange (PAKE)



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

- After registration, user needs not reveal password again
- Does not rely on PKI or certs
- Key shortcoming of PAKE2: **pre-computation attack.**

OPAQUE: An Asymmetric PAKE Protocol Secure Against Pre-Computation Attacks

Stanislaw Jarecki[1], Hugo Krawczyk[2], and Jiayu Xu[1]

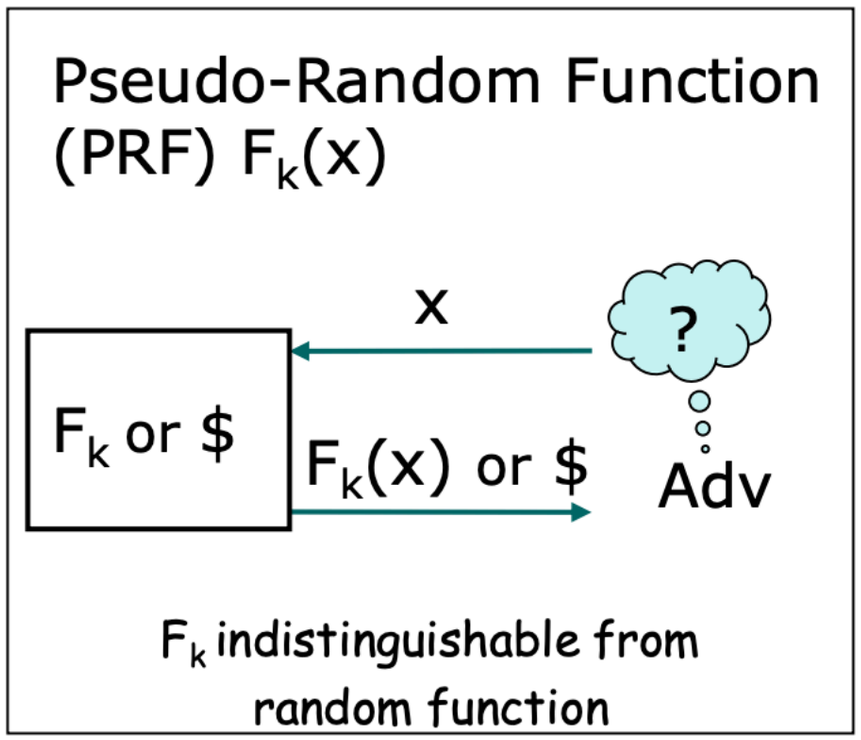
[1] University of California, Irvine. Email: {stasio@ics.,jiayux@}uci.edu.
[2] IBM Research. Email: hugo@ee.technion.ac.il.

Abstract. Password-Authenticated Key Exchange (PAKE) protocols allow two parties that only share a password to establish a shared key in a way that is immune to offline attacks. *Asymmetric* PAKE (aPAKE) strengthens this notion for the more common client-server setting where the server stores a mapping of the password and security is required even upon server compromise, that is, the only allowed attack in this case is an (inevitable) offline exhaustive dictionary attack against individual user passwords. Unfortunately, most suggested aPAKE protocols (that dispense with the use of servers' public keys) allow for *pre-computation attacks* that lead to the *instantaneous compromise* of user passwords upon server compromise, thus forgoing much of the intended aPAKE security. Indeed, these protocols use – in essential ways – deterministic password mappings or use random "salt" transmitted *in the clear* from servers to users, and thus are vulnerable to pre-computation attacks.

OPAQUE (Eurocrypt'18) is the first Strong aPAKE.

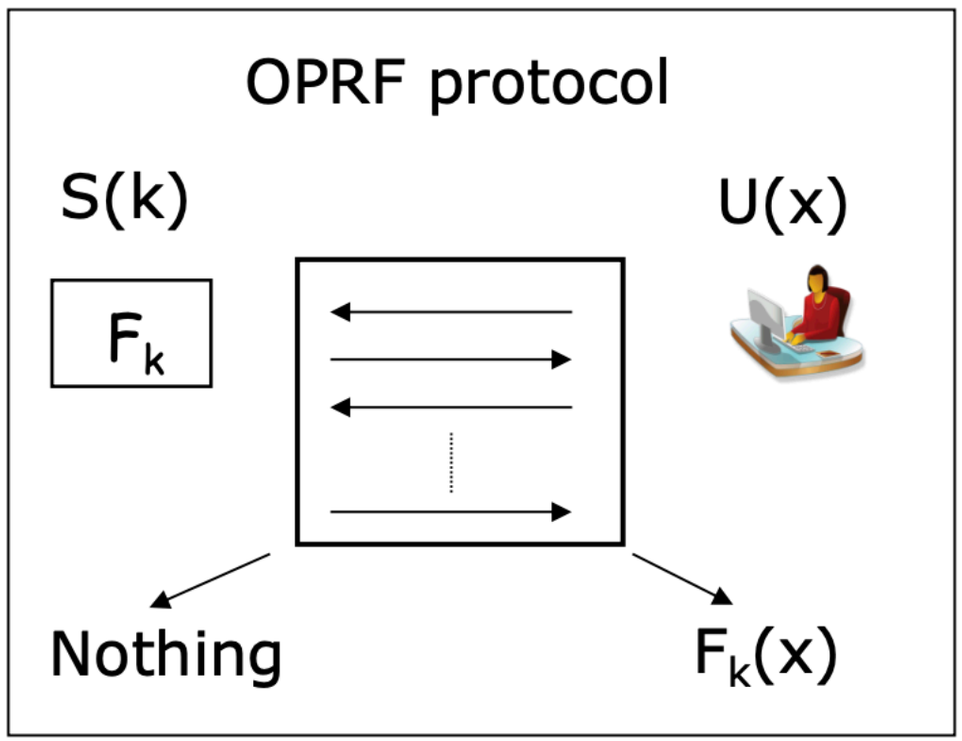
Oblivious PRF (OPRF)

Recall: PRF



- A PseudoRandom Function (PRF) is a family of function $\{F_1, F_2, \cdots\}$ such that for a uniform $k \in \{0,1\}^n$, $F_K(x)$ is indistinguishable from a random func with the same domains/co-domains.
- Examples
  - HMAC(K, .) is a PRF from {0,1}* to $\{0,1\}^{256}$
  - SHA-256(K, .) is not a PRF. Why?
  - If $H(\cdot)$ maps to a random element in a group $\mathbb{G}$ where DDH is hard, $H(x)^k$ is PRF from {0,1}* to $\mathbb{G}$

OPRF



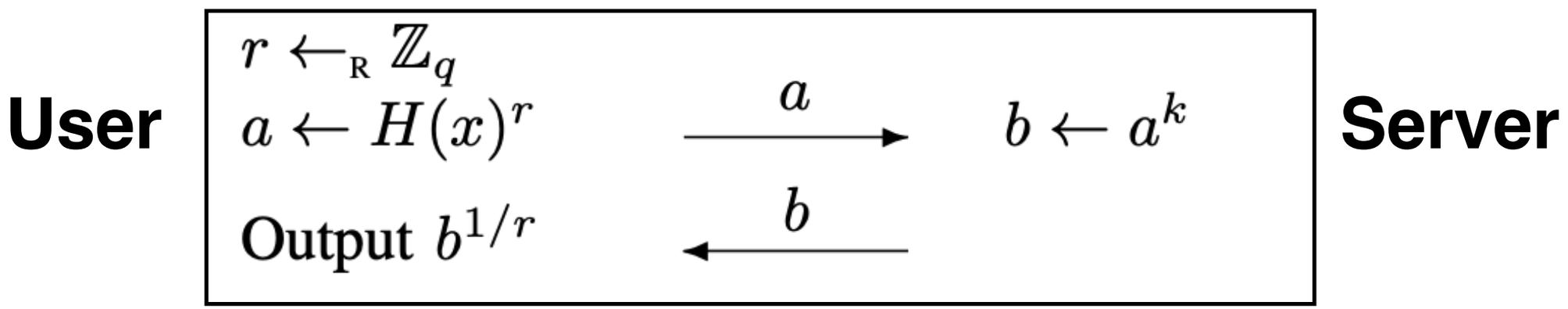
- Oblivious PRF: a two-party *interactive* protocol to compute PRF
- Sever inputs k, user inputs x
- At the end of the protocol:
  - user learns $F_k(x)$
  - server learns nothing (“oblivious” to the server)

OPRF has many applications

- Oblivious keyword search (KS)
- private information retrieval (PIR)
- secure pattern matching
- private set intersection (PSI)
- password-authenticated key exchange (PAKE)
- password-protected secret sharing (PPSS/TPASS)
- “untraceable" contact tracing

Example: HashDH

- How to convert $\mathrm{H}(x)^k$ to an OPRF?
  - Server holds k, user inputs x
- User can't send over $\mathrm{H}(x)$
  - If so, server learns $\mathrm{H}(x)^k$ (server shouldn't learn anything)
- Idea: blinding



Many OPRFs are known

PRF		Batching	Multi-point	Partially oblivious	Verifiable	Committed	Proactive sec.	Updatable	Distributed	Threshold	Special (Sect. 3.8)
FIPR05, 1st [4]	NR	○	●	○	○	○	○	○	○	○	○
HL08 [5]	NR	○	●	○	●	○	○	○	○	○	○
JKK14 [9]	NR	○	●	○	●	○	○	○	○	○	○
Hazay15 [26]	NR variant	○	●†	○	●	○	○	○	○	○	○
JL09 [7]	DY	○	●	○	●	●^{i}	○	○	○	○	○
CL17 [22]	DY	○	●†	○	○	●^{o}	○	○	○	○	○
MPRSY20 [30]	DY	●	●	○	●	○	○	○	●	○	○
TCRSTW21 [31]	DY variant	○	●	●	●	○	○	○	○	○	○
JL10 [8]	2HashDH	●†	●	○	○	○	○	○	○	○	○
JKK14 [9]	2HashDH/RSA	○	●	○	●	○	○	○	○	○	○
ECSJR15 [19]	HashDH	○	●	●	●	○	○	●	◉	●	○
JKKX16 [10]	2HashDH	○	●	○	○	○	○	○	○	○	○
JKKX17 [11]	2HashDH	○	●	○	○	○	○	○	◉	●	○
DGSTV18 [28]	2HashDH	●†	●†	○	●	○	○	○	○	○	○
Lehmann19 [57]	HashDH	●♣	●	◉	○	○	○	◉	○	○	●
BFHLY19 [12]	2HashDH	○	●	●	○	○	●	○	●	○	○
DHL22 [13]	2HashDH	○	●	●	○	○	○	○	●	○	●
KKRT16 [6]	Any	●✶	○	○	○	○	○	○	○	○	○
KMPRT17 [34]	Any	○	○	○	○	○	○	○	○	○	●
JKR18 [58]	Any	○	●◇	●	●	○	○	●	◉	●	○

TABLE 3: Properties of OPRFs from the literature.

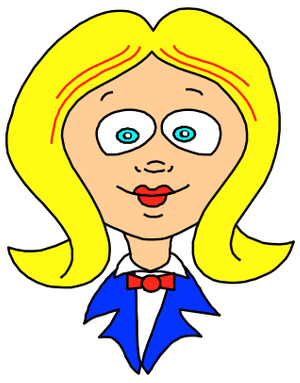
● = applicable
◉ = trivially implied
○ = not satisfied

✶ = with related keys
† = enforcing same key
◇ = input-dependent key

♣ = for same input
sh = semi-honest server
i = input, o = output

- We saw HashDH

OPAQUE v1: registration



pw, k

$rw = F_k(pw)$

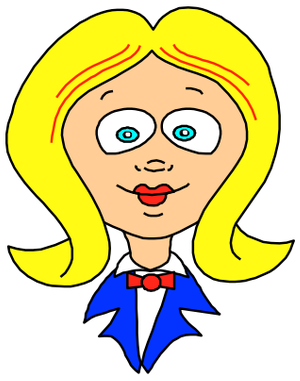
OPRF

k

Server stores (Alice, k, rw)

- rw can be viewed as a **upgraded** password (high entropy)
- Yet users only need to remember the (low entropy) password.

OPAQUE v1: login



pw

$rw = F_k(pw)$

OPRF

k

Server stores (Alice, k, rw)

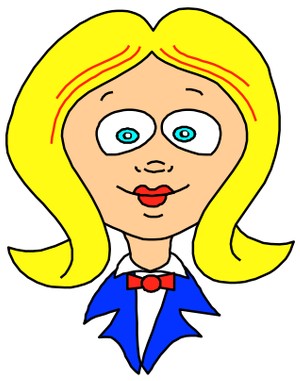
rw

aPAKE
(e.g., PAKE2)

rw

- **Defeats pre-computation attacks:** Recovering pw from rw can only be done after compromising the server & learning k

OPAQUE v2: registration



pw

$rw = F_k(pw)$

OPRF

k

Sample (sk_S, pk_S)

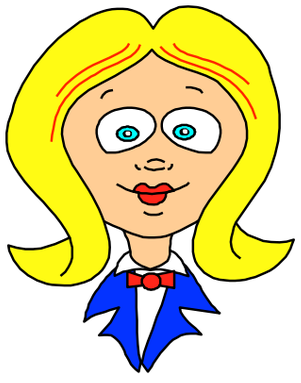
pk_S

Sample (sk_U, pk_U)

$C = \mathsf{Enc}(rw, (\mathsf{sk}_U, \mathsf{pk}_S))$

Store (**Alice**, C, sk_S, pk_U)

OPAQUE v2: login



pw

$rw = F_k(pw)$

OPRF

k

Store (**Alice**, C, sk_S, pk_U)

Obtain (**Alice**, C)

decrypt using rw to get sk_U, pk_S

Only Alice can proceed

sk_U, pk_S

AKE

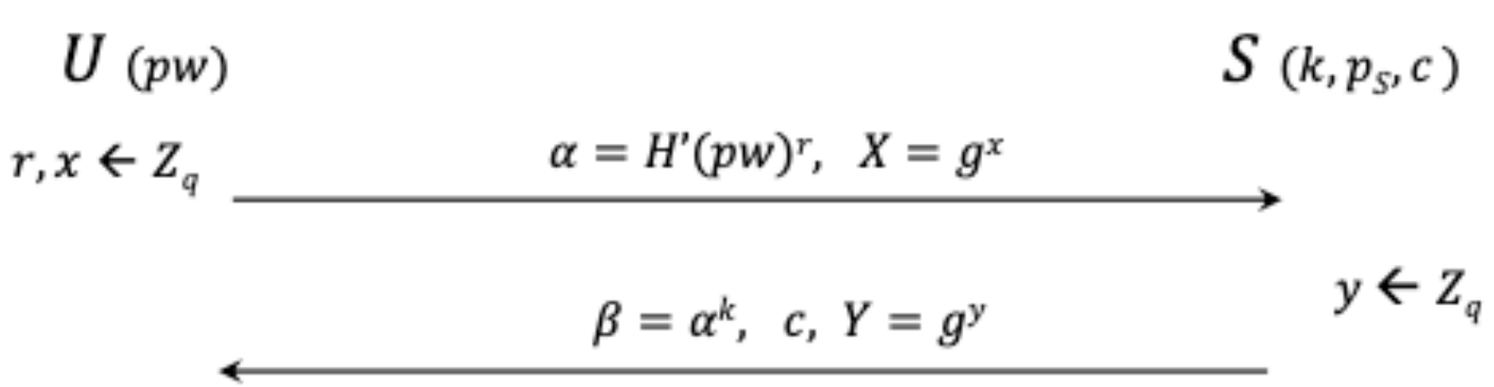
sk_S, pk_U

The OPAQUE protocol

S. Jarecki, H. Krawczyk, and J. Xu, “OPAQUE: An Asymmetric PAKE Protocol Secure Against Pre-computation Attacks,” in Advances in Cryptology – EUROCRYPT 2018

Init: On input pw, p_U by U and k, PS by S, U computes $rw = H(pw, H'(pw)^k)$ and $c = AuthEnc_{rw}(p_U, P_U, P_S)$. S stores (k, p_S, c). U only keeps pw.

Login:



- $rw \leftarrow H(pw, \beta^{1/r})$
- $p_U, PK_U, PK_S \leftarrow AuthDec_{rw}(c)$
- $K = KE(p_U, x, P_S, Y)$ $\quad\quad$ $K = KE(p_S, y, P_U, X)$

Stream: Internet Research Task Force (IRTF)
RFC: 9807
Category: Informational
Published: July 2025
ISSN: 2070-1721
Authors: D. Bourdrez, *AWS*; H. Krawczyk, *Meta*; K. Lewi, *Cloudflare, Inc.*; C. A. Wood

RFC 9807
The OPAQUE Augmented Password-Authenticated Key Exchange (aPAKE) Protocol

Applications of OPAQUE

- Replacing passwords authentication!
  - Ongoing standardization by IETF
- [Quiz] what are the benefits passwords over TLS?
  - never reveals the password to server after set up
  - does not rely on PKI
- Retrieval mode: convert passwords to strong keys (i.e., only doing the OPRF part), converting passwords to keys
  - WhatsApp uses OPAQUE for end-to-end encrypted backups.
  - Password managers
  - cryptocurrency wallets
  - **Potential danger: relies on server availability**

Threshold OPRF (t out of n)

- E.g., 3 out of 5 servers are required
  - 3 or more servers can compute Fk(x)
  - Breaching 2 or less won’t leak k
- Tolerate corruption and server unavailability.

Secret sharing

- $(s_1, \cdots, s_n) \leftarrow^{\$} \text{Share}(s, t, n)$
  - Split secret s into n shares, with a recover threshold t
  - The process of computing and distributing s_i is called **dealing**
- $s = \text{Reconstruct}(\{s_i\}_{i \in T})$
  - T is any subset of at least t shares
- This is the most basic form of secret sharing protocols; other flavors include:
  - Verifiable SS (VSS)
  - Publicly Verifiable SS (PVSS)
  - …

Shamir secret sharing ('79)

- Fact: a m-degree polynomial $p(x) = c_0 + c_1 x + c_2 x^2 \cdots + c_t x^m$ is uniquely determined from any $m + 1$ evaluations
- $(s_1, \cdots, s_n) \leftarrow^{\$} \text{Share}(s, t, n)$
  - Sample random $(t-1)$-degree polynomial such that $p(0) = s$
  - $s_i = p(i)$ // to use FFT, evaluate on roots of unity
- $s = \text{Reconstruct}(\{s_i\}_{i \in T})$
  - recover $p(0)$ using Lagrange interpolation from (i, s_i)

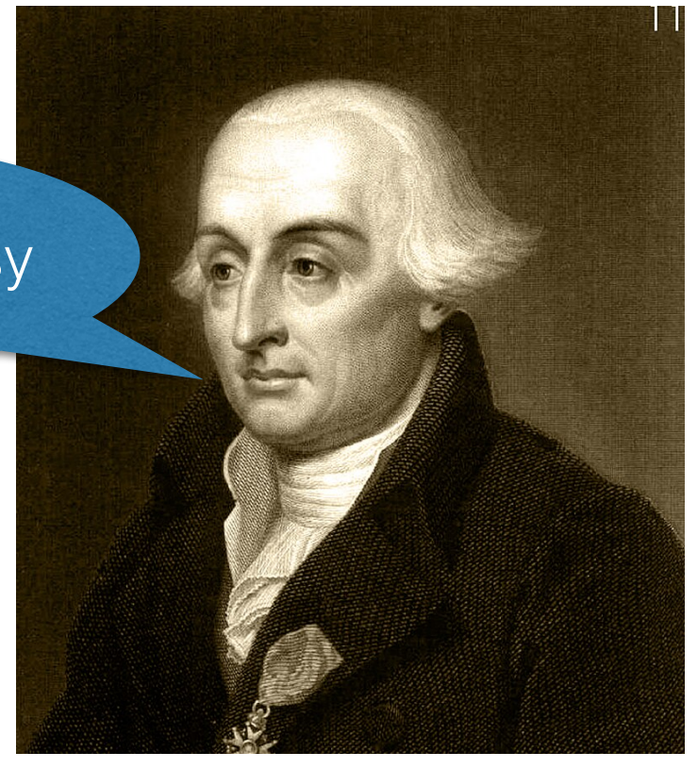
(O(t log t) using FFT)

Lagrange interpolation

- Recover a (t-1)-deg poly $p(x)$ from t points $\{(x_i, y_i)\}_{i=1}^{t}$ where $y_i = p(x_i)$

1. $\ell_j(x) = \prod_{m \neq j} \frac{x - x_m}{x_j - x_m}$

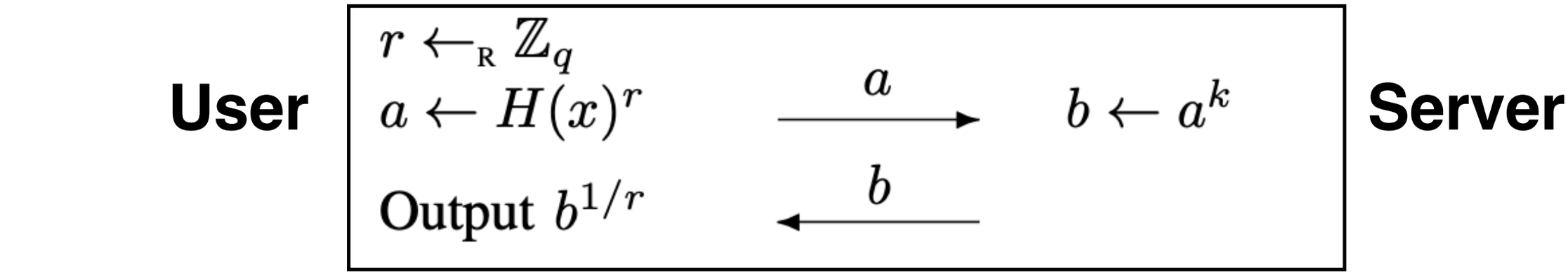
2. $p(x) = \sum_{j=1}^{t} y_j \ell_j(x)$



Joseph-Louis Lagrange (1736-1813)

- In short: $p(0) = \sum_{j=1}^{t} y_j \lambda_j$ // $\lambda_j = \ell_j(0)$

$p(0)$ is a linear combination of $\{y_i\}$.



- Setup phase: Secret-share k cross n servers using (t, n)-Shamir secret sharing
- User: sends a to all servers
- Server i: sends back $b_i = a^{k_i}$
  - Assuming servers are honest…
- User: waits for t response, computes $b = \prod_i b_i^{\lambda_i} = a^{\sum \lambda_i k_i} = a^k$
- Client removes blinding as before: $b^{1/r}$
- That’s a threshold OPRF (TOPRF)

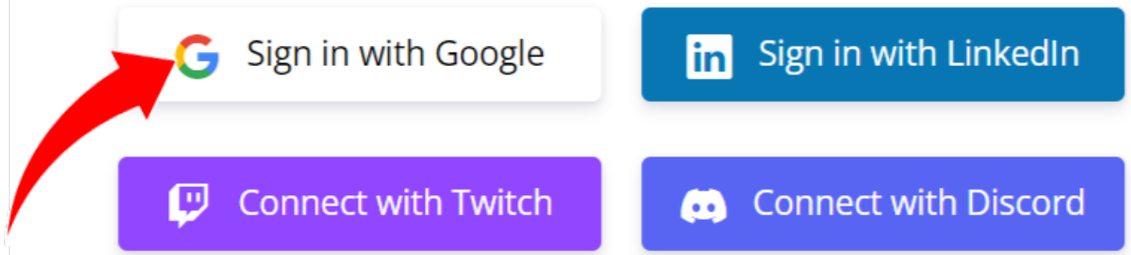
Other OPRF flavors

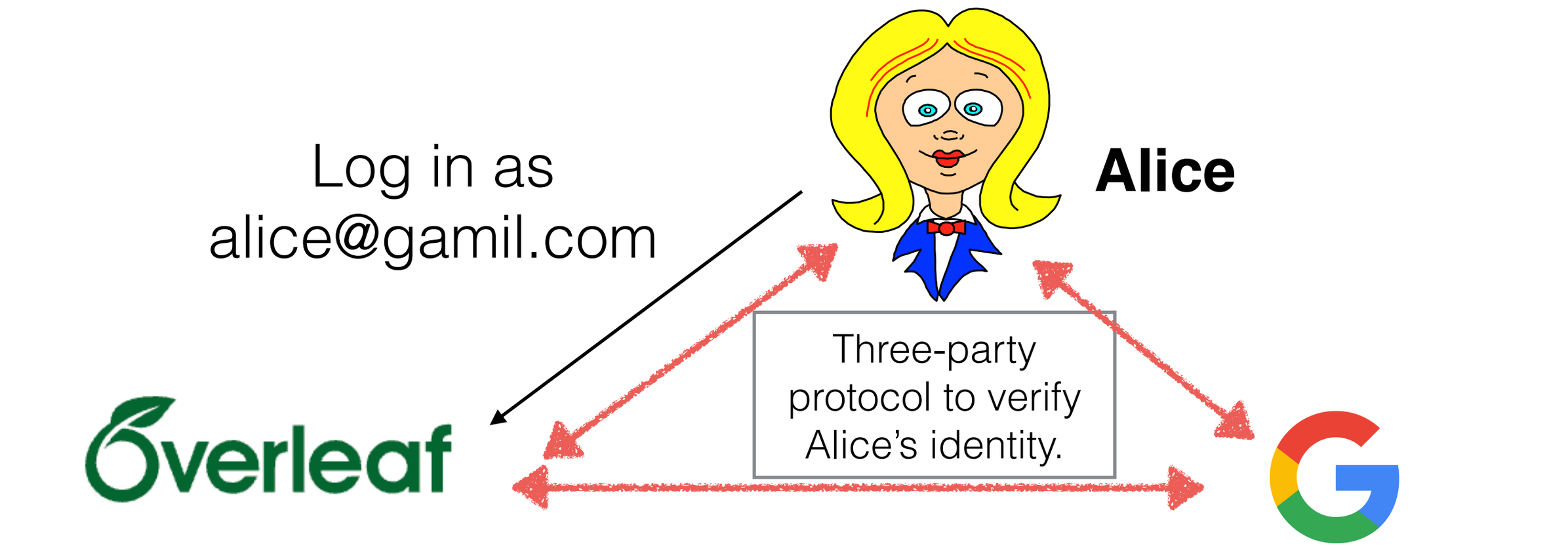
- Verifiable OPRF
- Partially oblivious PRF
  - Part of user input is learned by server
- Batched OPRF
- Updatable OPRF
- …
- Talk: *Fantastic OPRFs and where to find them* (https://ctrlc.hu/~stef/fantastic-oprfs.pdf)


INTERMISSION

Federated Authentication

Single-sign on (SSO)





- **Benefits (for user, for RP, for IdP)?**
- Avoid creating *more* passwords on user side
- Google does a good job balancing recoverability v.s. security
- Avoid attack surfaces: compromising RP does not leak passwords
- Any benefits for IdP?

Top 10 Breakthrough in 2022

MIT Technology Review

COMPUTING

The end of passwords

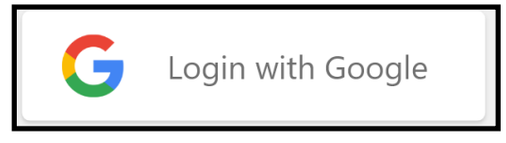
Companies are finally shifting away from notoriously insecure alphanumerics to other methods of authentication.

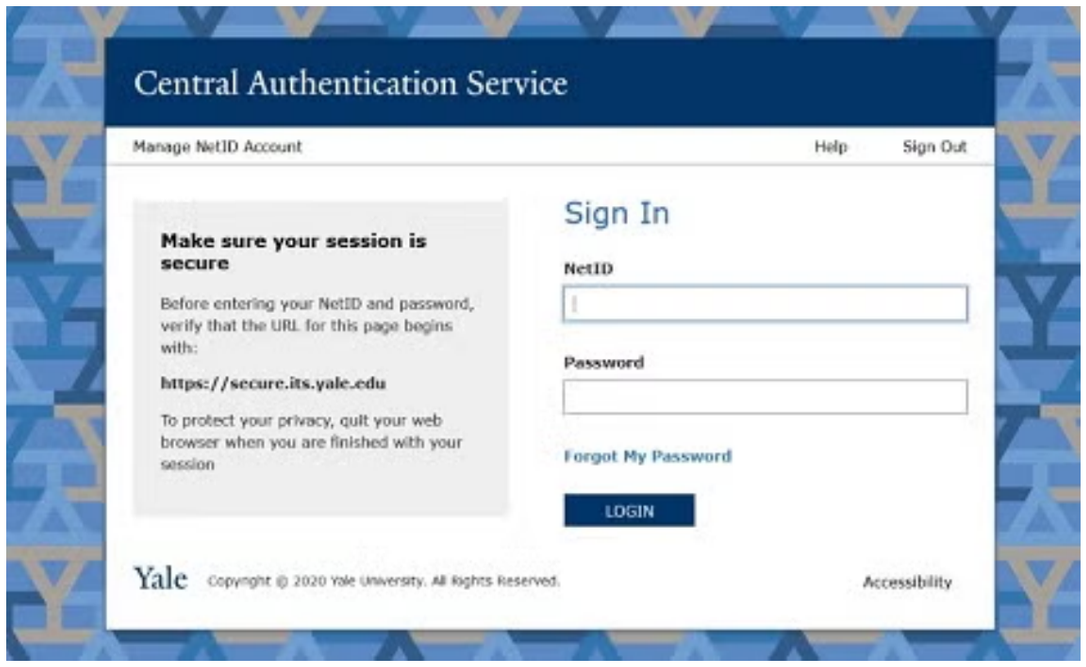
By Mat Honan

February 23, 2022

Single-sign on (SSO)

- Common protocols
  - OpenID Connect (OCID)
  - SAML (Security Assertion Markup Language) — enterprise
  - Kerberos — enterprise
  - Central Authentication Service (CAS)

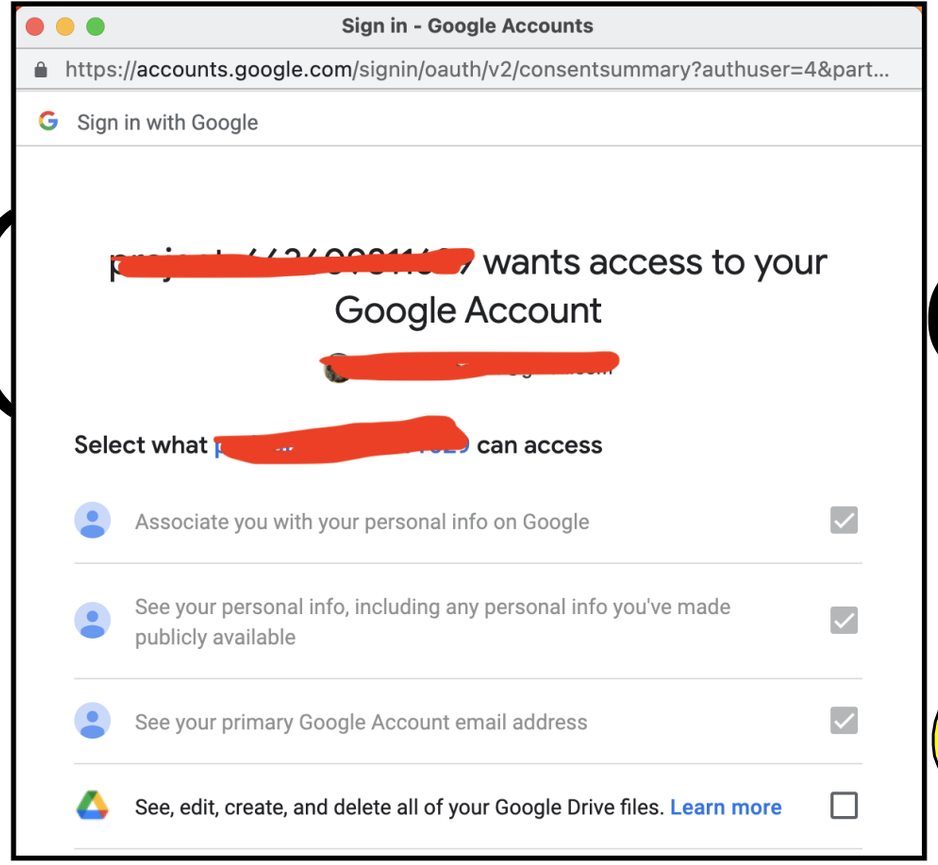




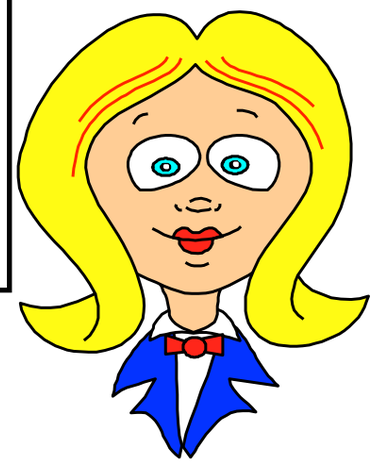
OAuth

- OAuth is an *authorization* framework
- **Authorization** (goal: manage who can access what resources)
- ***Authentication*** (goal: verify identity)
- OAuth can be used for authentication:
  - Alice allows RPs to access her identity information (e.g., gmail address)

OAuth (



evel)



User

Signed Token

Signed Token

1) Username, Password
2) Request from Calendar

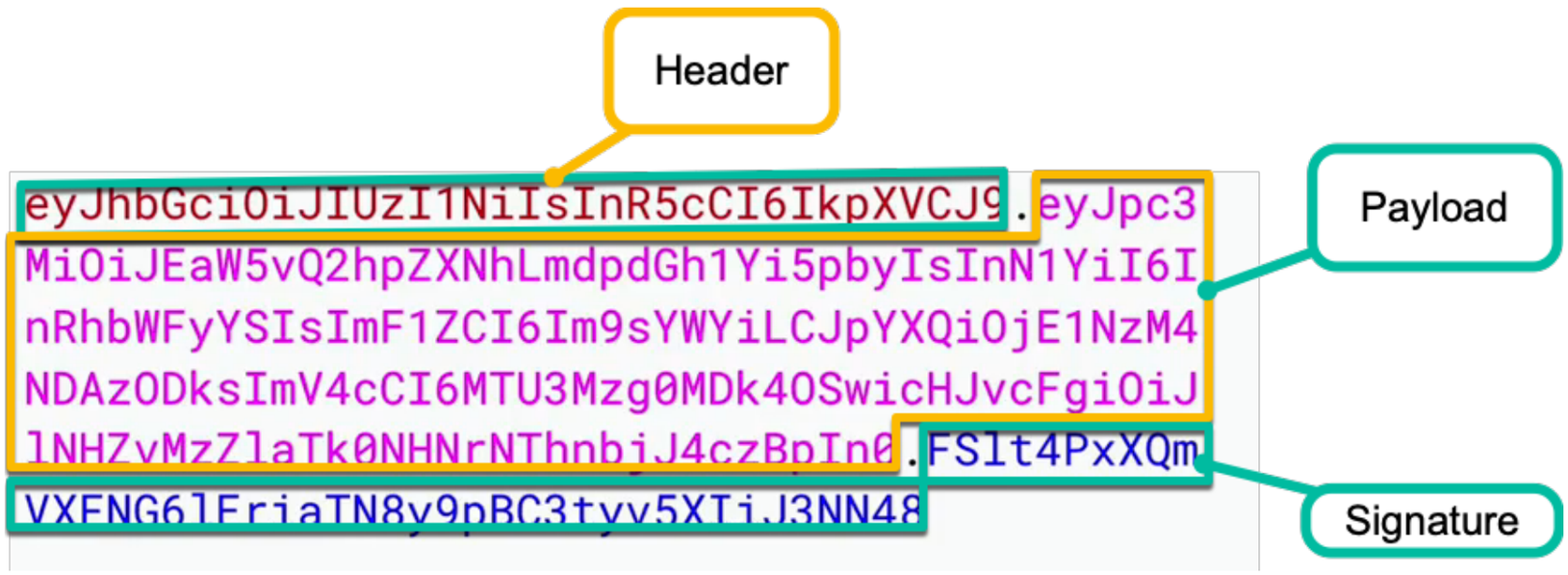
Read User's calendar

Signed Token

Client needing access to User's Google Calendar

Authorization server

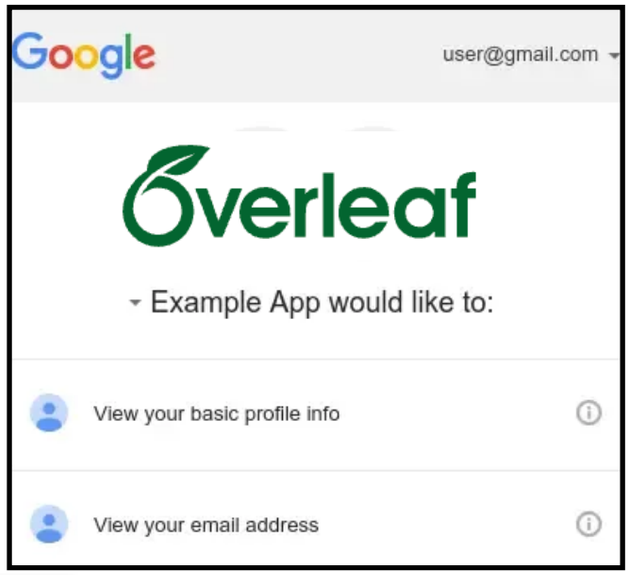
JSON Web Tokens

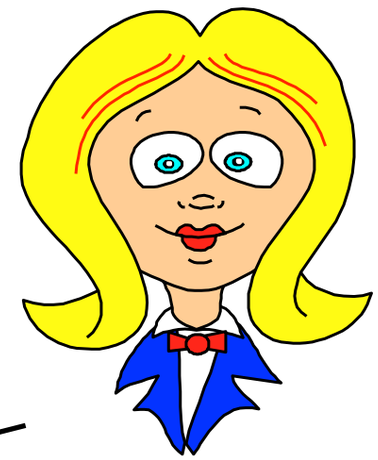


Issuer: IdP
Audience: RP
Subject: user/owner
Scope: what's allowed

```
{
      "alg": "RS256",
      "typ": "JWT"
    }
    .
    {
      "iss": "https://example.auth0.com/",
      "aud": "https://api.example.com/calendar/v1/",
      "sub": "usr_123",
      "scope": "read write",
      "iat": 1458785796,
      "exp": 1458872196
    }
```

“Log in with Google”
“OpenID Connect (OIDC) over OAuth”





Alice

Signed Token

Let user access the account with alice@gmail.com

Signed Token

1) Username, Password

Read email address

Signed Token

Overleaf

Relying Party (RP)

alice@gmail.com

Identity Provider (IdP)

Security problems

- **Single-point failure**: If IdP is compromised, attacker gains access to all third party apps.
  - Similar, if user's main account is compromised…
- **Tracking**: Google (IdP) learns when and what apps you login (tracking)
- **Linkability**: Apps (RPs) learns your gmail address (IdP identifier)

Hiding emails?

- This is **hard** b/c many apps require email address for other reasons
- One solution: one-time email
  - “Login with Apple” offers email relay

Addressing Single-point failures

PASTA: PASsword-based Threshold Authentication

Shashank Agrawal
Visa Research
shaagraw@visa.com

Peihan Miao*
University of California, Berkeley
peihan@berkeley.edu

Payman Mohassel
Visa Research
pmohasse@visa.com

Pratyay Mukherjee
Visa Research
pratmukh@visa.com

ABSTRACT

Token-based authentication is commonly used to enable a single-sign-on experience on the web, in mobile applications and on enterprise networks using a wide range of open standards and network authentication protocols: clients sign on to an identity provider using their username/password to obtain a cryptographic token generated with a master secret key, and store the token for future accesses to various services and applications. The authentication server(s) are single point of failures that if breached, enable attackers to forge arbitrary tokens or mount offline dictionary attacks to

KEYWORDS

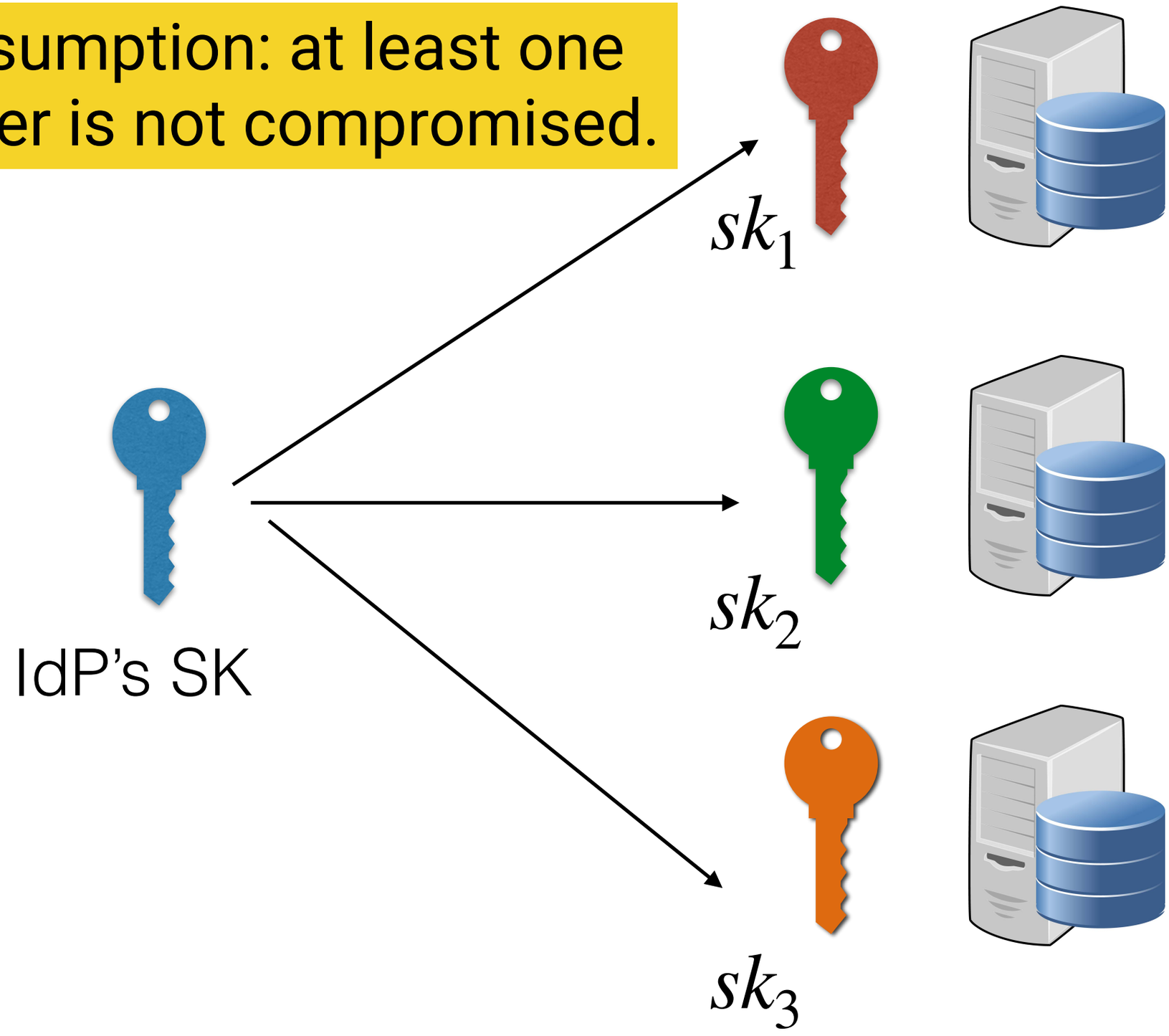
passwords; token-based authentication; threshold cryptography; digital signature; message authentication code; oblivious pseudo-random function

ACM Reference Format:
Shashank Agrawal, Peihan Miao, Payman Mohassel, and Pratyay Mukherjee. 2018. PASTA: PASsword-based Threshold Authentication. In *2018 ACM SIGSAC Conference on Computer and Communications Security (CCS '18), October 15–19, 2018, Toronto, ON, Canada.* ACM, New York, NY, USA, 18 pages. https://doi.org/10.1145/3243734.3243839

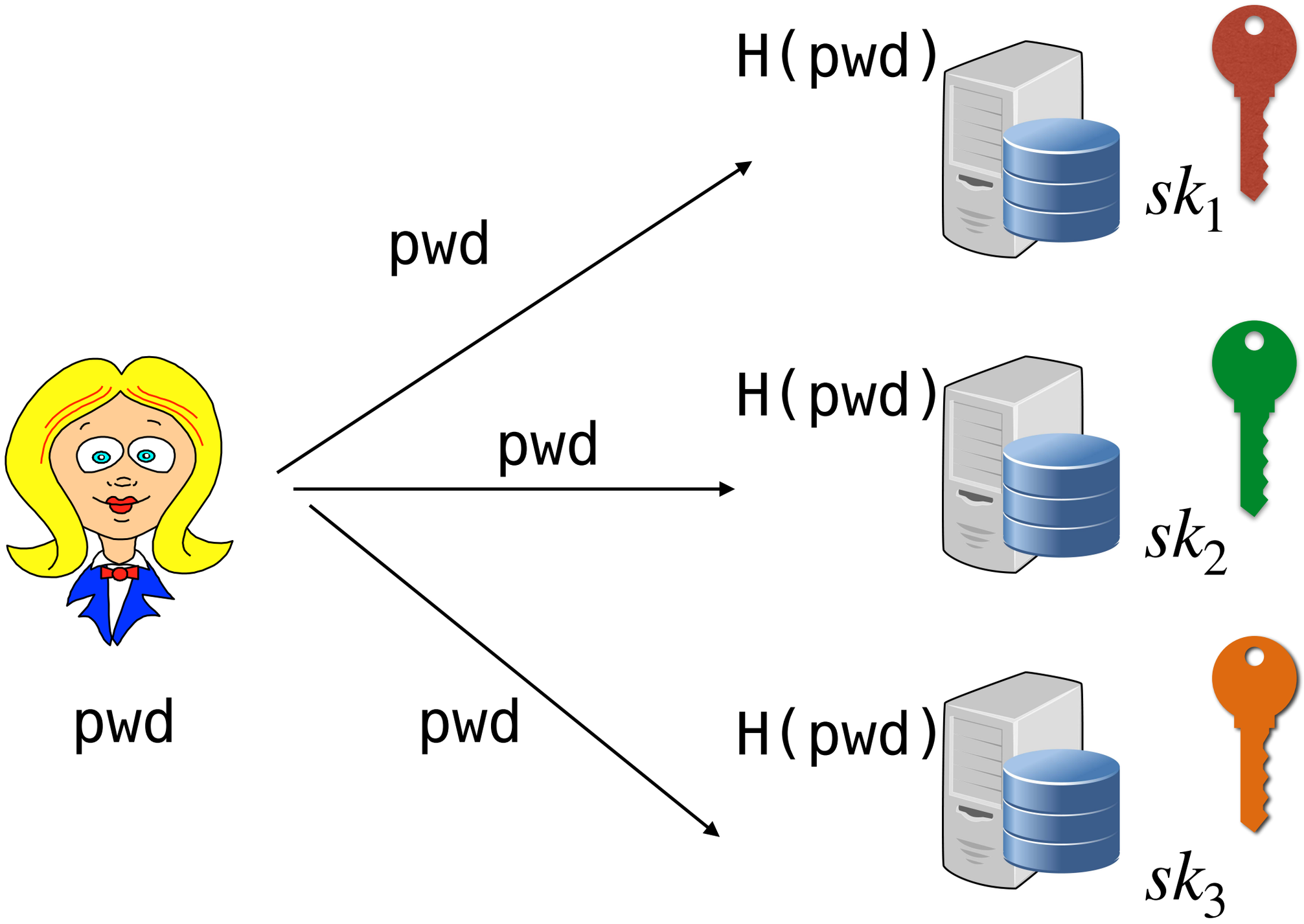
Shashank Agrawal, Peihan Miao, Payman Mohassel, and Pratyay Mukherjee. 2018. PASTA: PASsword-based Threshold Authentication. In Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS '18).

Threshold Token Issuance

Assumption: at least one server is not compromised.

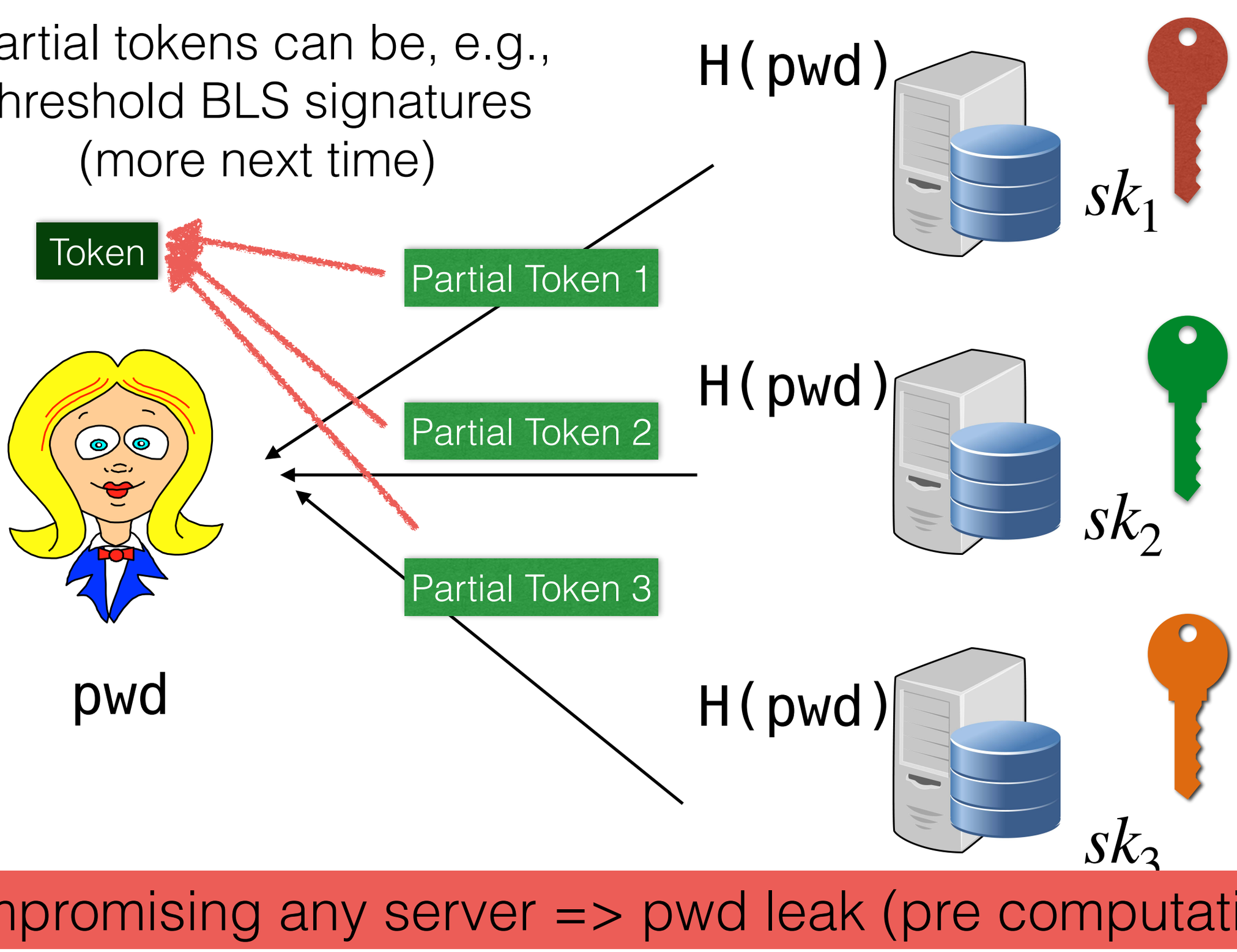


Naive protocol (registration)



Naive protocol (authorize)

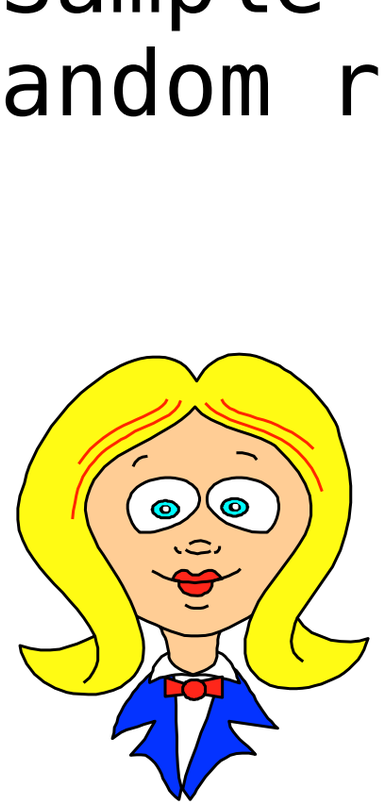
Partial tokens can be, e.g., threshold BLS signatures (more next time)



Compromising any server => pwd leak (pre computation)

Threshold OPRF to avoid leaking passwords

Sample
random r

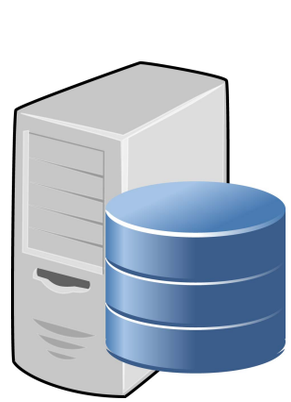


pwd

$H(pwd)^r$

$H(pwd)^r$

$H(pwd)^r$

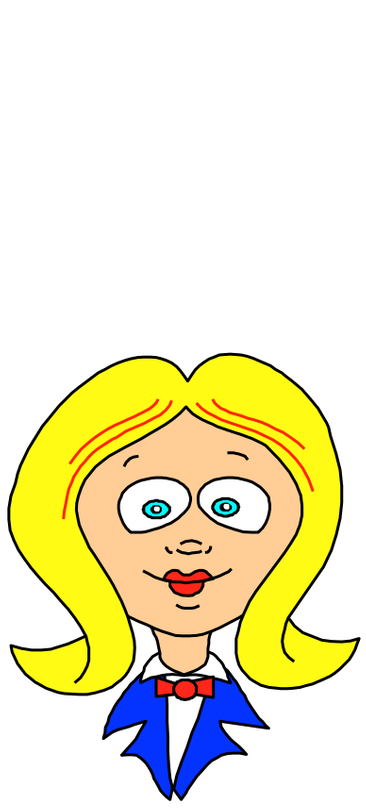


sk_1 $k_{PRF,1}$

sk_2 $k_{PRF,2}$

sk_3 $k_{PRF,3}$

Threshold OPRF to avoid leaking passwords

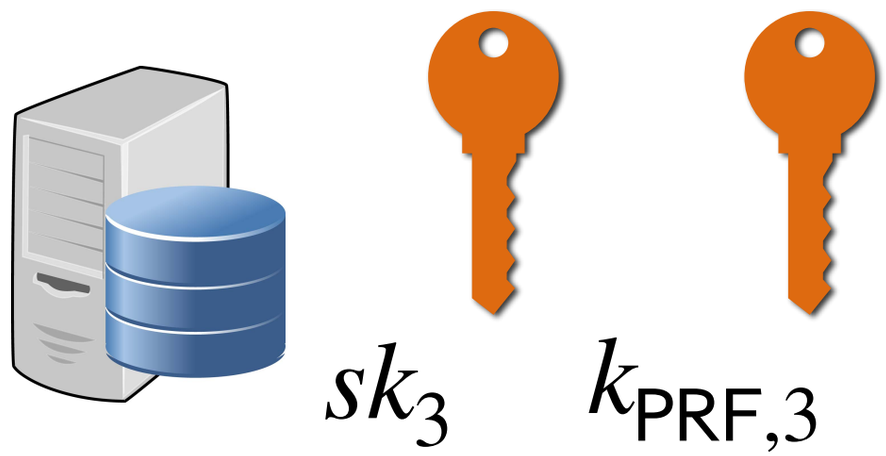


pwd

$\mathrm{H}(\mathrm{pwd})^{r \cdot k_1}$

$\mathrm{H}(\mathrm{pwd})^{r \cdot k_2}$

$\mathrm{H}(\mathrm{pwd})^{r \cdot k_3}$

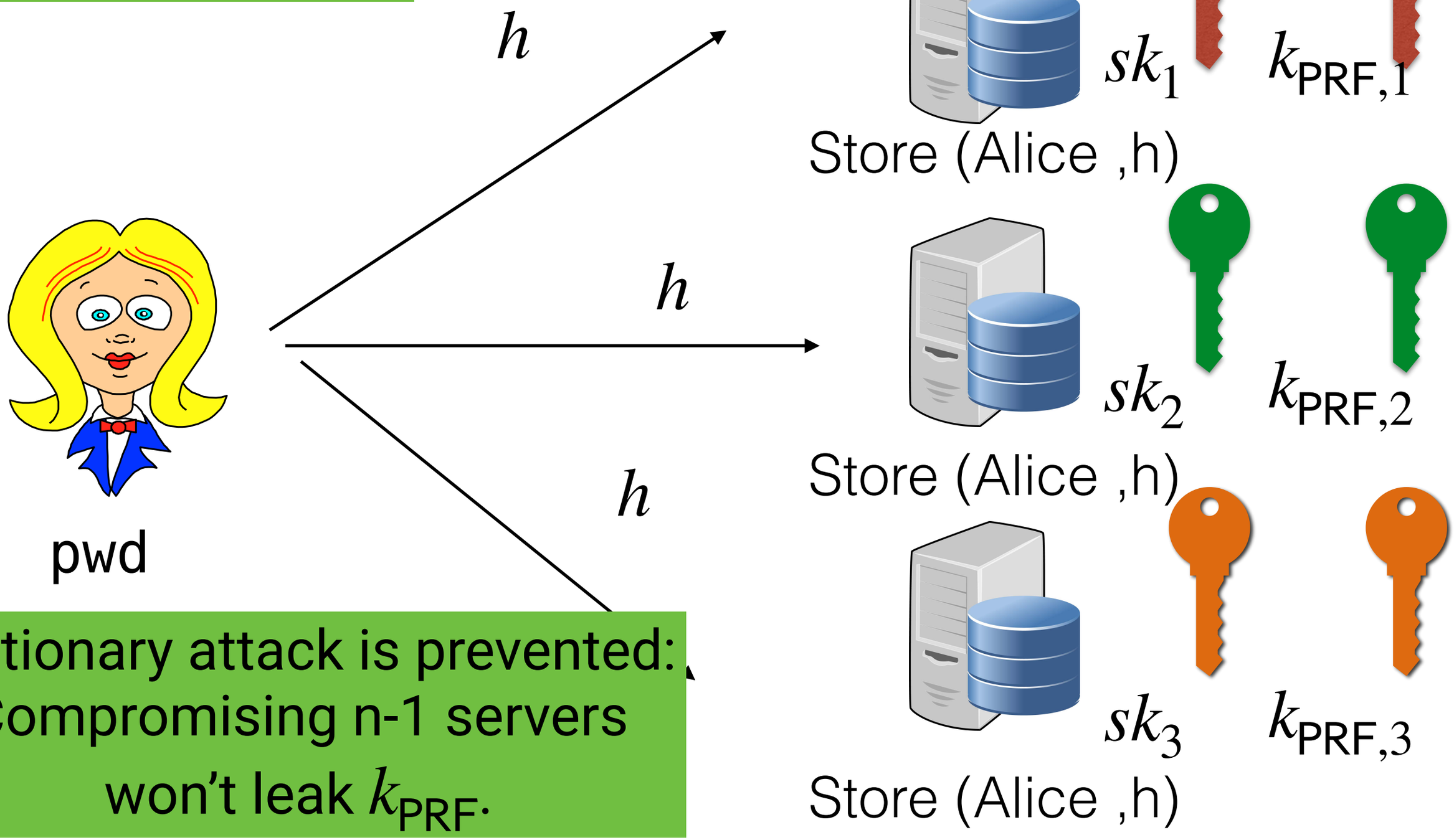


User computes result:

$h = H(\mathrm{pwd})^{\mathrm{k}}$

Threshold OPRF to avoid leaking passwords

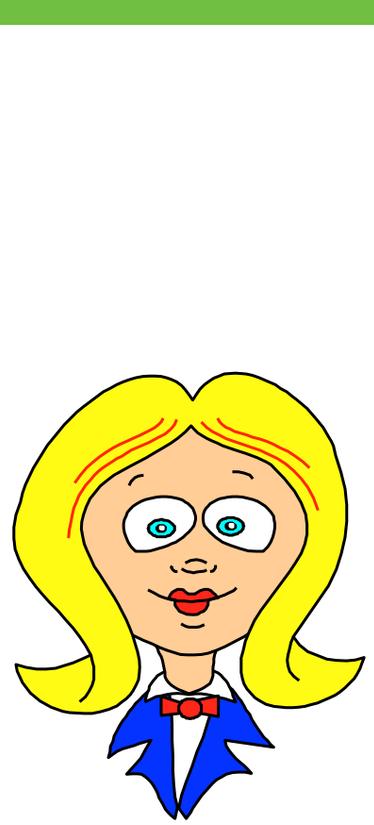
Now the user is registered.



Dictionary attack is prevented: Compromising n-1 servers won't leak k_{PRF}.

Threshold OPRF to avoid leaking passwords

Same flow for authorization



pwd

$H(pwd)^r$

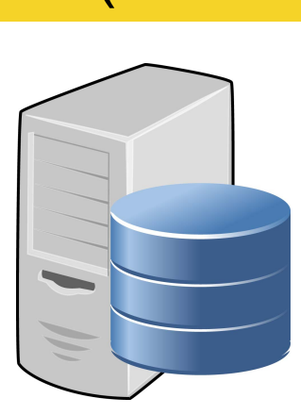
$H(pwd)^r$

$H(pwd)^r$

sk_1

$k_{PRF,1}$

Store (Alice ,h)



sk_2

$k_{PRF,2}$

Store (Alice ,h)

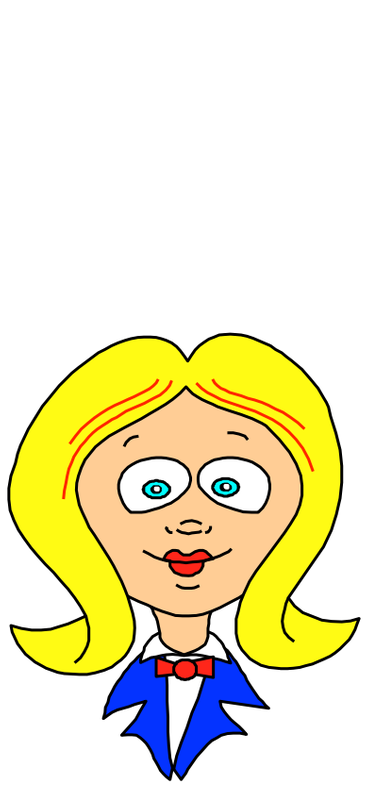
sk_3

$k_{PRF,3}$

Store (Alice ,h)

Threshold OPRF to avoid leaking passwords

Same flow for authorization



pwd

$\mathrm{H}(\mathrm{pwd})^{r \cdot k_1}$

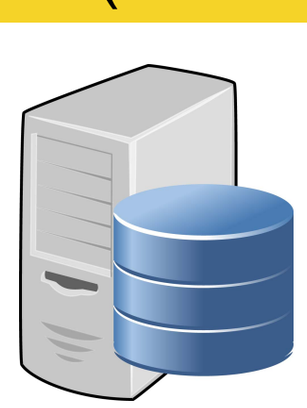
$\mathrm{H}(\mathrm{pwd})^{r \cdot k_2}$

$\mathrm{H}(\mathrm{pwd})^{r \cdot k_3}$

sk_1

$k_{\mathrm{PRF},1}$

Store (Alice ,h)



sk_2

$k_{\mathrm{PRF},2}$

Store (Alice ,h)

sk_3

$k_{\mathrm{PRF},3}$

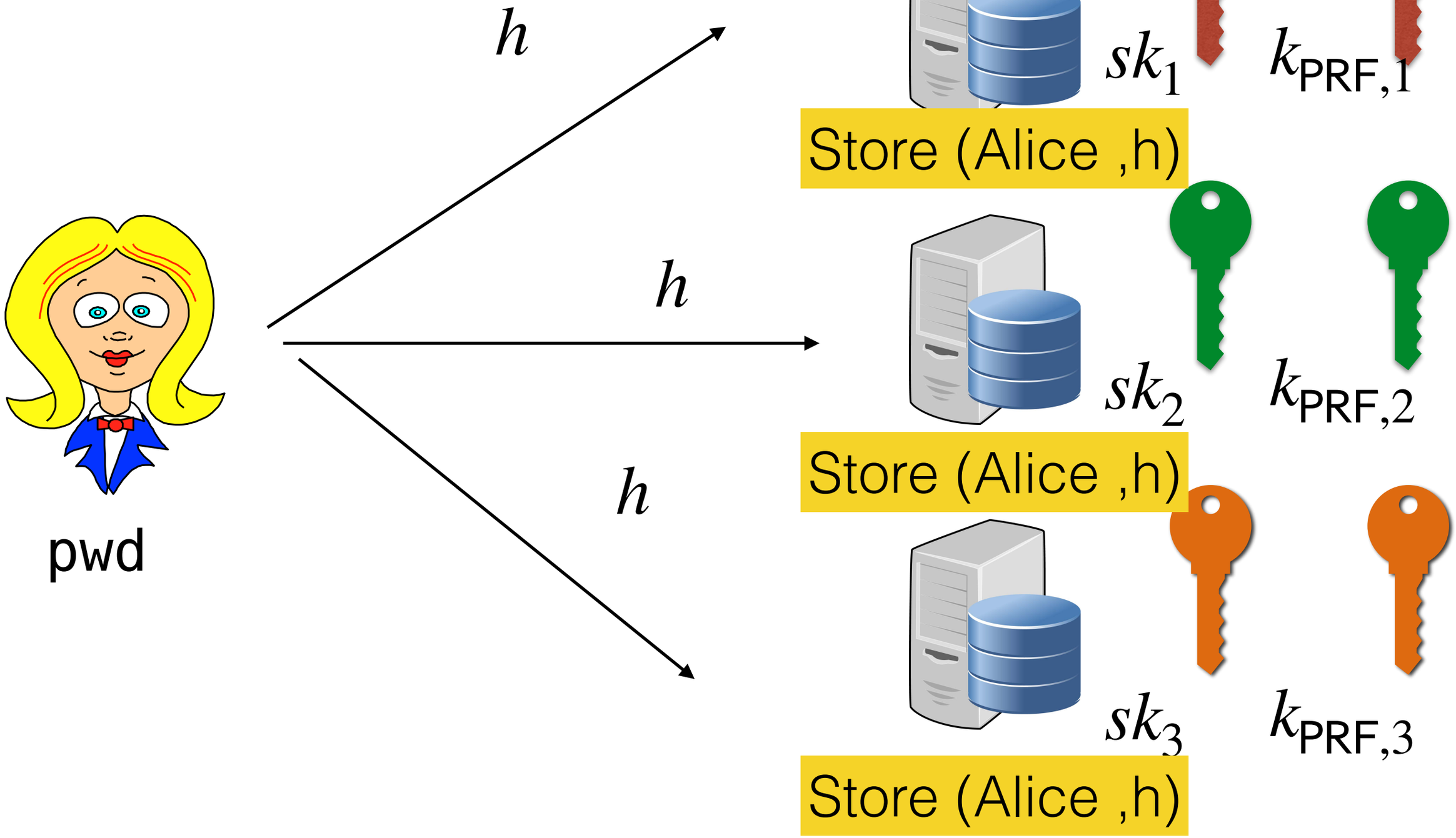
Store (Alice ,h)

User computes result:

$$h = H(\mathrm{pwd})^{k}$$

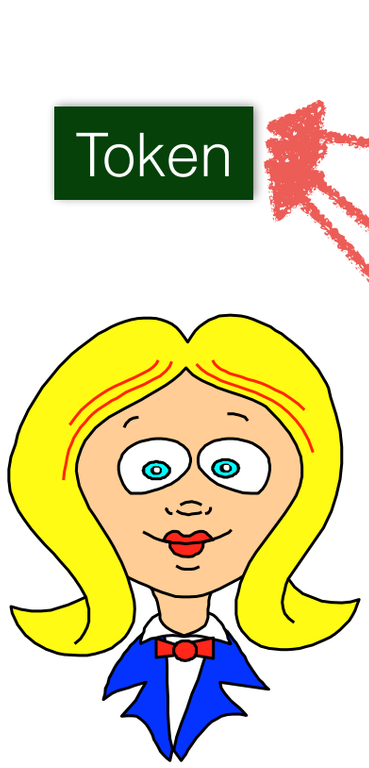
Threshold OPRF to avoid leaking passwords

Same flow for authorization



Threshold OPRF to avoid leaking passwords

Same flow for authorization



pwd

Token

Partial Token 1

Partial Token 2

Partial Token 3

sk_1

$k_{\text{PRF},1}$

Store (Alice ,h)

sk_2

$k_{\text{PRF},2}$

Store (Alice ,h)

sk_3

$k_{\text{PRF},3}$

Store (Alice ,h)

That’s a lot of round trips!

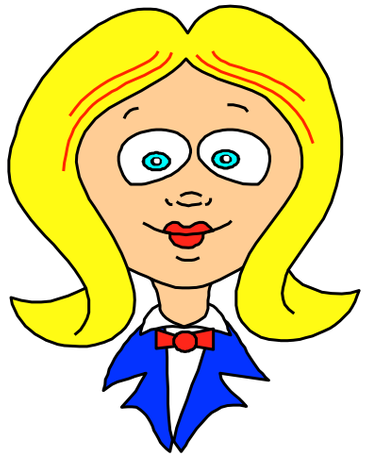
Recall

- “Every **100ms** delay cost Amazon **1%** of sales.” — Amazon 2006
- “With a 0.1s improvement in site speed, we observed that retail consumers spent almost 10% more” — Deloitte 2020

Threshold OPRF to avoid leaking passwords

I'm Alice. $h(\text{pwd})^r$

Store (Alice, h)



Ci := Enc(h, partial token i)

$$h_i = H(\text{pwd})^{r \cdot k_{\text{PRF},i}}$$

Locally, Alice

- recovers h from h_i
- decrypt Ci to get partial tokens
- assemble the final token.

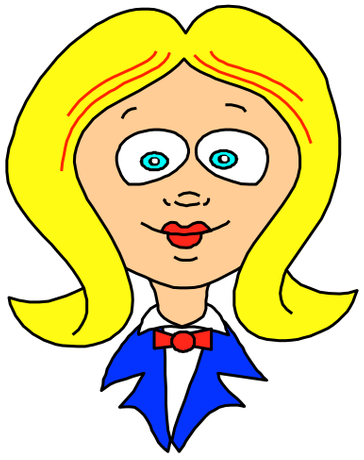
This introduces a security bug!

Fix in PASTA: use pair-wise keys to encrypt

PASTA protocol

I'm Alice. $\mathsf{H}(\mathsf{pwd})^r$ →

sk_i $k_{\mathsf{PRF},i}$



← $h_i = H(\mathsf{pwd})^{r \cdot k_{\mathsf{PRF},i}}$

Locally, Alice gets h from h_i

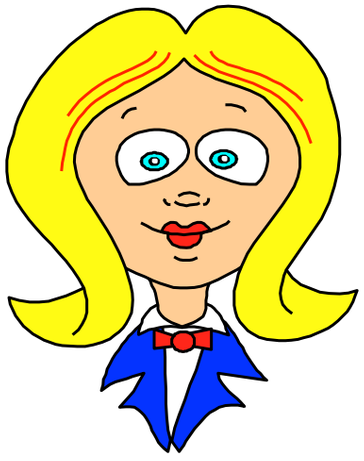
$k_{\mathsf{Alice},i} = \mathsf{H}(h, i)$ →

Store (Alice, $k_{\mathsf{Alice},i}$)

PASTA protocol

I'm Alice. $h(\text{pwd})^r$ →

Store (Alice, $k_{\text{Alice},i}$)



Ci := Enc($k_{\text{Alice},i}$, partial token i)

← $h_i = H(\text{pwd})^{r \cdot k_{\text{PRF},i}}$

Locally, Alice

- recovers h from h_i
- decrypt Ci using $k_{\text{Alice},i} := H(h, i)$ to get partial tokens
- assemble the final token.

PASTA: PASsword-based Threshold Authentication

Shashank Agrawal
Visa Research
shaagraw@visa.com

Peihan Miao*
University of California, Berkeley
peihan@berkeley.edu

Payman Mohassel
Visa Research
pmohasse@visa.com

Pratyay Mukherjee
Visa Research
pratmukh@visa.com

ABSTRACT

Token-based authentication is commonly used to enable a single-sign-on experience on the web, in mobile applications and on enterprise networks using a wide range of open standards and network authentication protocols: clients sign on to an identity provider using their username/password to obtain a cryptographic token generated with a master secret key, and store the token for future accesses to various services and applications. The authentication server(s) are single point of failures that if breached, enable attackers to forge arbitrary tokens or mount offline dictionary attacks to

KEYWORDS

passwords; token-based authentication; threshold cryptography; digital signature; message authentication code; oblivious pseudo-random function

ACM Reference Format:
Shashank Agrawal, Peihan Miao, Payman Mohassel, and Pratyay Mukherjee. 2018. PASTA: PASsword-based Threshold Authentication. In *2018 ACM SIGSAC Conference on Computer and Communications Security (CCS '18), October 15–19, 2018, Toronto, ON, Canada.* ACM, New York, NY, USA, 18 pages. https://doi.org/10.1145/3243734.3243839

Good ideas

- OPRF to defeat dictionary attacks
- Encrypting without authentication to reduce round trips

Downsides

- require changes to IdP, high adoption barrier.
- All servers are run by the same party, increasing the risk of simultaneous compromise

Next:

Anonymous Credentials