2550 Intro to cybersecurity

L5

abhi shelat

How U2F foils phishing

User,sk

My browser

1. In the beginning, I register with G and setup 2FA.

User,pk

How U2F foils phishing

User,sk

My browser

2. I am tricked into clicking on fake G login, who tries a PITM attack.

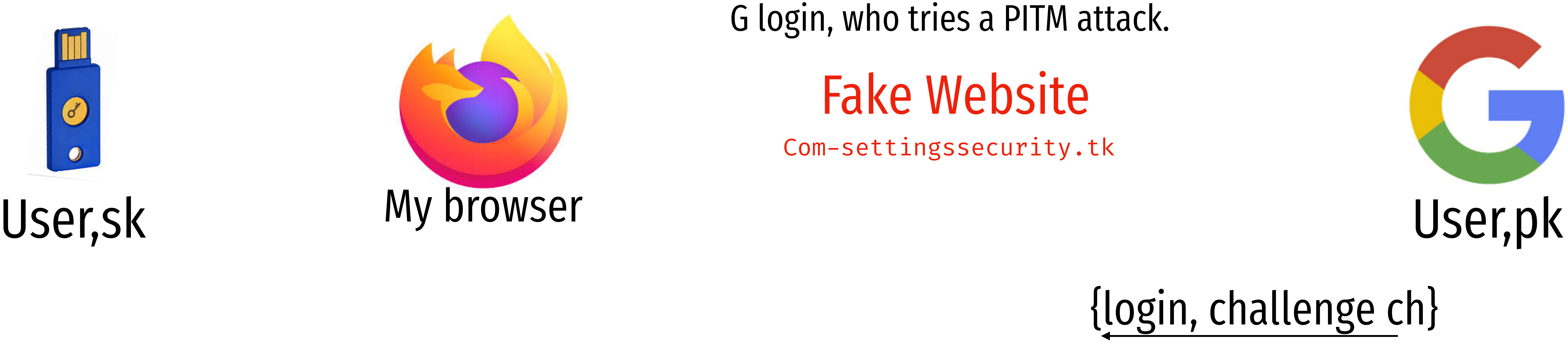
Fake Website

Com-settingssecurity.tk

User,pk

How U2F foils phishing

2. I am tricked into clicking on fake G login, who tries a PITM attack.



User,sk

My browser

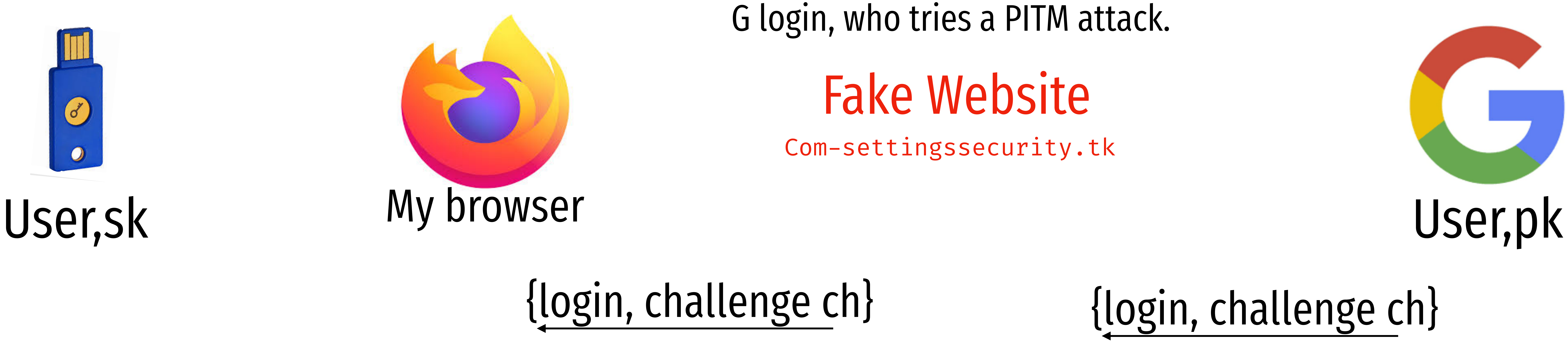
Fake Website

Com-settingssecurity.tk

User,pk

{login, challenge ch}

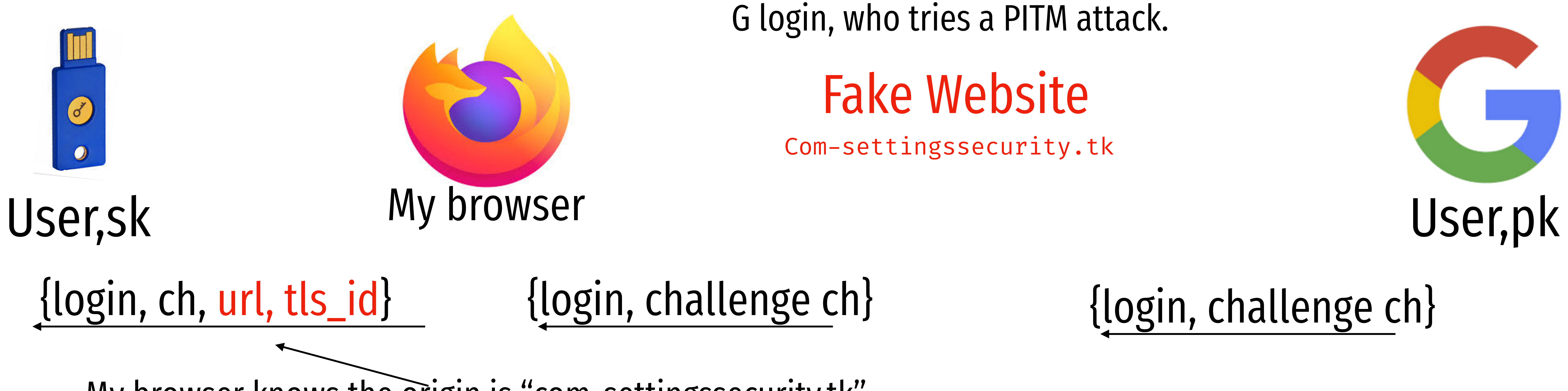
How U2F foils phishing



2. I am tricked into clicking on fake G login, who tries a PITM attack.

How U2F foils phishing

2. I am tricked into clicking on fake
G login, who tries a PITM attack.



Fake Website

Com-settingssecurity.tk

User,sk

My browser

User,pk

{login, ch, url, tls_id}

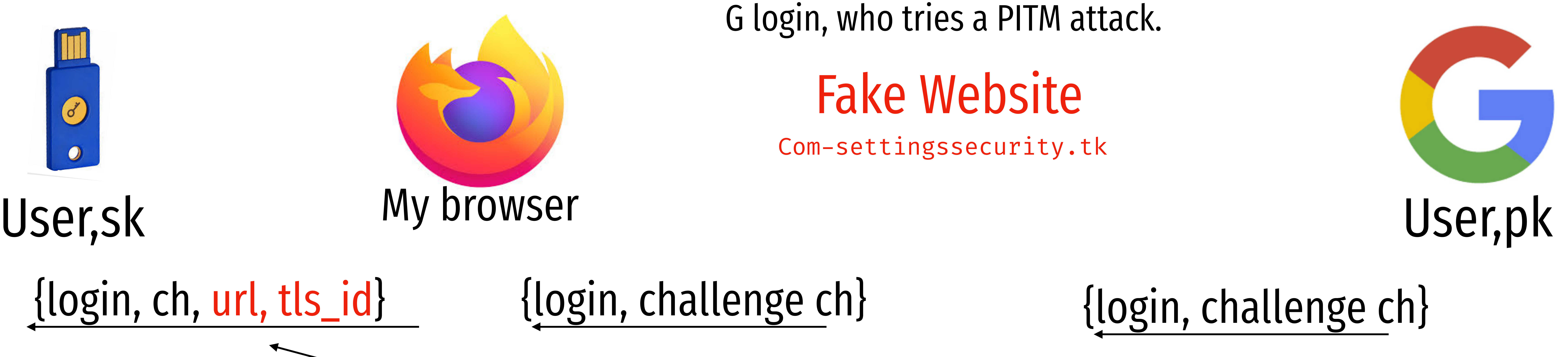
{login, challenge ch}

{login, challenge ch}

My browser knows the origin is “com-settingssecurity.tk”
instead of google.com, and passes this string as url.

How U2F foils phishing

2. I am tricked into clicking on fake
G login, who tries a PITM attack.



User,sk

My browser

Fake Website

Com-settingssecurity.tk

User,pk

{login, ch, url, tls_id}

{login, challenge ch}

{login, challenge ch}

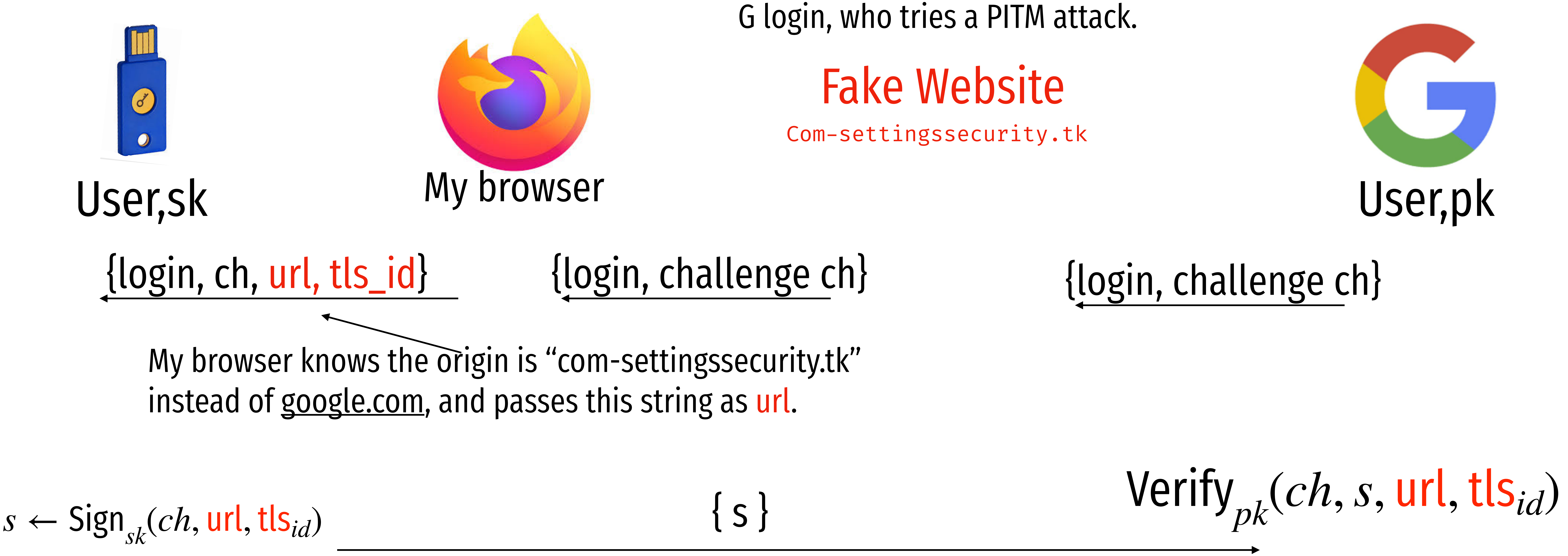
My browser knows the origin is “com-settingssecurity.tk”
instead of google.com, and passes this string as url.

$s \leftarrow \text{Sign}_{sk}(ch, \text{url}, \text{tls}_{id})$

Sign challenge using sk

The 2FA key signs this with url=com-settings...

How U2F foils phishing



2. I am tricked into clicking on fake G login, who tries a PITM attack.

Fake Website

Com-settingssecurity.tk

User,sk

My browser

User,pk

{login, ch, url, tls_id}

{login, challenge ch}

{login, challenge ch}

My browser knows the origin is “com-settingssecurity.tk” instead of google.com, and passes this string as url.

$s \leftarrow \text{Sign}_{sk}(ch, \text{url}, \text{tls}_{id})$

Sign challenge using sk

{ s }

$\text{Verify}_{pk}(ch, s, \text{url}, \text{tls}_{id})$

The 2FA key signs this with url=com-settings...

Google reject the authentication and detects the attack!

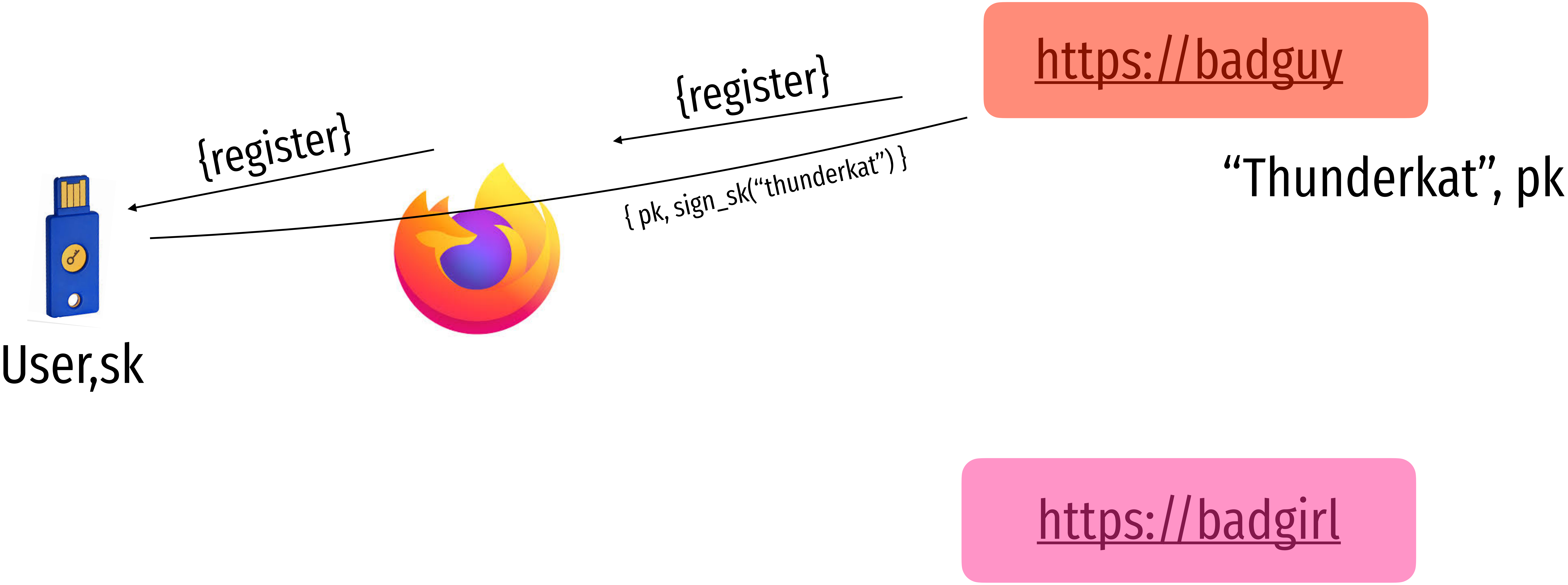
The Tracking problem

https://badguy

User,sk

https://badgirl

The Tracking problem



https://badguy

{register}

{register}

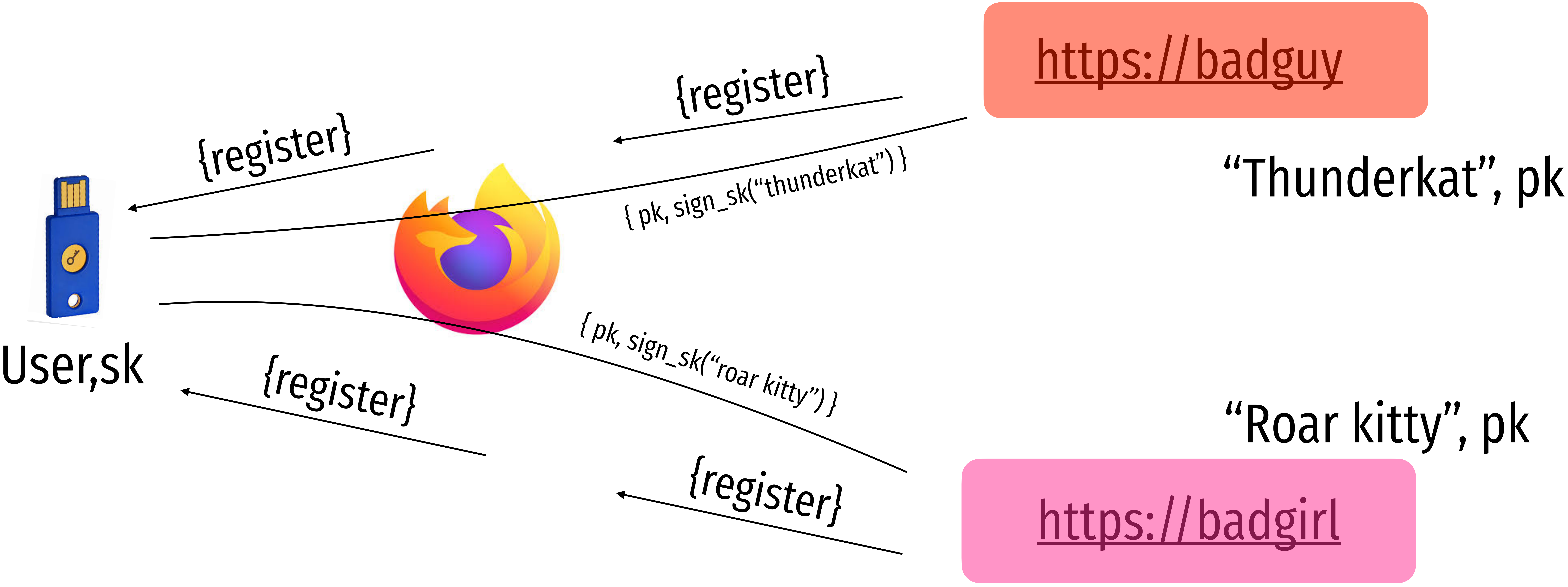
{ pk, sign_sk(“thunderkat”) }

“Thunderkat”, pk

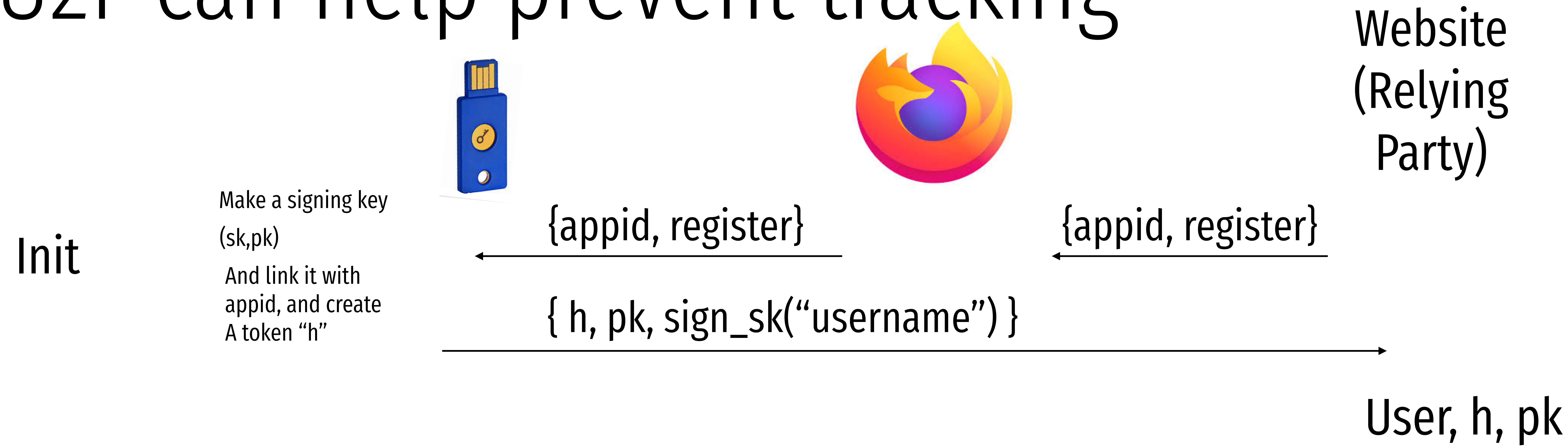
User,sk

https://badgirl

The Tracking problem



U2F can help prevent tracking



Website (Relying Party)

Init

Make a signing key (sk,pk)

And link it with appid, and create A token “h”

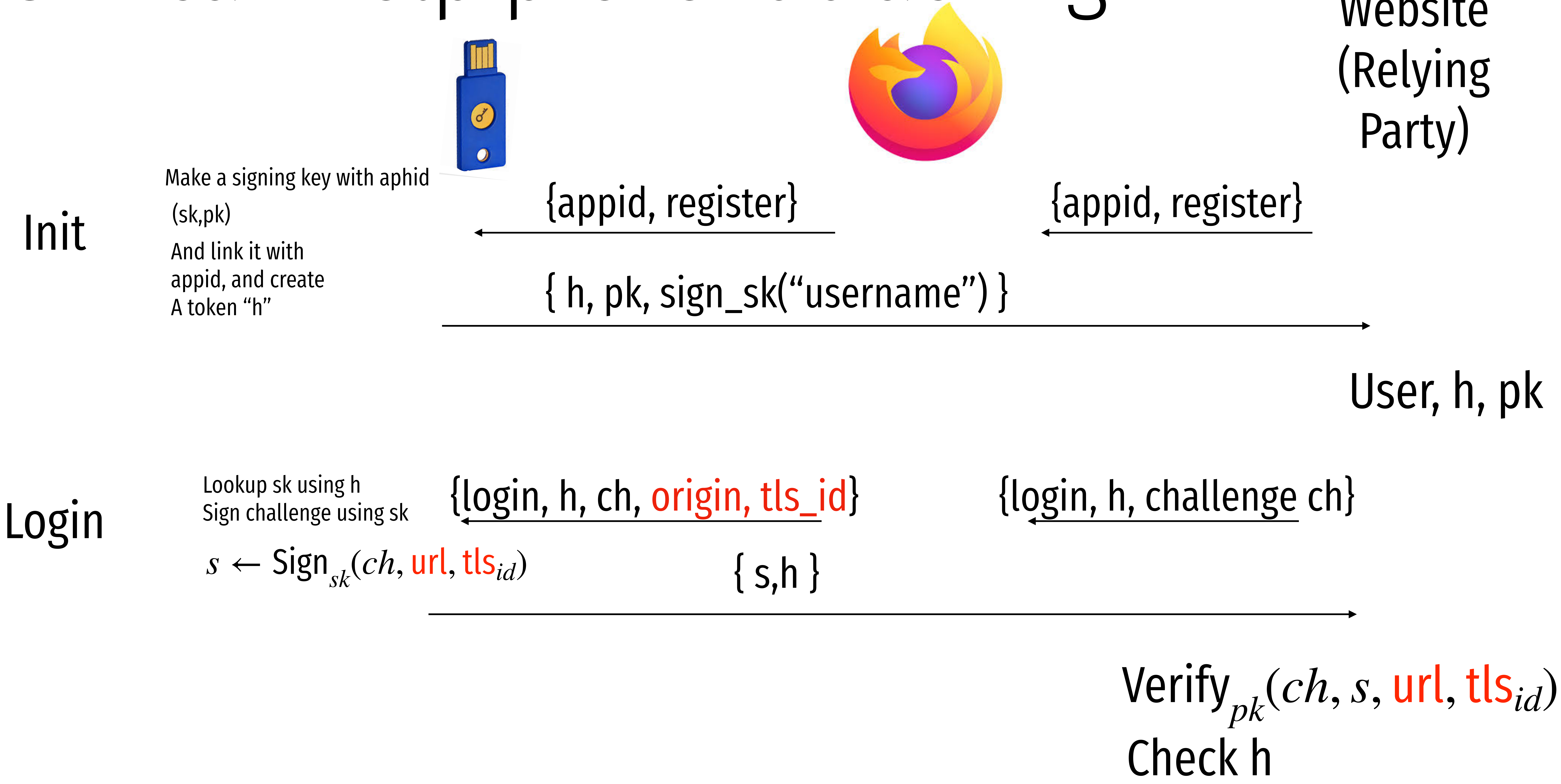
{appid, register}

{appid, register}

{ h, pk, sign_sk(“username”) }

User, h, pk

U2F can help prevent tracking



Website (Relying Party)

Init

Make a signing key with aphid (sk,pk)

And link it with appid, and create A token “h”

{appid, register}

{appid, register}

{ h, pk, sign_sk(“username”) }

User, h, pk

Login

Lookup sk using h
Sign challenge using sk

$s \leftarrow \text{Sign}_{sk}(ch, \text{url}, \text{tls}_{id})$

{login, h, ch, origin, tls_id}

{login, h, challenge ch}

{ s,h }

$\text{Verify}_{pk}(ch, s, \text{url}, \text{tls}_{id})$

Check h

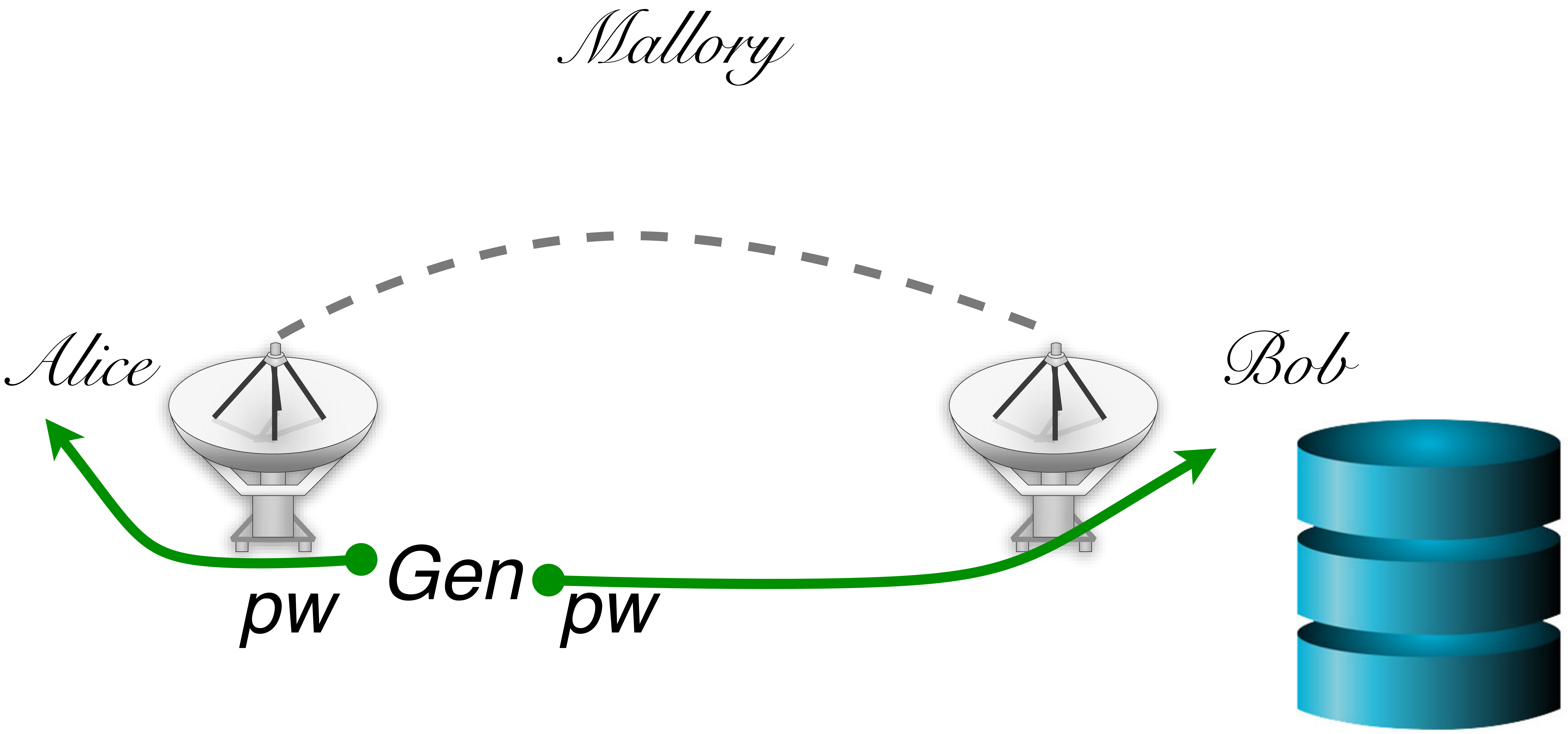
```
Sending request with appId: https://u2f.bin.coffee
{
  "version": "U2F_V2",
  "challenge": "uQnl3M4Rj3FZgs6WjyLaZAfwRh4"
}

Got response:
{
  "clientData": "eyJjaGFsbGVuZ2UiOiJ1UW5sM000UmozRlpnczZXanlMYVpBZndSaDQiLCJvcmlnaW4iOiJodHRwczovL3UyZi5iaW4uY29mZmVlIiwidHlwIjoibmF2
  "errorCode": 0,
  "registrationData": “BQRSuRLPv0p5udQ55vVhucf3N50q6…”,
  "version": "U2F_V2"
}

Key Handle: 0r0Z0p0F0E0-0d0W0c0Q0b0X0i020C0w0-0E0v0h0t0T0T0P0_0-090_0a050P0e030u0b0z0l0K0Q0r0O0f0u030_0P020B0J0M0x0D050J0_0d0P0Q0e0j0
Certificate: 3082021c3082…
Attestation Cert
Subject: Yubico U2F EE Serial 14803321578
Issuer: Yubico U2F Root CA Serial 457200631
Validity (in millis): 1136332800000
Attestation Signature
R: 00b11e3efe5ae5ac7ca0e0d4fe2c5b5cf18a2531c0f4f70b11c30b72b5f946a9a3
S: 0f37ab2d4f93ebcdaed0a51b4b17fb93403db9873f0e9cce36f17b1502734bb2
[PASS] Signature buffer has no unnecessary bytes.: 71 == 71
[PASS] navigator.id.finishEnrollment == navigator.id.finishEnrollment
[PASS] uQnl3M4Rj3FZgs6WjyLaZAfwRh4 == uQnl3M4Rj3FZgs6WjyLaZAfwRh4
[PASS] https://u2f.bin.coffee == https://u2f.bin.coffee
[PASS] Verified certificate attestation signature
[PASS] Imported credential public key
Failures: 0 TODOs: 0
```

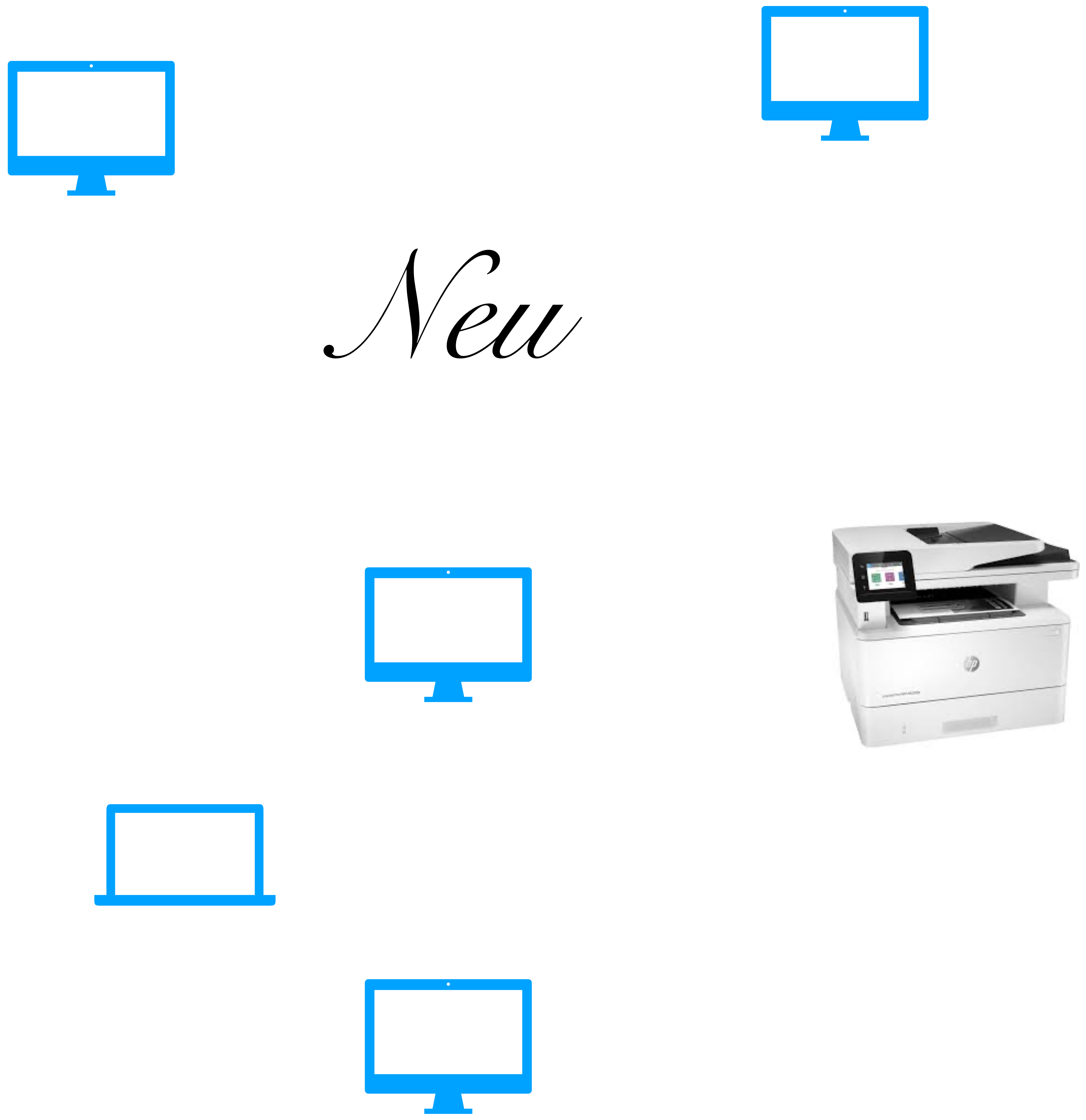
Future without passwords?

Password Security game



More realistic picture of the world

Alice
pw

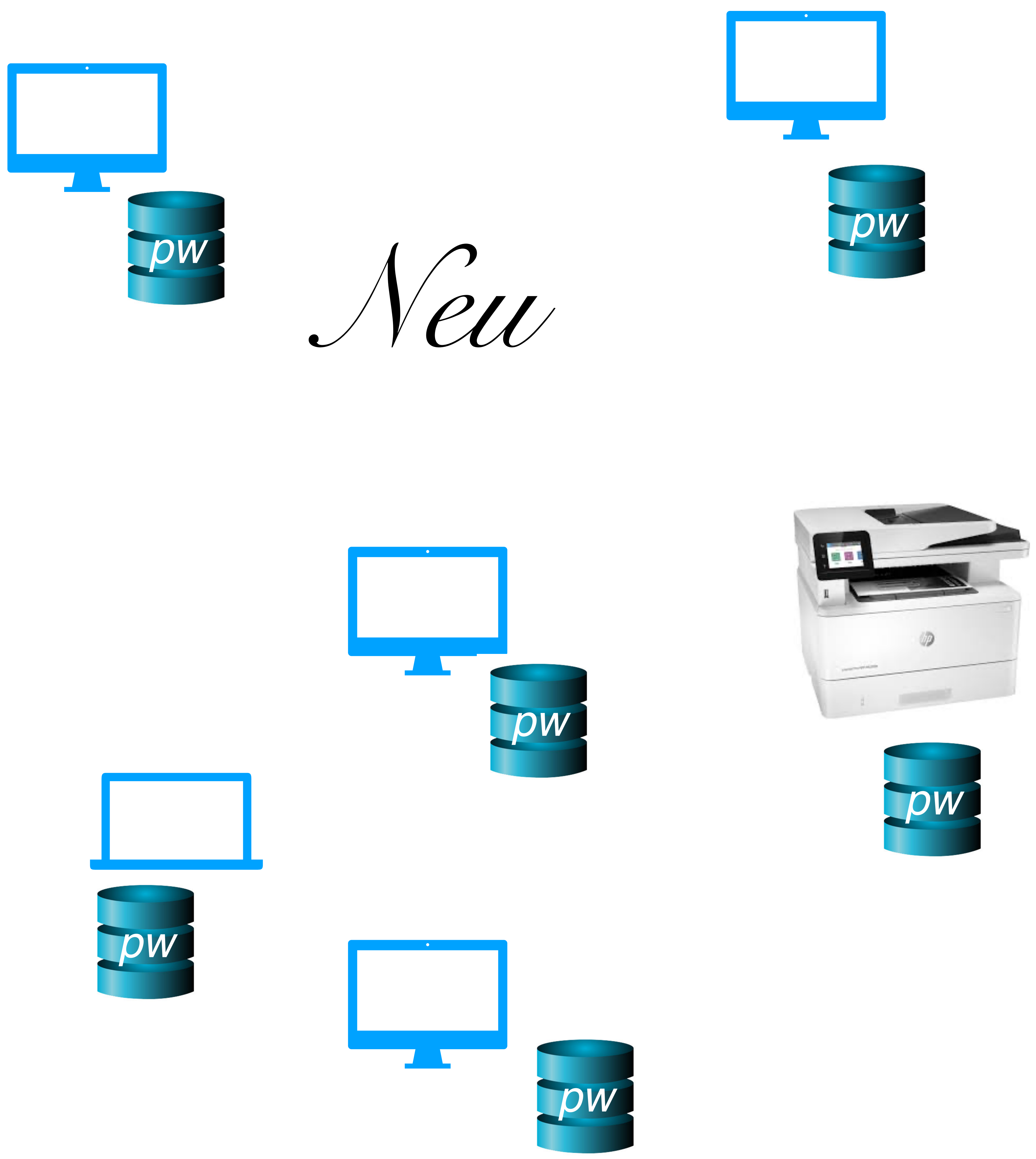


More realistic picture of the world

What are the problems with this solution?

Alice

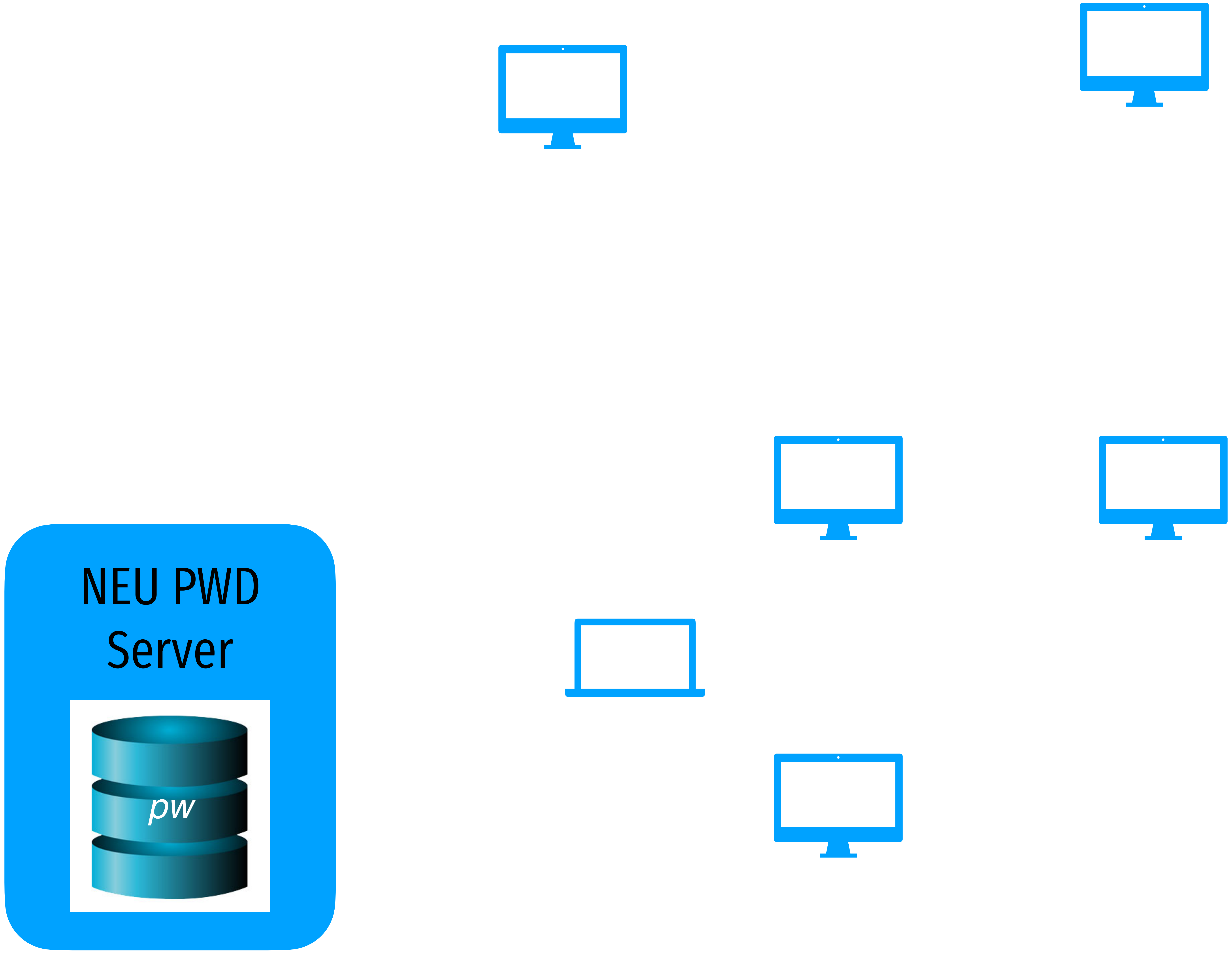
pw



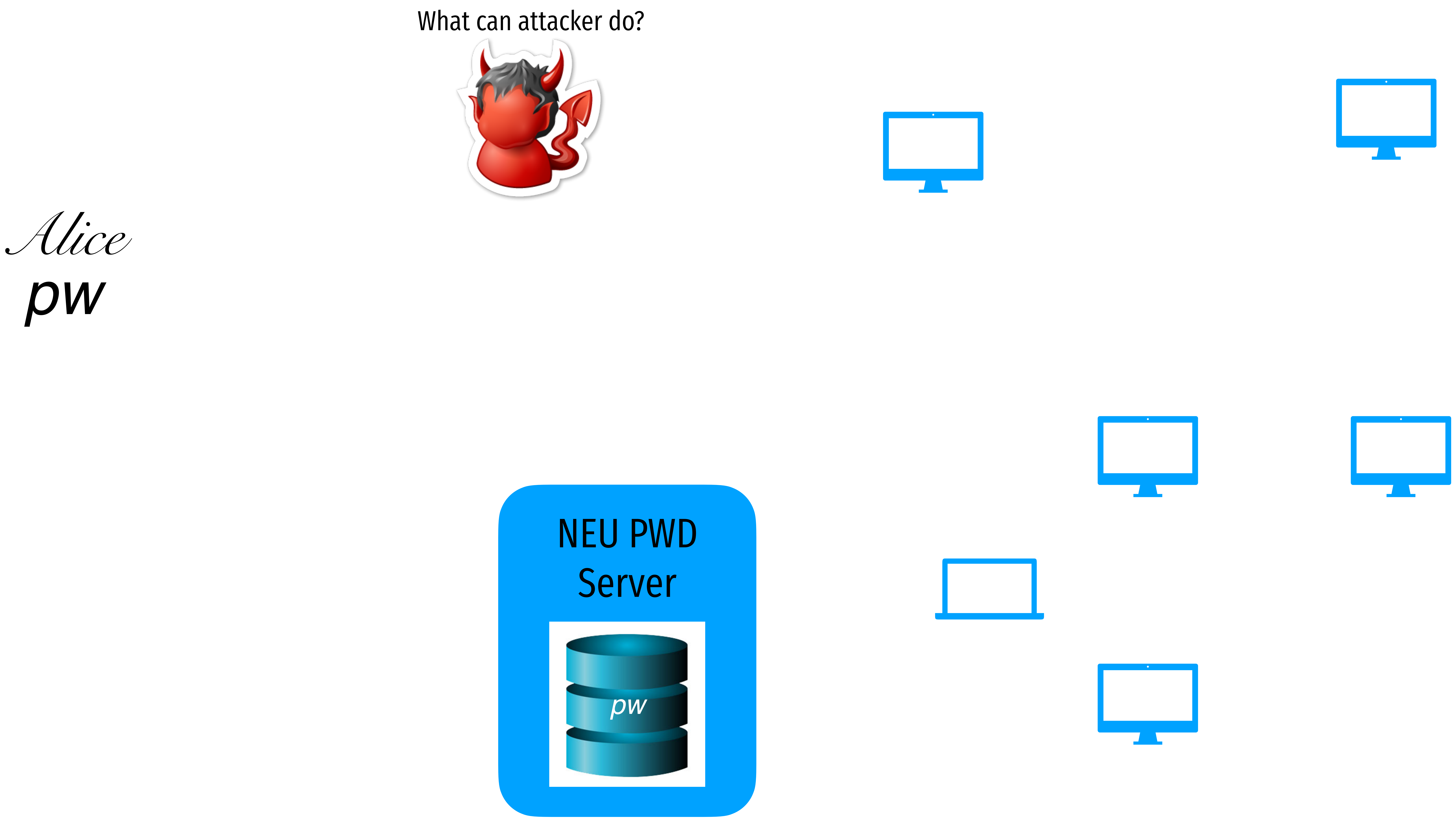
The problem of distributed authentication

Alice

pw



Distributed authentication: Attacker model



What can attacker do?

Alice

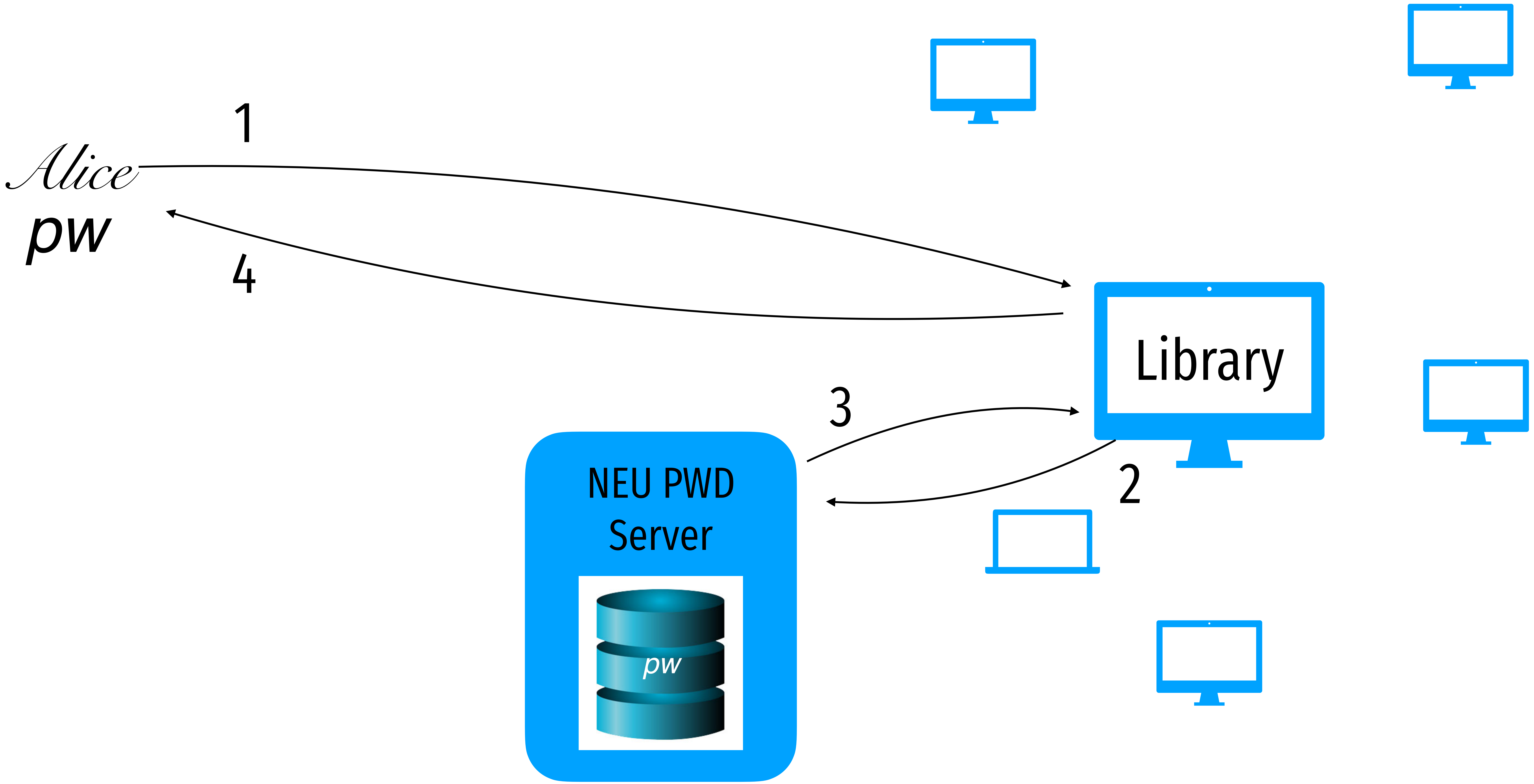
pw

NEU PWD Server

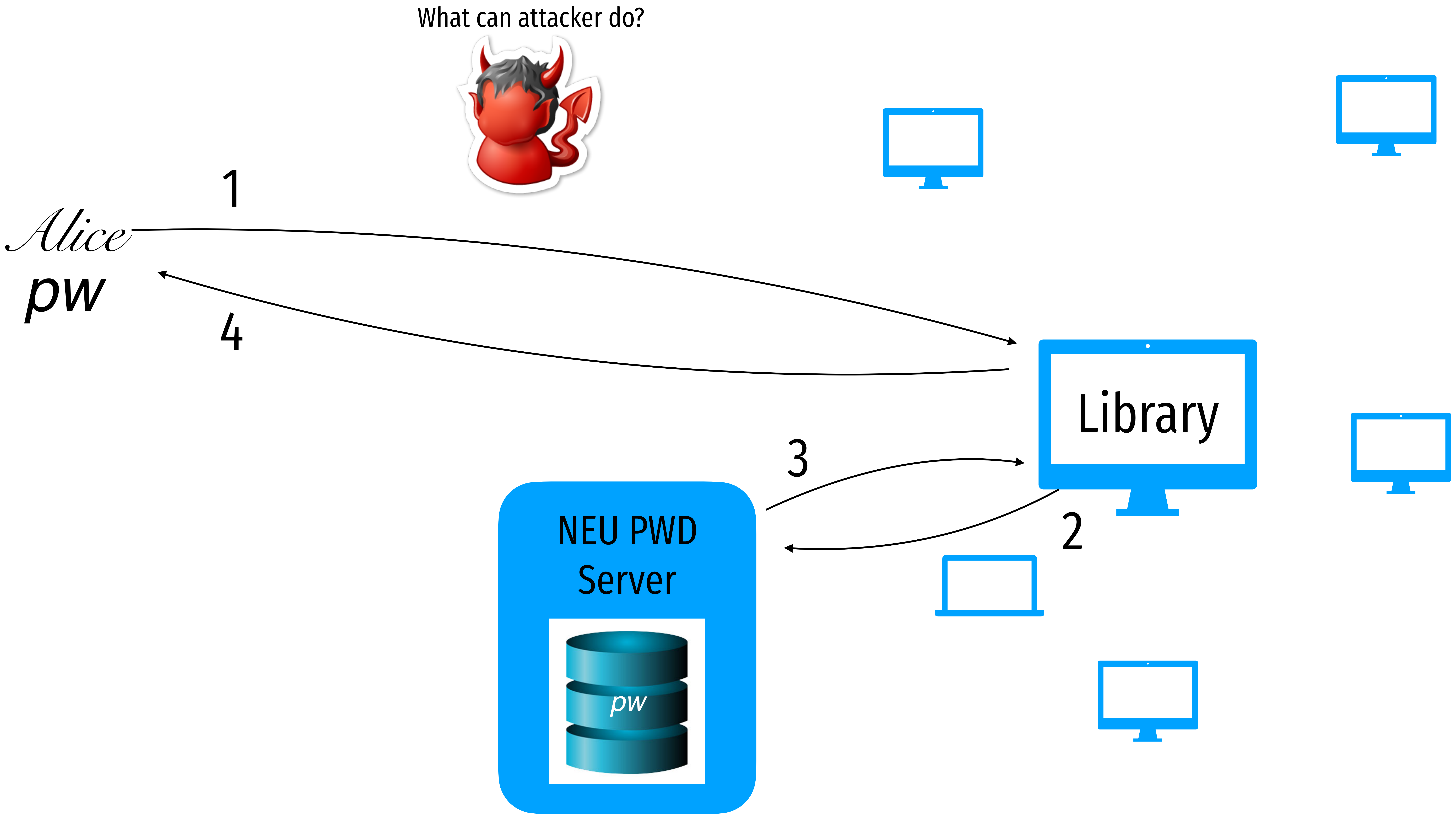
pw

Distributed authentication: Bad Solution

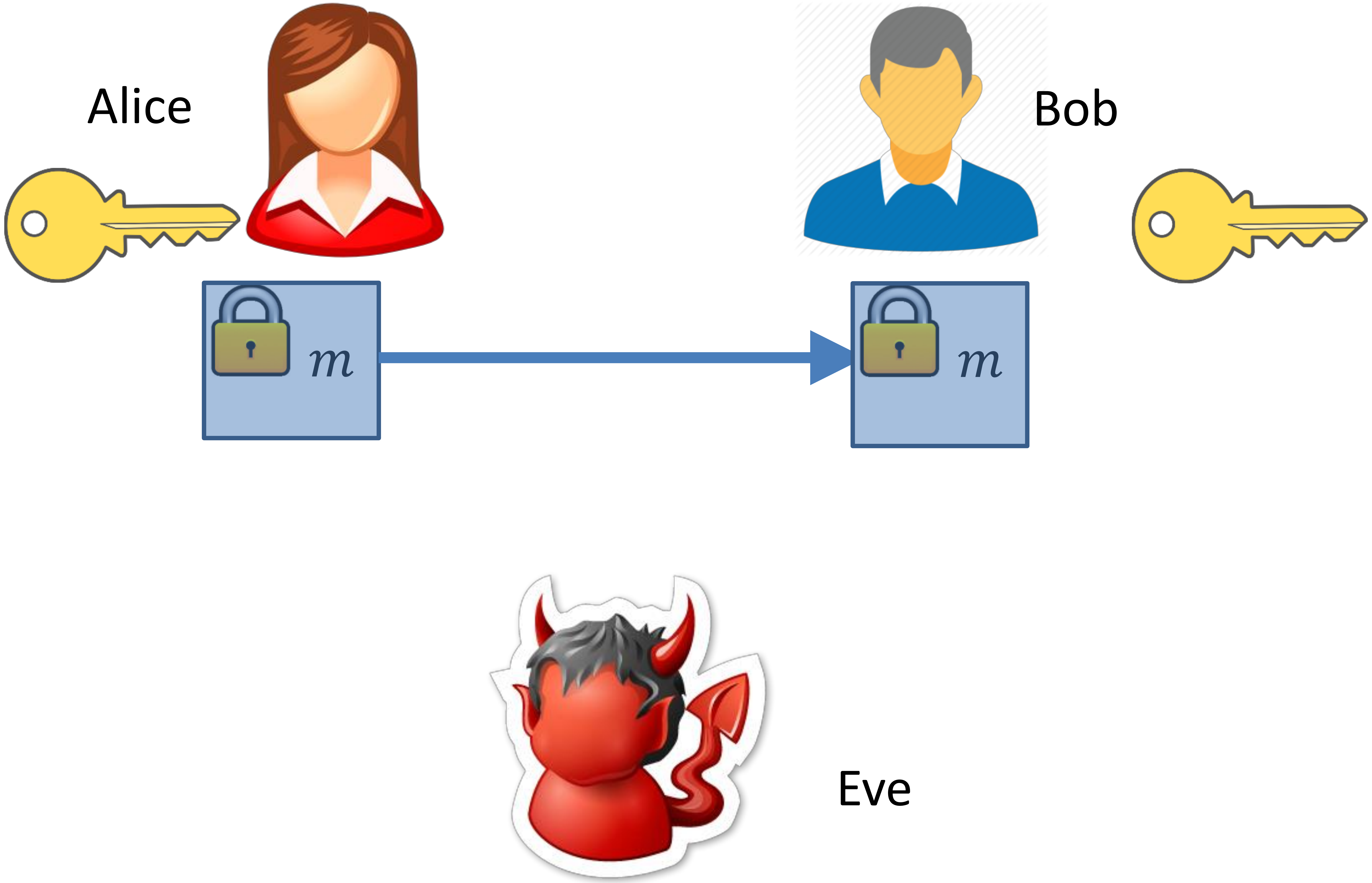
What can attacker do?



Distributed authentication: Bad Solution

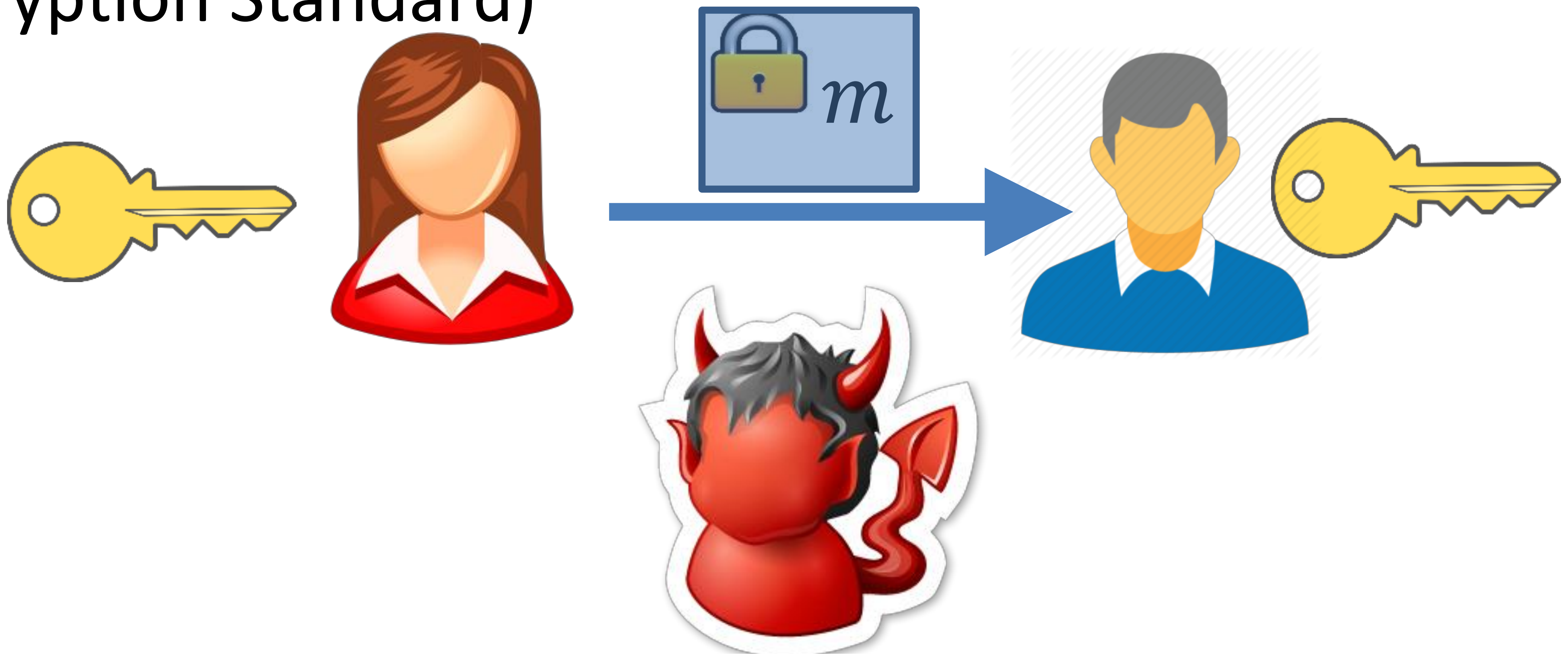


Basic tool: symmetric encryption



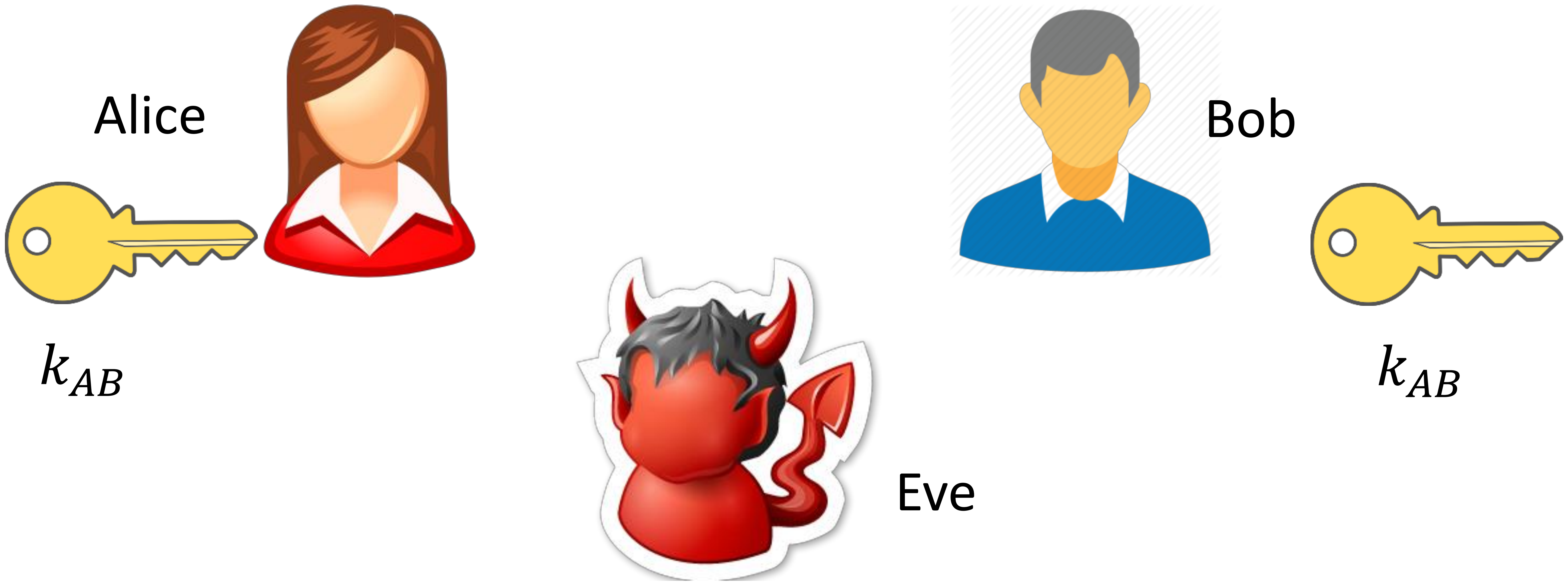
Basic tool: symmetric encryption

- Gen: generates secret key k
- Enc: given k and m output a ciphertext c
  Denote $Enc_k(m)$, $E_k(m)$, $\{m\}_k$
- Dec: given k and c output a message m
- Security (informal):
  Whatever Eve can learn on m given c can be learned without c
- Examples:
  - DES (Data Encryption Standard)
  - AES (Advanced Encryption Standard)



Authentication from Encryption

- Alice and Bob share a key
- They communicate over an insecure channel
- Alice wants to prove her identity to Bob
- Eve's goal: impersonate Alice



Alice

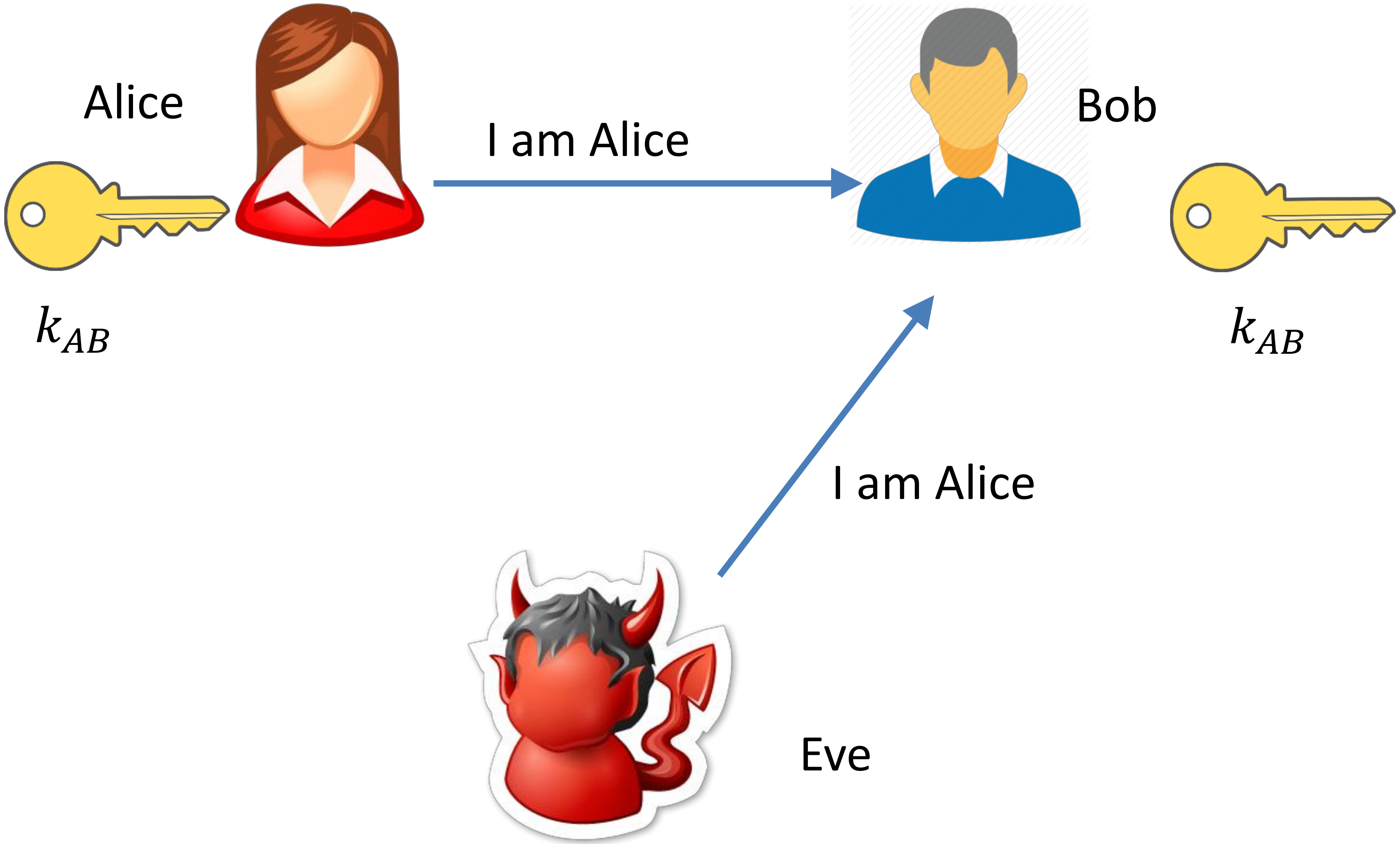
Bob

k_{AB}

k_{AB}

Eve

Attempt #1



Alice

I am Alice

Bob

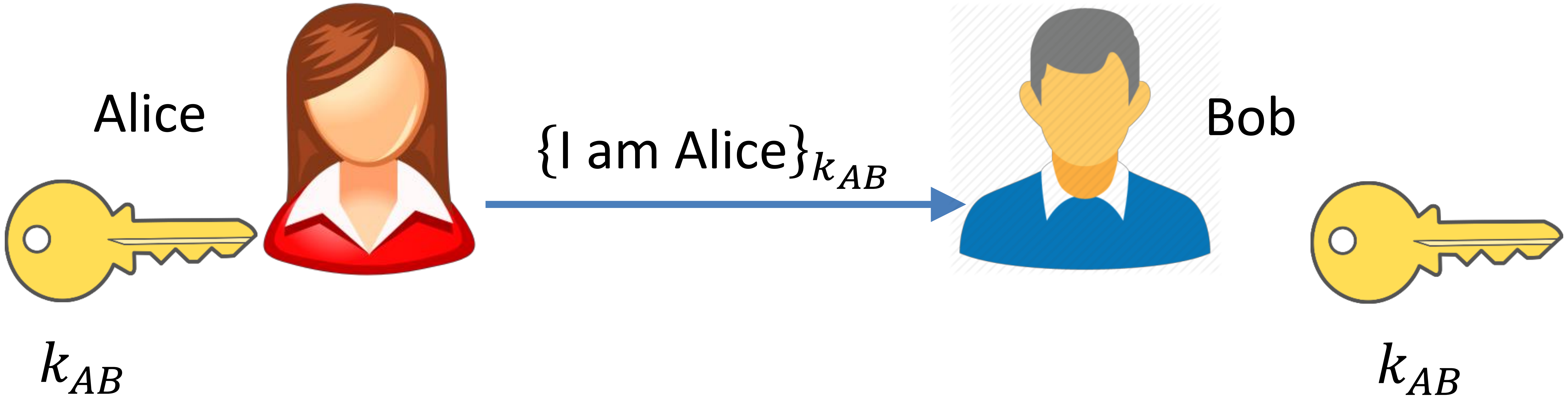
k_{AB}

k_{AB}

I am Alice

Eve

Attempt #2: use the key



Alice

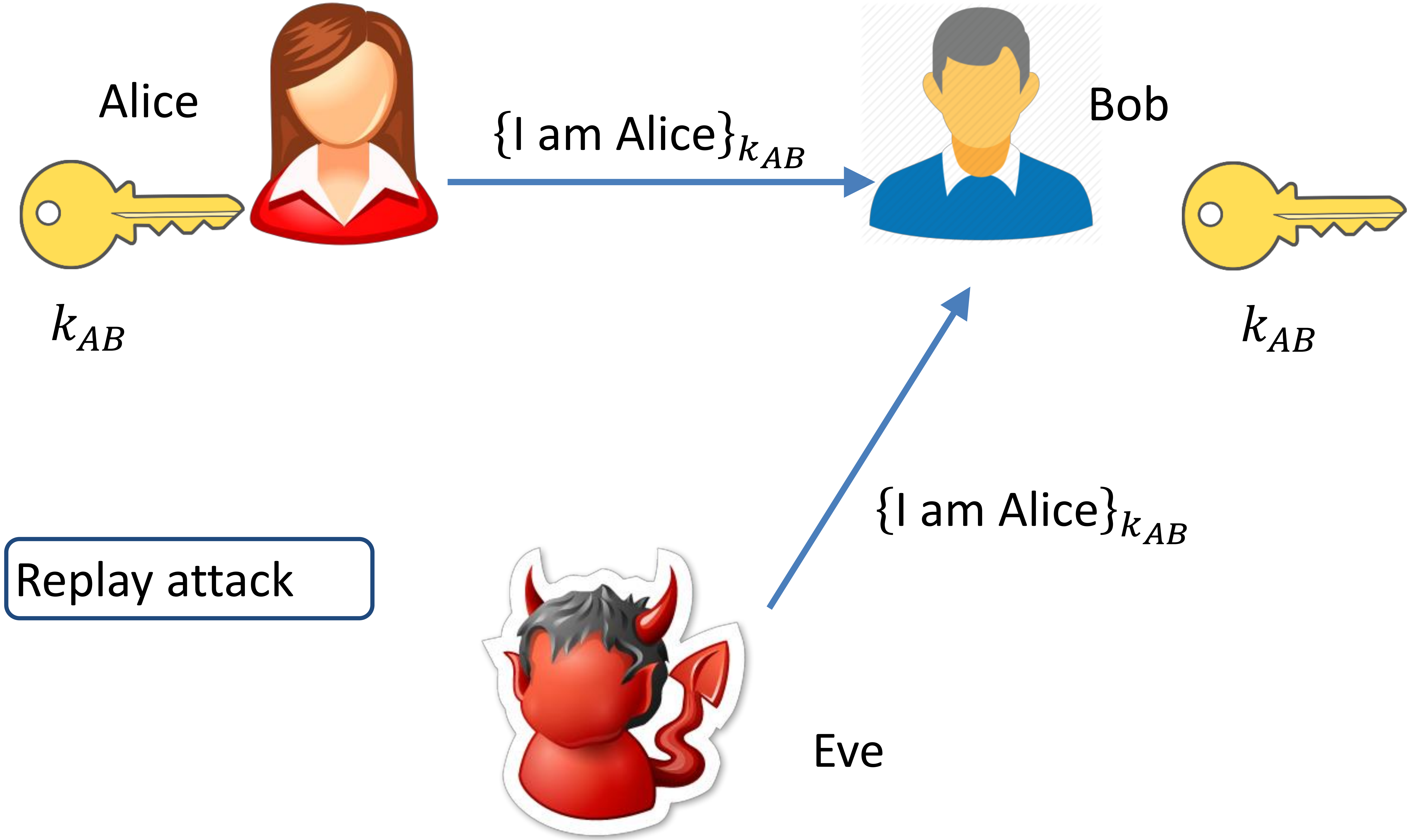
$\{\text{I am Alice}\}_{k_{AB}}$

Bob

k_{AB}

k_{AB}

Attempt #2: use the key



Alice

$\{\text{I am Alice}\}_{k_{AB}}$

Bob

k_{AB}

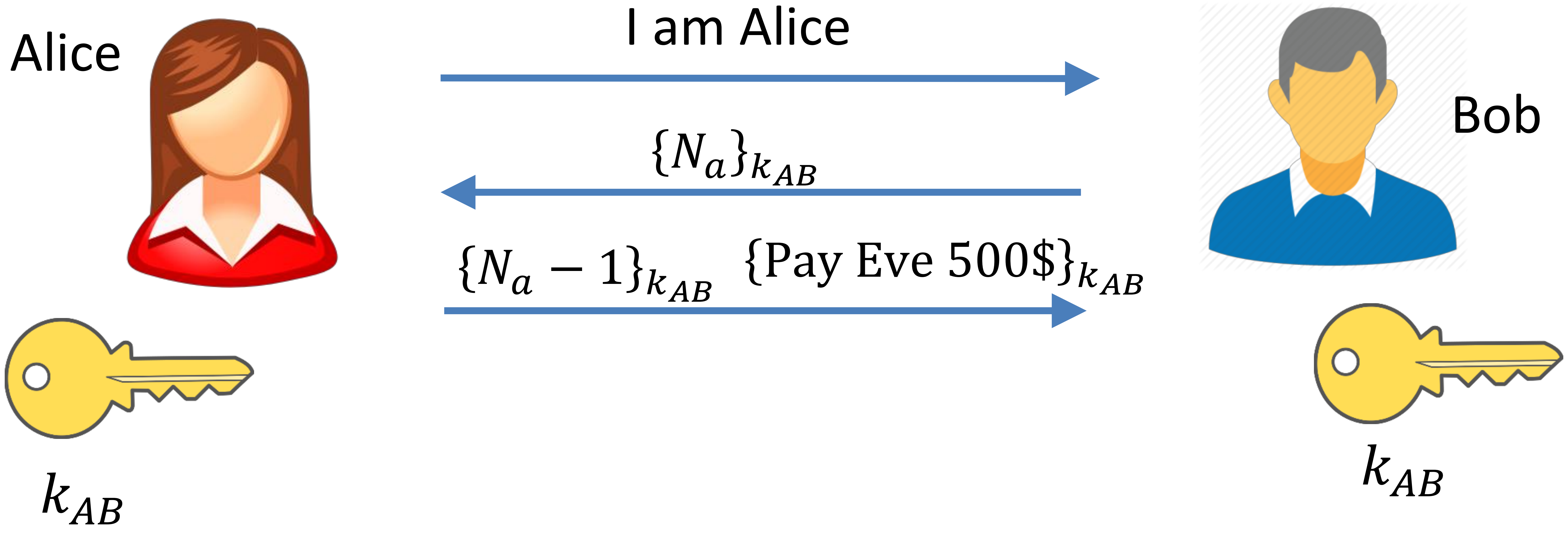
k_{AB}

$\{\text{I am Alice}\}_{k_{AB}}$

Eve

Replay attack

Attempt #3: use nonce



Alice

I am Alice

$\{N_a\}_{k_{AB}}$

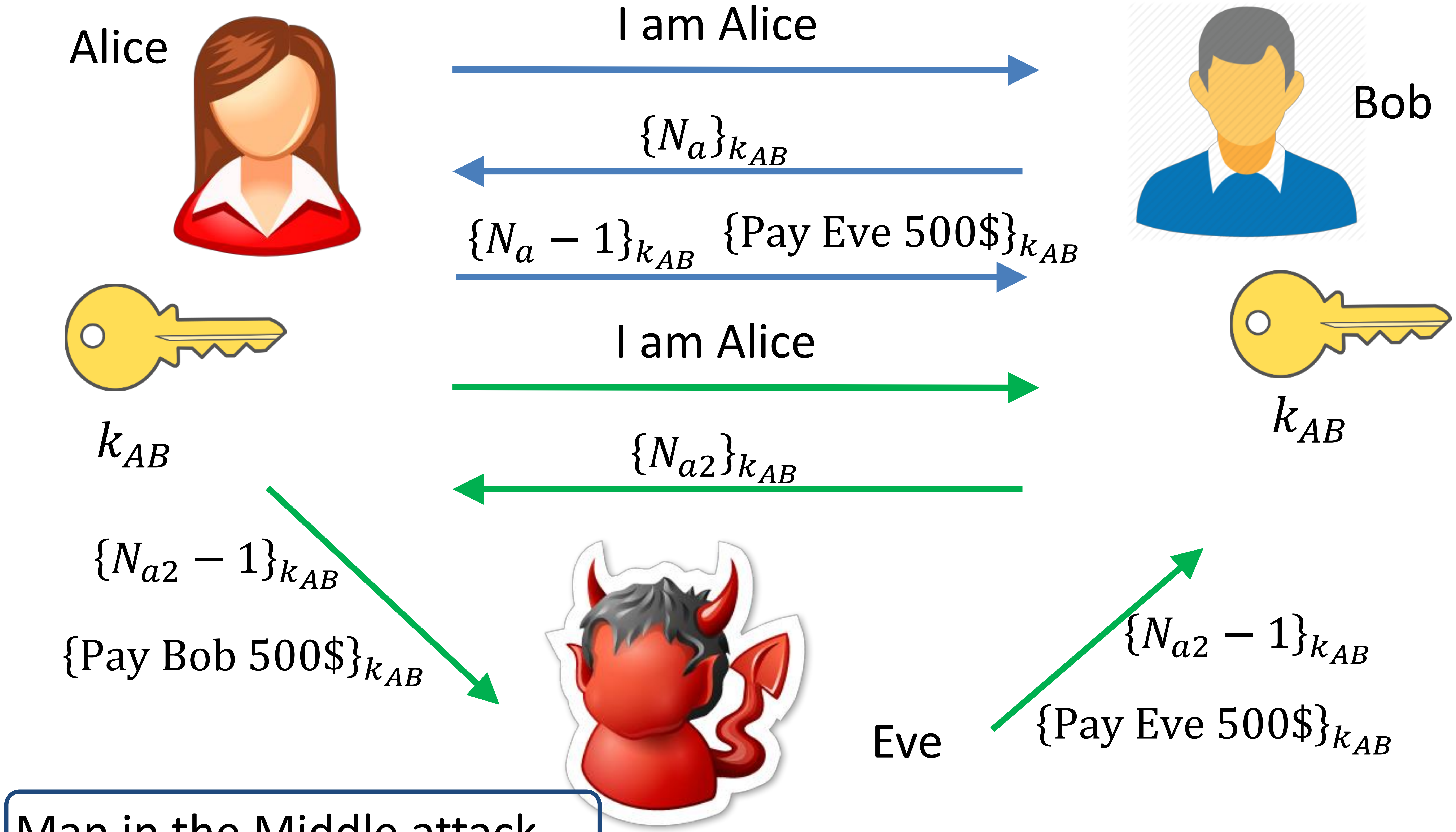
$\{N_a - 1\}_{k_{AB}}$ $\{\text{Pay Eve } 500\$\}_{k_{AB}}$

Bob

k_{AB}

k_{AB}

Attempt #3: use nonce



Alice

Bob

I am Alice

$\{N_a\}_{k_{AB}}$

$\{N_a - 1\}_{k_{AB}}$ $\{\text{Pay Eve 500\$}\}_{k_{AB}}$

I am Alice

$\{N_{a2}\}_{k_{AB}}$

k_{AB}

k_{AB}

$\{N_{a2} - 1\}_{k_{AB}}$

$\{\text{Pay Bob 500\$}\}_{k_{AB}}$

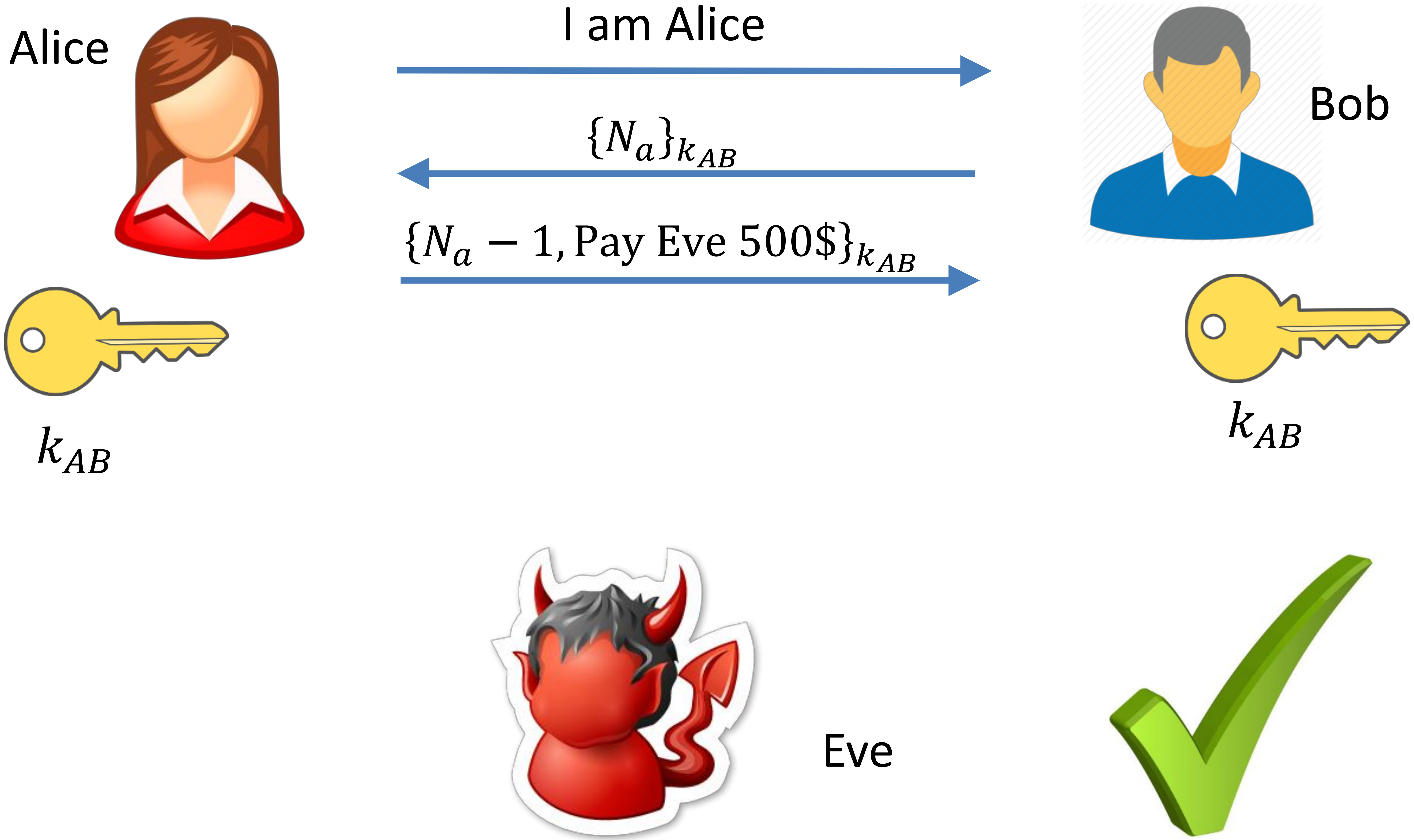
Eve

$\{N_{a2} - 1\}_{k_{AB}}$

$\{\text{Pay Eve 500\$}\}_{k_{AB}}$

Man in the Middle attack

Attempt #4



Alice

I am Alice

$\{N_a\}_{k_{AB}}$

$\{N_a - 1, \text{Pay Eve } 500\$\}_{k_{AB}}$

Bob

k_{AB}

k_{AB}

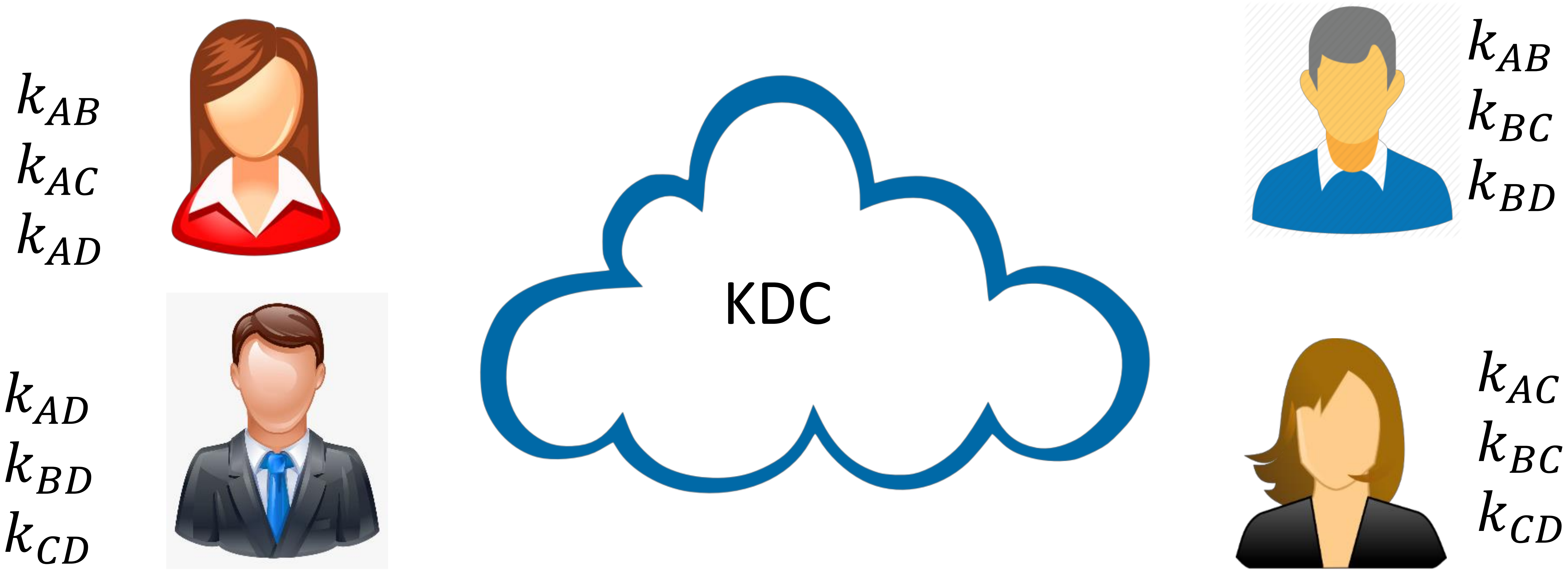
Eve

Key establishment

- The protocol worked because Alice and Bob shared a key
- How do parties agree on a key?
  - Run a key agreement protocol (later in the semester)
  - Use a trusted third party (this lecture)
- Key distribution center (KDC):
  - Shares a key with each entity
  - Single point of failure
  - Reasonable assumption for organizations
  - Not useful for open environments (e.g. the Internet)

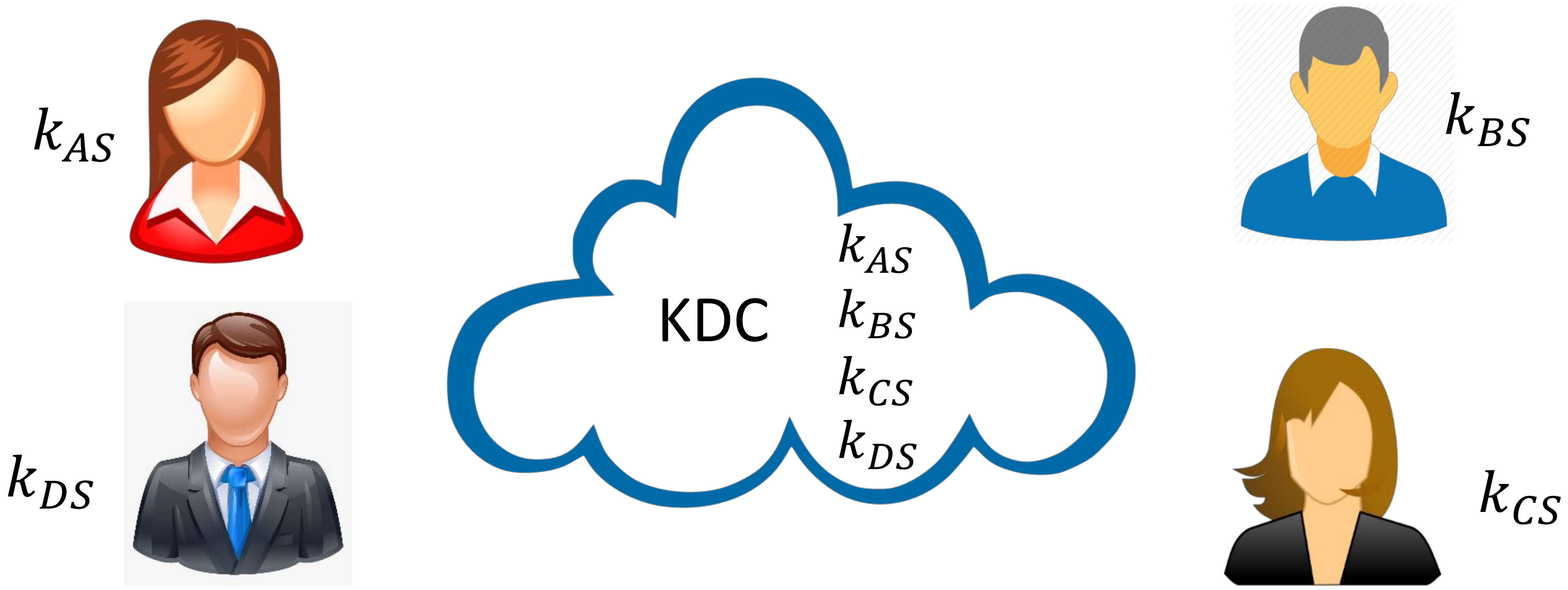
Naïve solution

- KDC generates a key for each pair
- Number of keys $n(n-1)$, number of key pairs $\frac{n(n-1)}{2} = \binom{n}{2}$
- Drawbacks:
  - Quadratic number of keys
  - Adding new users is complex
- May be useful for static small networks

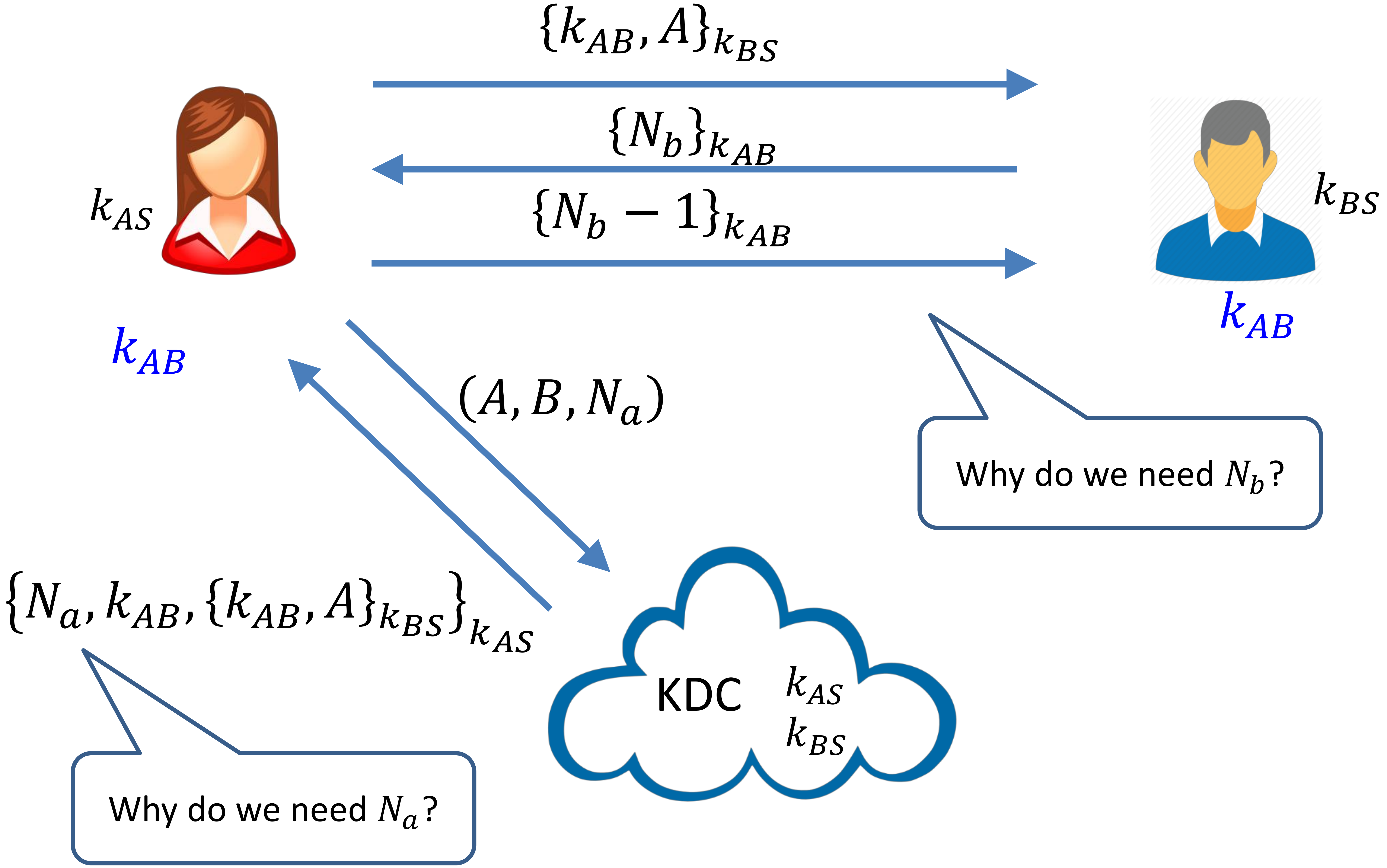


Desire: solution with linear keys

- KDC shares a key with each user
- Number of keys $2n$
- Number of key pairs n
- These are long-term keys
- Alice and Bob establish a fresh session key



Needham-Schroeder Protocol (1978)



$\{k_{AB}, A\}_{k_{BS}}$

$\{N_b\}_{k_{AB}}$

$\{N_b - 1\}_{k_{AB}}$

k_{AS}

k_{AB}

k_{BS}

k_{AB}

(A, B, N_a)

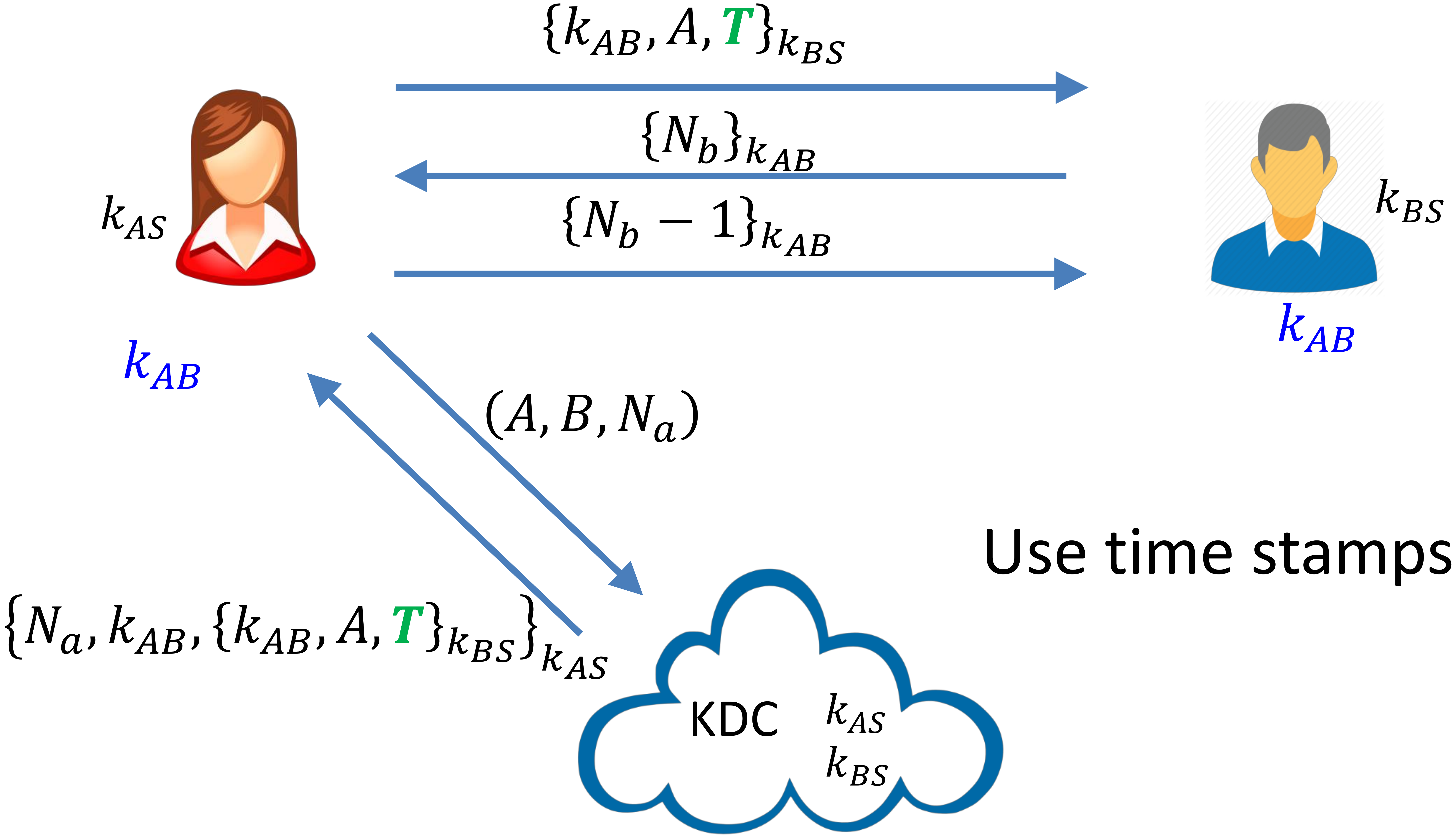
Why do we need N_b?

$\{N_a, k_{AB}, \{k_{AB}, A\}_{k_{BS}}\}_{k_{AS}}$

KDC k_{AS} k_{BS}

Why do we need N_a?

Fixed Needham-Schroeder



$\{k_{AB}, A, \mathbf{T}\}_{k_{BS}}$

$\{N_b\}_{k_{AB}}$

$\{N_b - 1\}_{k_{AB}}$

k_{AS}

k_{AB}

k_{BS}

k_{AB}

(A, B, N_a)

$\{N_a, k_{AB}, \{k_{AB}, A, \mathbf{T}\}_{k_{BS}}\}_{k_{AS}}$

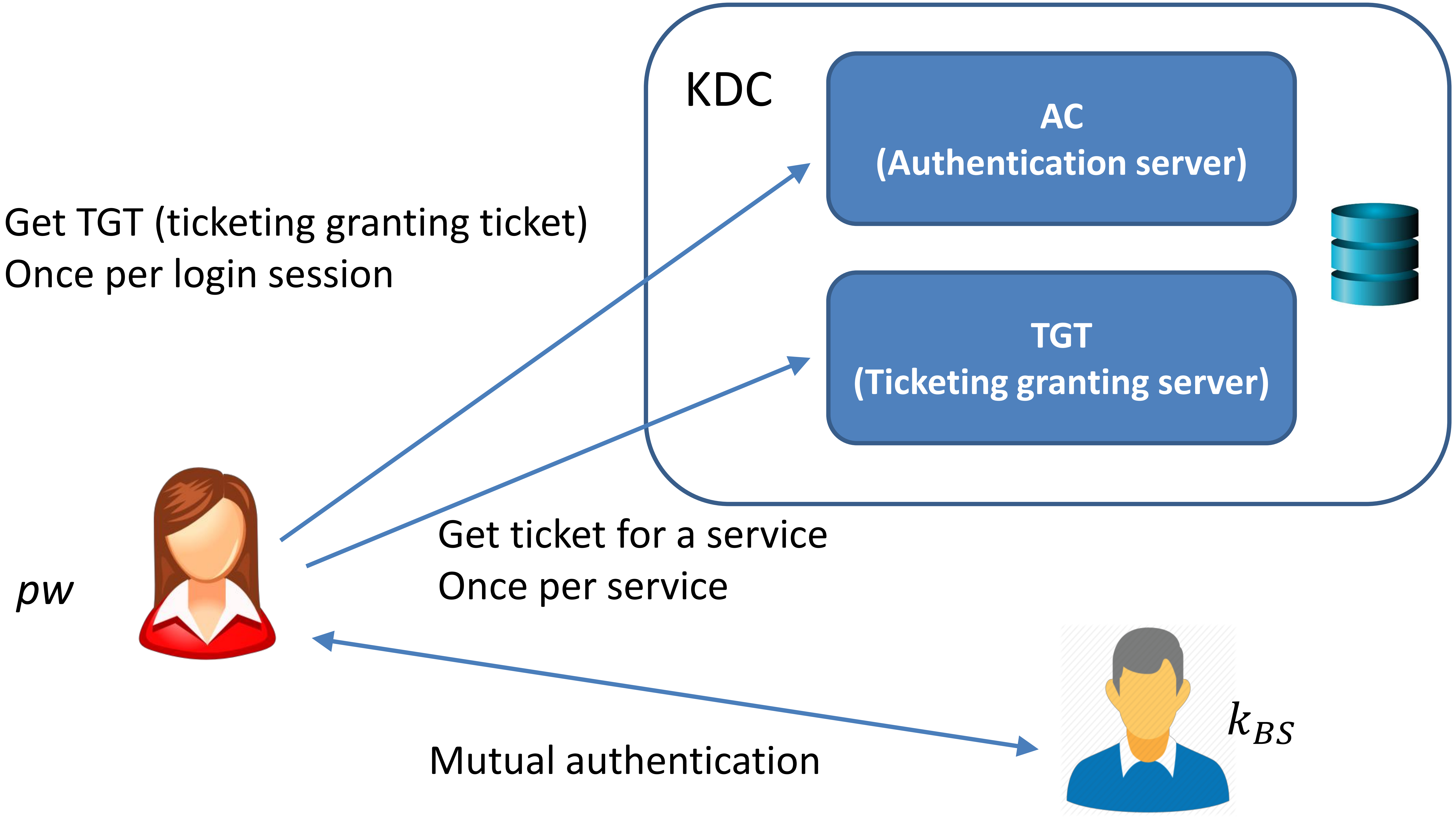
KDC k_{AS} k_{BS}

Use time stamps

Kerberos

- Developed in MIT in the ‘80s
- Based on Needham-Schroeder
  - Versions 1-3 not published
  - Version 4 not secure
  - Version 5 published in 1993
- Widely used nowadays:
  - The basis of Microsoft’s active directory
  - Many Unix versions

Kerberos



KDC

AC
(Authentication server)

TGT
(Ticketing granting server)

Get TGT (ticketing granting ticket)
Once per login session

Get ticket for a service
Once per service

pw

k_{BS}

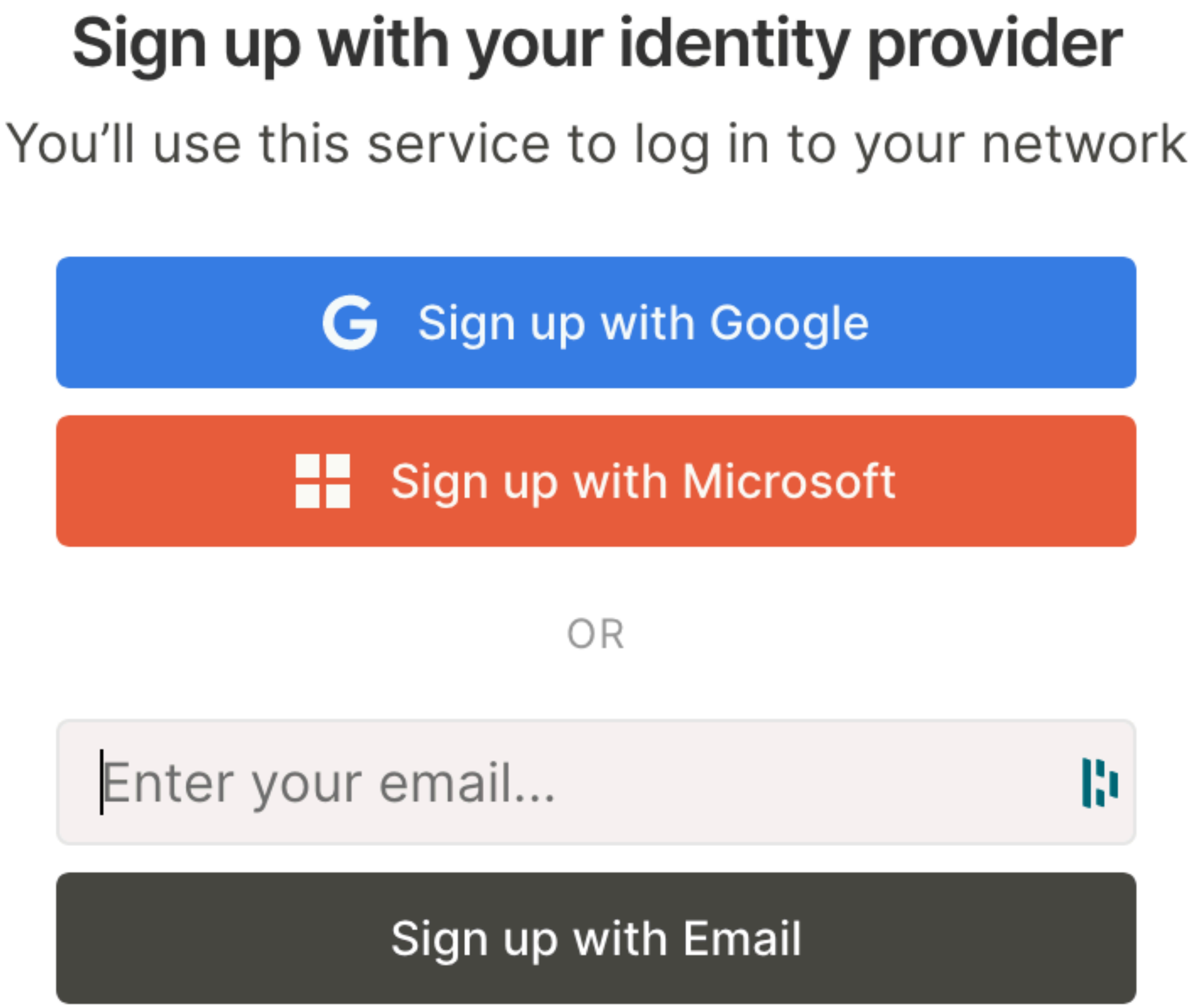
Mutual authentication

Kerberos

- Passwords are not sent over the network
- Alice's key k_{AS} is a hash of her password

- Kerberos weaknesses:
  - KDC is a single point of failure
  - DoS the KDC and the network ceases to function
  - Compromise the KDC leads to network-wide compromise
  - Time synchronization is a very hard problem

“Single Sign on”



Sign up with your identity provider

You’ll use this service to log in to your network

Sign up with Google

Sign up with Microsoft

OR

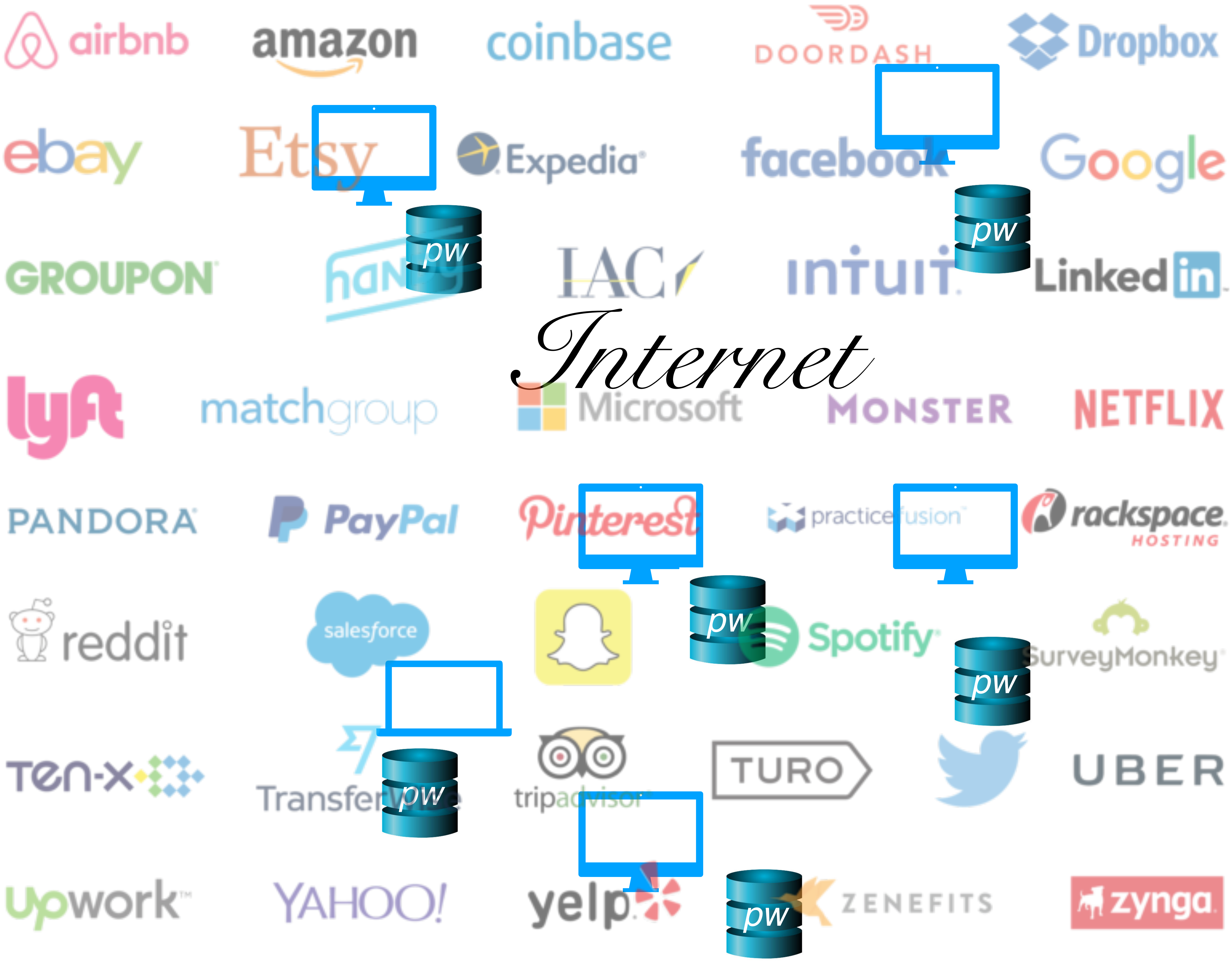
Enter your email...

Sign up with Email

Same problem as before

Alice

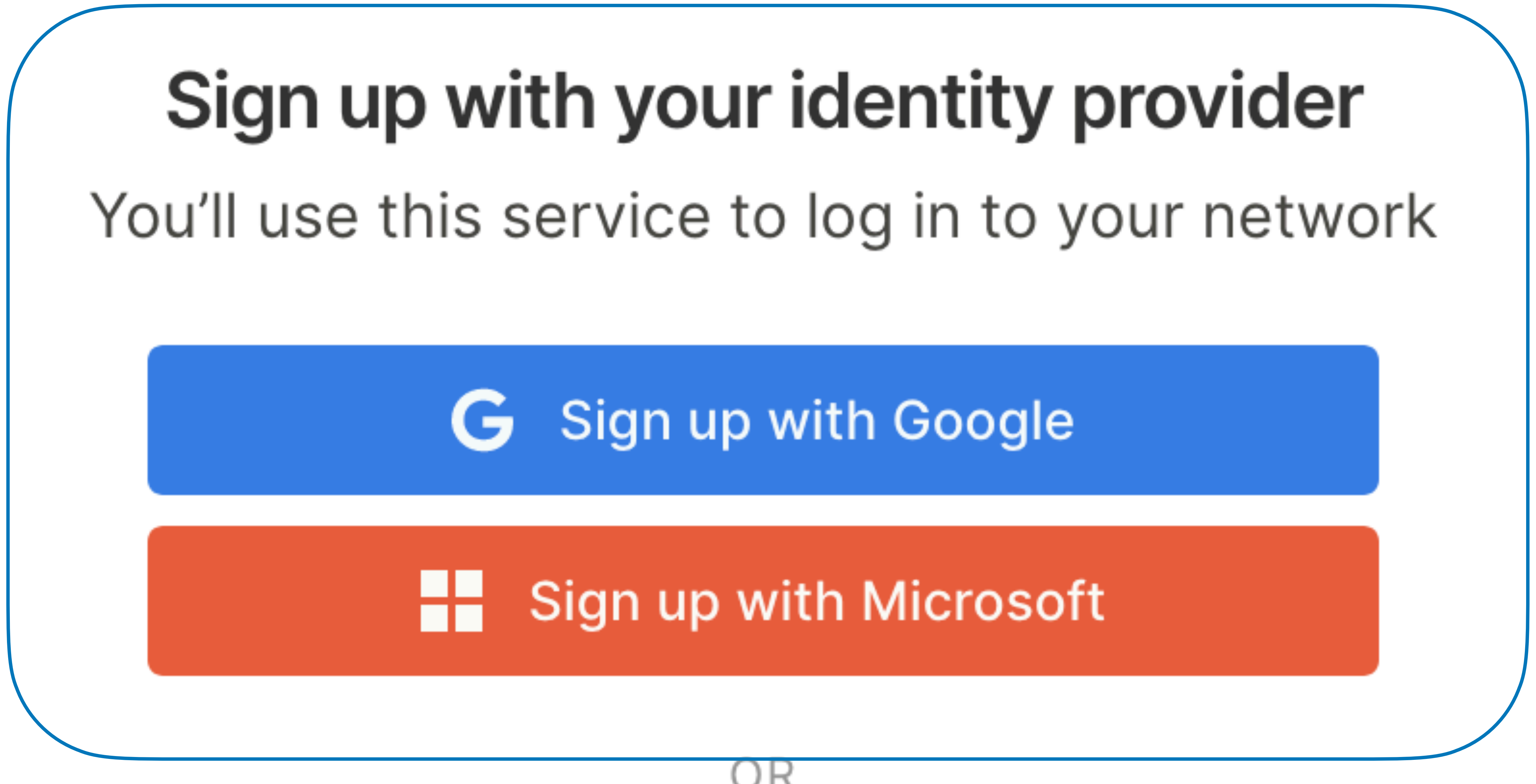
pw



“Single Sign on”

Alice

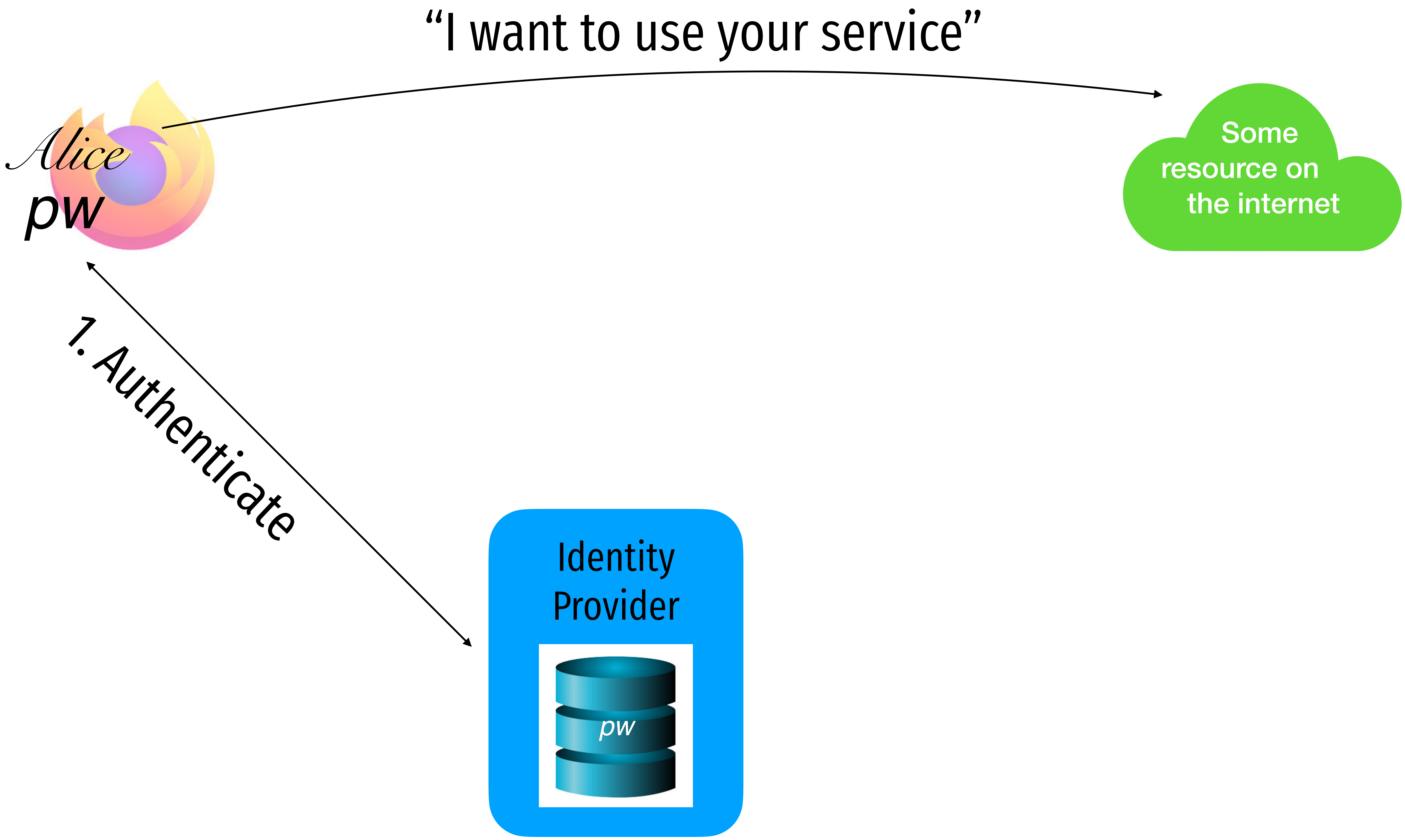
pw



Sign up with your identity provider

You’ll use this service to log in to your network

Oauth



“I want to use your service”

Alice

pw

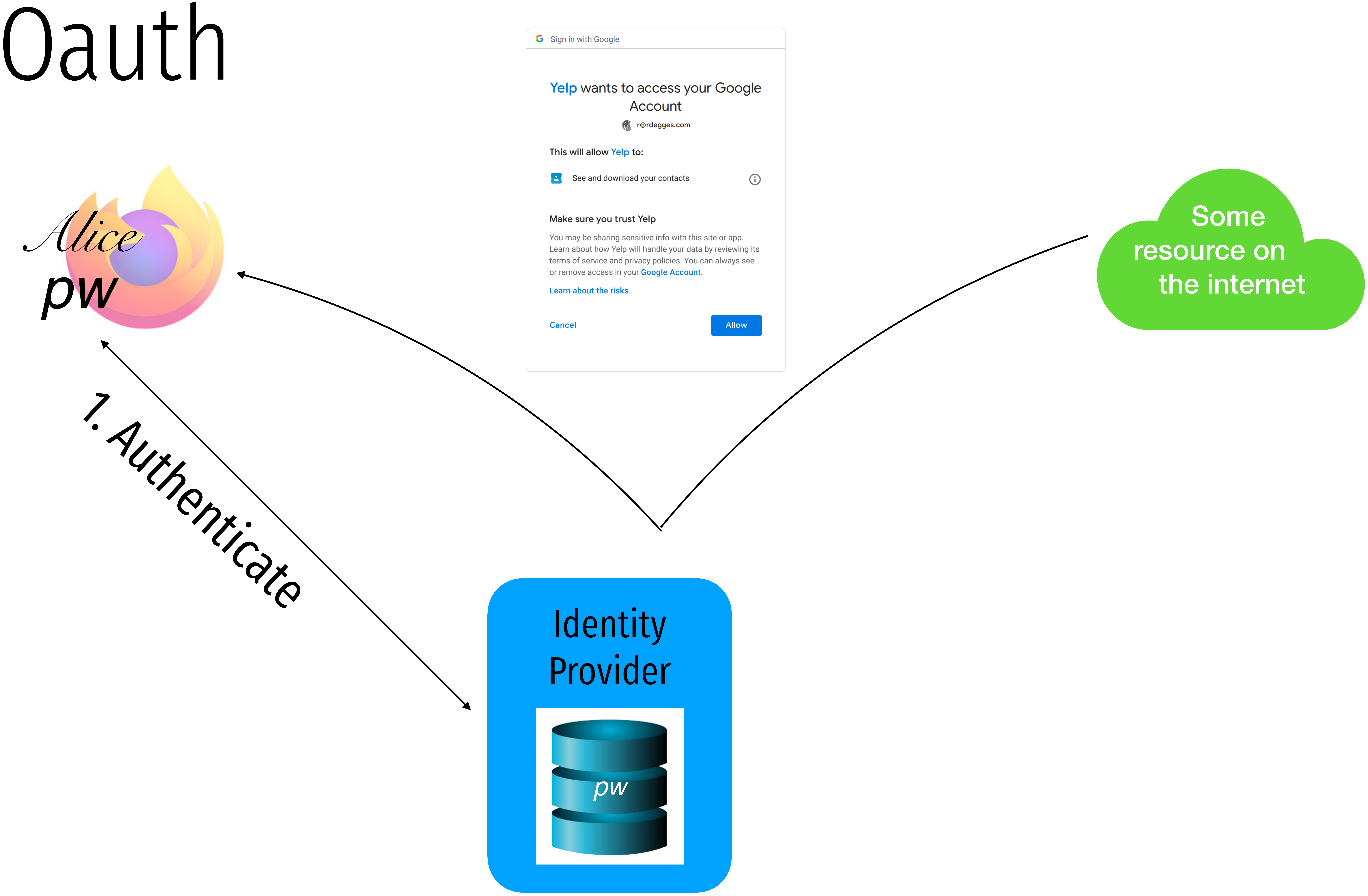
1. Authenticate

Some resource on the internet

Identity Provider

pw

Oauth



Sign in with Google

Yelp wants to access your Google Account

r@rdegges.com

This will allow Yelp to:

See and download your contacts

Make sure you trust Yelp

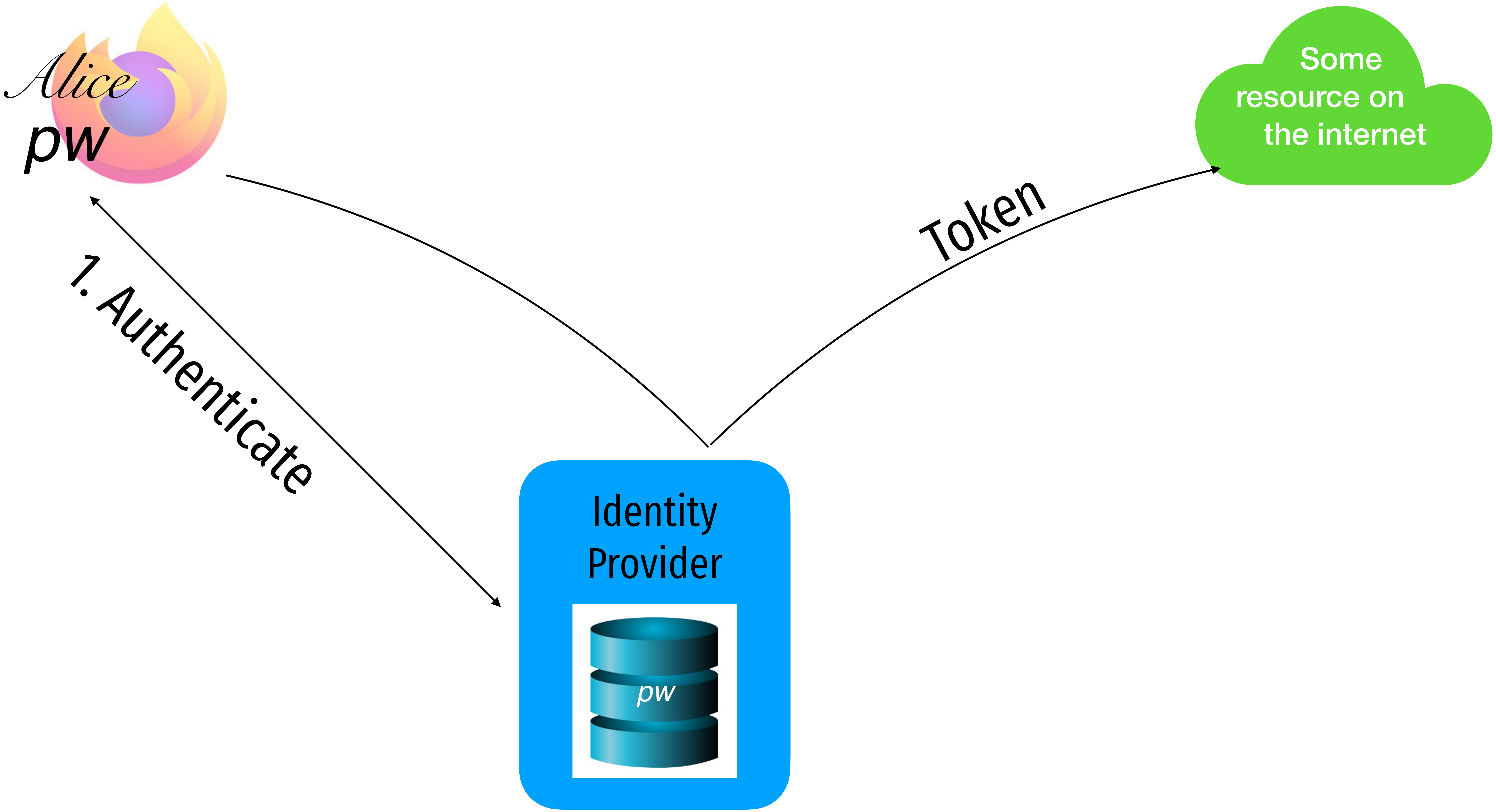
You may be sharing sensitive info with this site or app. Learn about how Yelp will handle your data by reviewing its terms of service and privacy policies. You can always see or remove access in your Google Account.

Learn about the risks

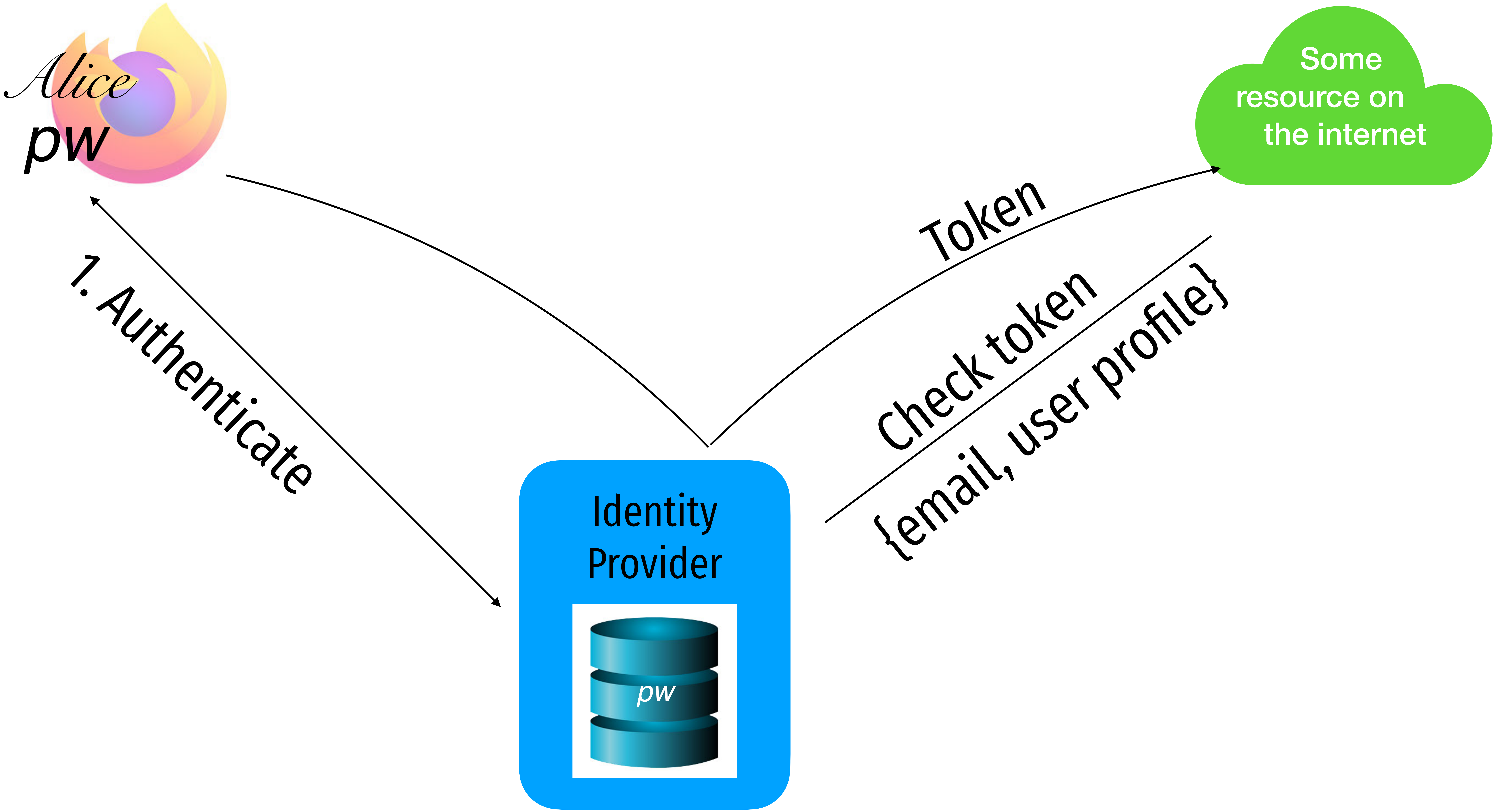
Cancel

Allow

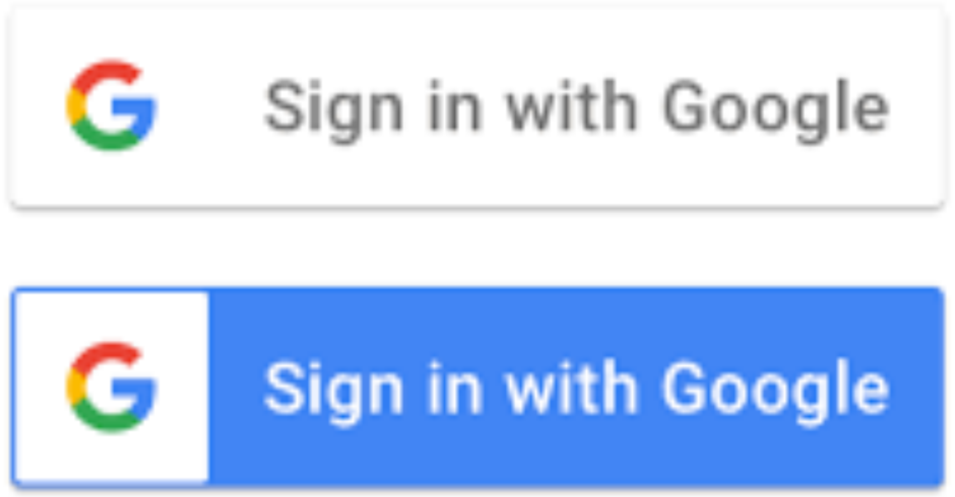
Oauth


Alice
pw
1. Authenticate
Token
Some resource on the internet
Identity Provider
pw

Oauth


Alice
pw
1. Authenticate
Token
Check token
{email, user profile}
Some resource on the internet
Identity Provider
pw

Attacks against “Login with…” services



Use Sign in with Apple on your Apple device

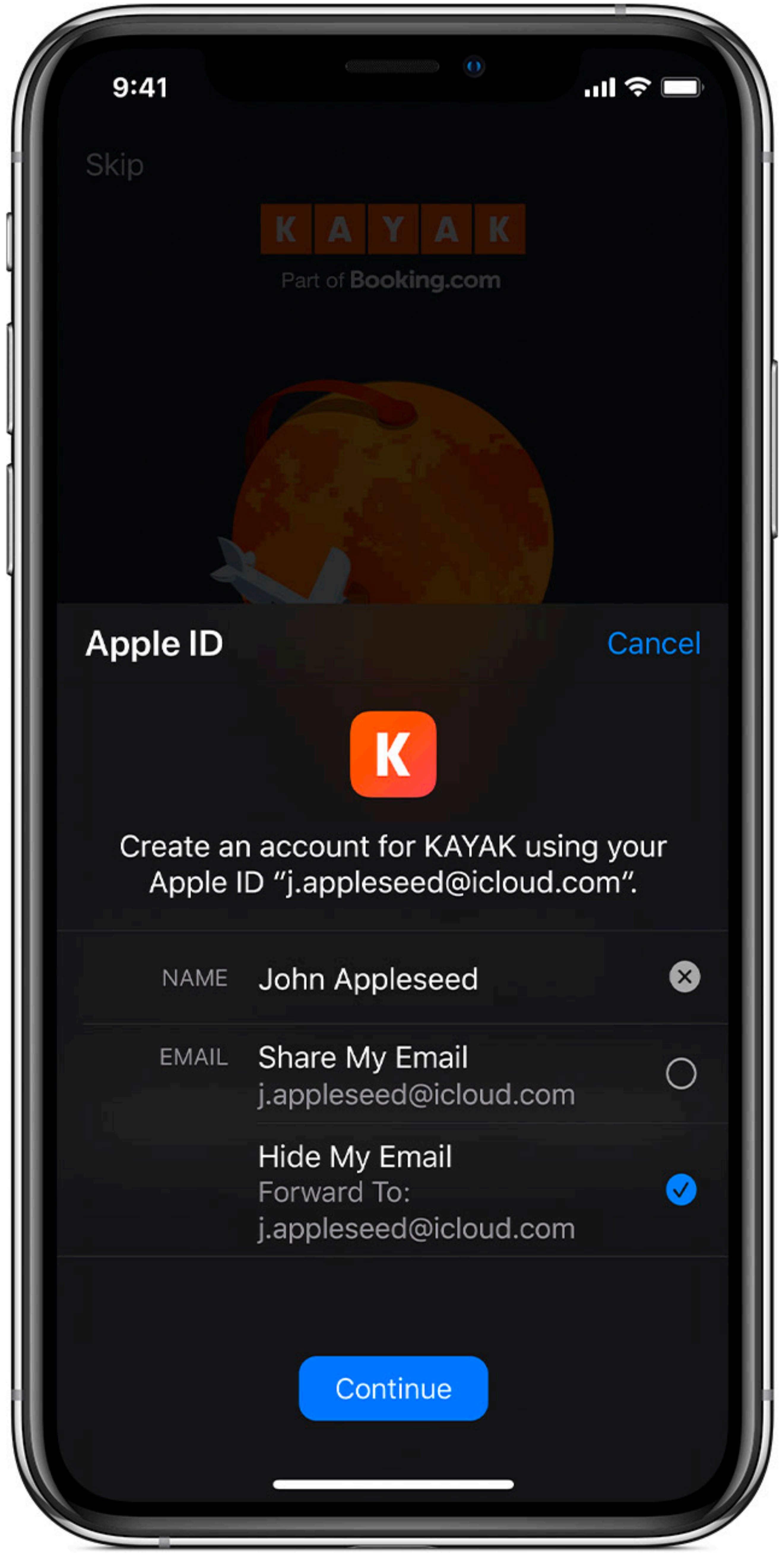
Using Sign in with Apple is quick and easy on any Apple device with the latest software. Make sure you're signed in with your Apple ID on your device.

1. Tap the Sign in with Apple button on the participating app or website.

   If the app or site has not requested any information to set up your account, check that your Apple ID is correct and go to Step 4.

   If you're asked to provide your name and email address, Sign in with Apple automatically fills in the information from your Apple ID. You can edit your name if you like and choose Share My Email or Hide My Email.

   Tap Continue and confirm with a quick Face ID, Touch ID, or device passcode to sign in. If you don't have Face ID, Touch ID, or a passcode set up, enter your Apple ID password.



Create an account for KAYAK using your Apple ID “j.appleseed@icloud.com”.

NAME John Appleseed

EMAIL Share My Email j.appleseed@icloud.com ○

Hide My Email Forward To: j.appleseed@icloud.com ✓

Authentication:

Authorization

After Authenticating a subject, what next?

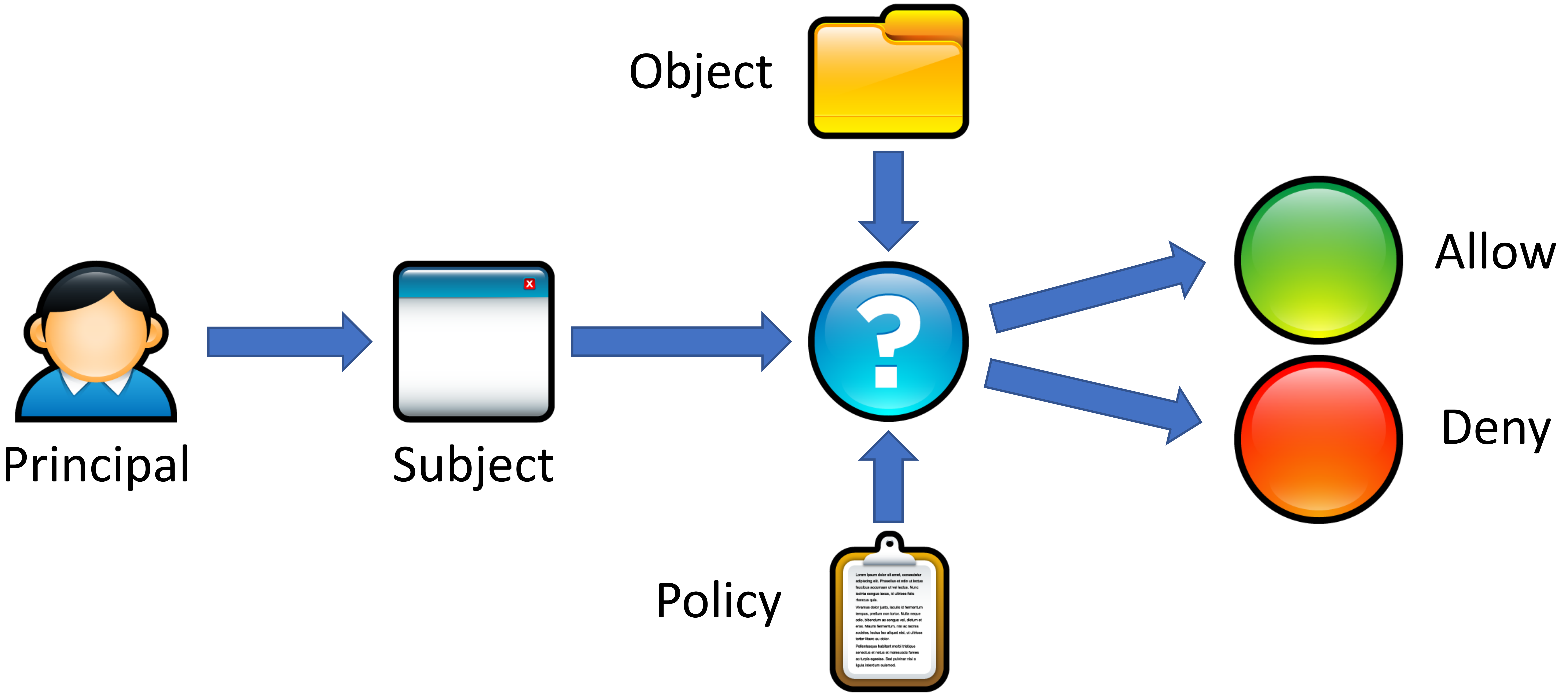
Access Control

- Policy specifying how entities can interact with resources
  - i.e., Who can access what?
  - Requires authentication and authorization
- Access control primitives

Principal	User of a system	
Subject	Entity that acts on behalf of principals	Software program
Object	Resource acted upon by subjects	Files Sockets Devices OS APIs

Access Control Check

- Given an access request from a subject, on behalf of a principal, for an object, return an access control decision based on the policy



Access Control Models

- Discretionary Access Control (DAC)
  - The kind of access control you are familiar with
  - Access rights propagate and may be changed at subject’s discretion

Access Control Models

- Discretionary Access Control (DAC)
  - The kind of access control you are familiar with
  - Access rights propagate and may be changed at subject's discretion

- Mandatory Access Control (MAC)
  - Access of subjects to objects is based on a system-wide policy
  - Denies users full control over resources they create

Sources

1. Many slides courtesy of Wil Robertson: https://wkr.io
2. Many slides courtesy of Ran Cohen