Nothing To Hide

Privacy-Preserving Cryptographic Authentication In Practice

Who Am I

Abdullah Joseph

@MalwareCheese

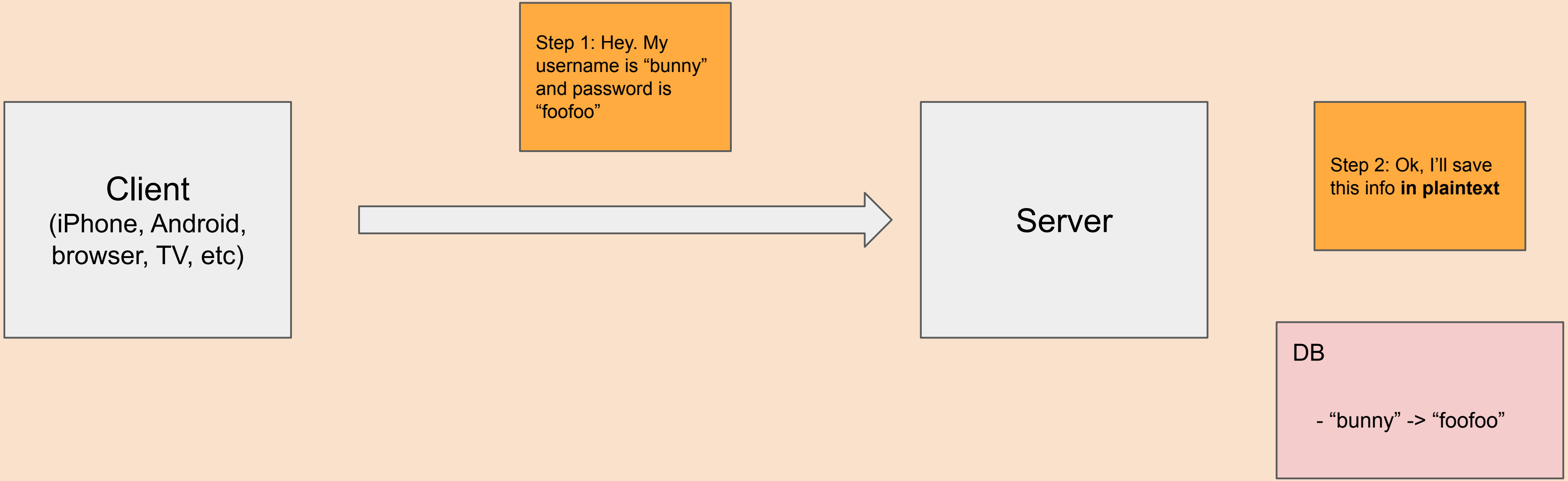
Software Engineer ~12 years

Security Research ~8 years

Currently working in the adtech industry as a security researcher

Typical Registration/Login Implementation

Try #1 (with plaintext passwords): Registration



Step 1: Hey. My username is “bunny” and password is “foofoo”

Client (iPhone, Android, browser, TV, etc)

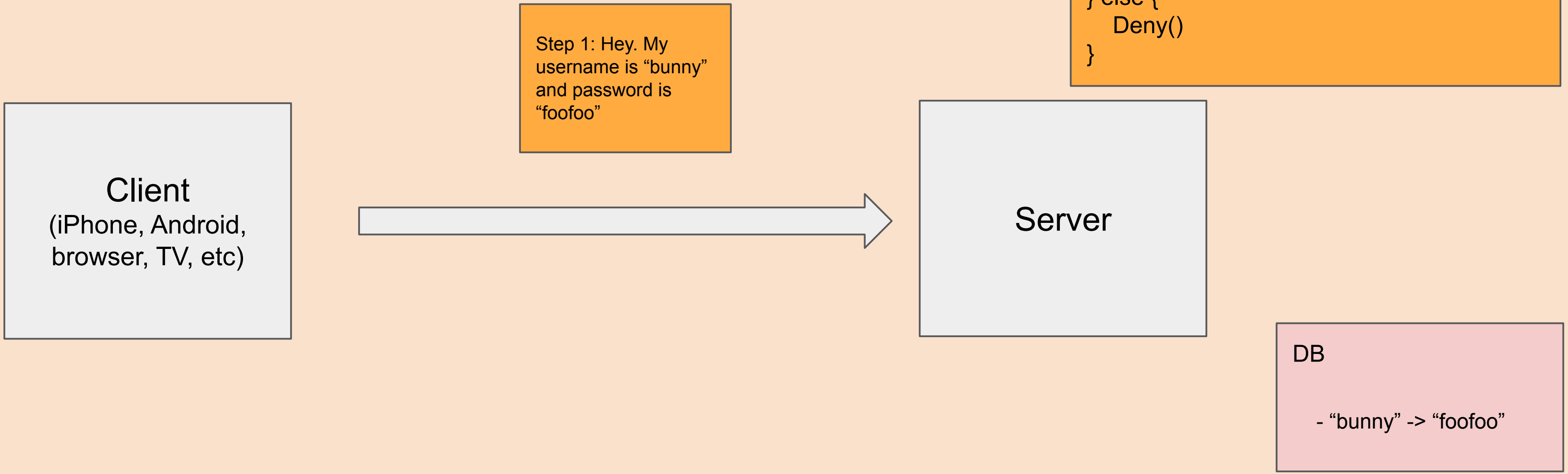
Server

Step 2: Ok, I’ll save this info **in plaintext**

DB

- “bunny” -> “foofoo”

Try #1 (with plaintext passwords): Login



Client
(iPhone, Android, browser, TV, etc)

Step 1: Hey. My username is “bunny” and password is “foofoo”

Server

Step 2: Lemme check the DB.

```
If inputPass == savedPass {
  Allow();
} else {
   Deny()
}
```

DB

- “bunny” -> “foofoo”

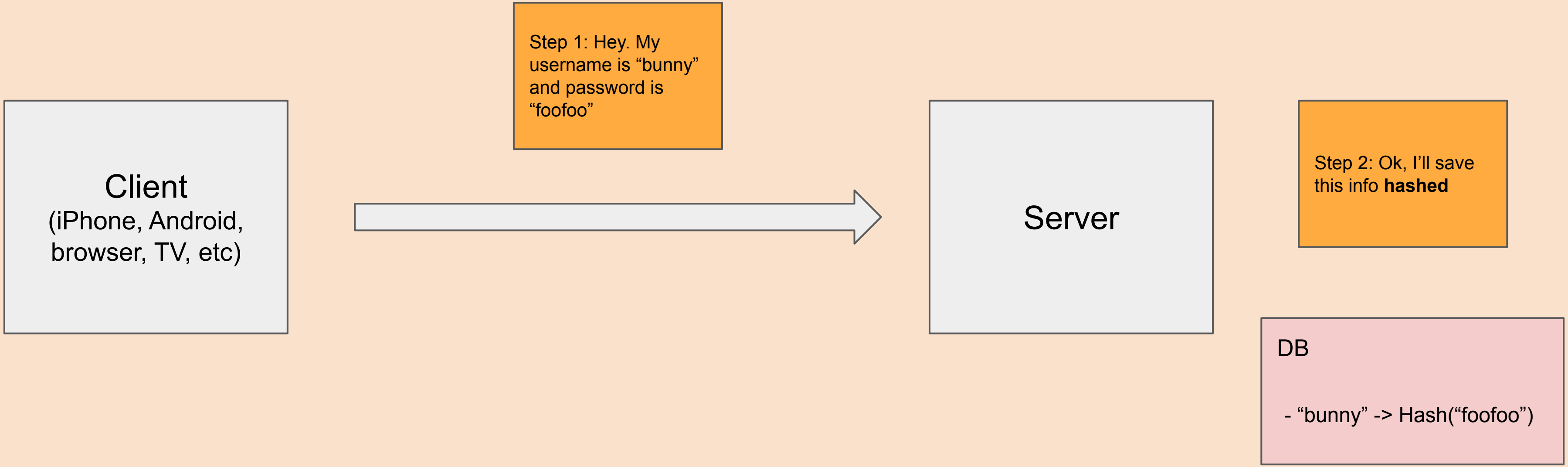
Try #1 (with plaintext passwords): Issues

- Server saves client’s password in plaintext
- Client sends their password in plaintext

Try #1 (with plaintext passwords): Solutions

- Server saves client’s password in plaintext
    - Solution: Server can maybe hash it before saving it?
- Client sends their password in plaintext
    - Solution: ???

Try #2 (server-side hashing): Registration



Step 1: Hey. My username is “bunny” and password is “foofoo”

Client
(iPhone, Android, browser, TV, etc)

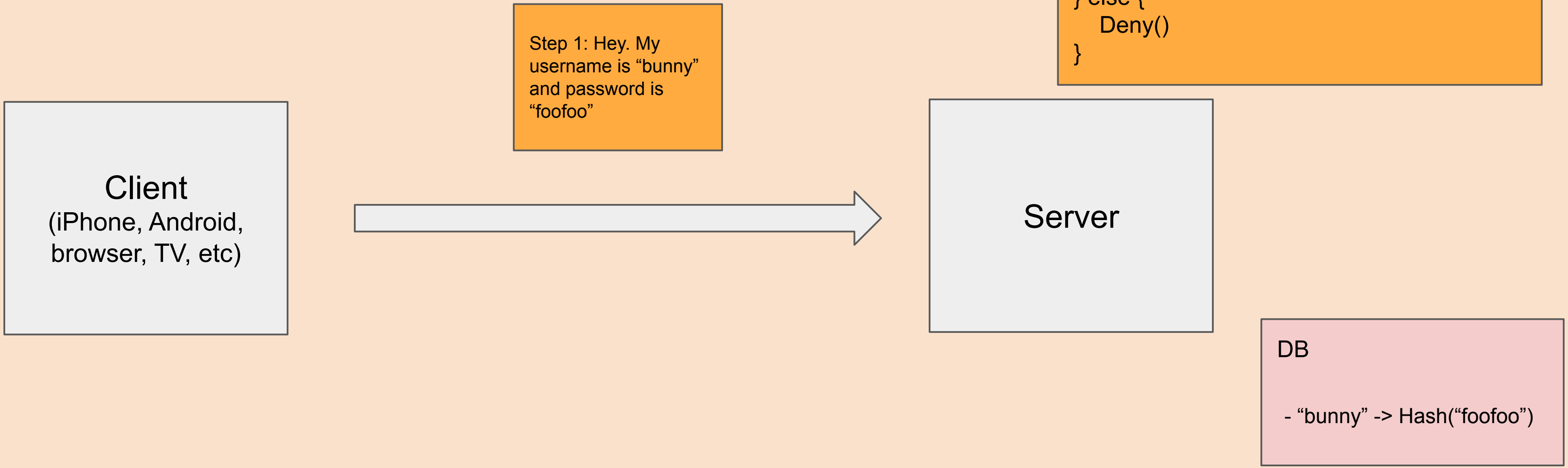
Server

Step 2: Ok, I’ll save this info **hashed**

DB

- “bunny” -> Hash(“foofoo”)

Try #2 (server-side hashing): Login



Client
(iPhone, Android, browser, TV, etc)

Step 1: Hey. My username is “bunny” and password is “foofoo”

Server

Step 2: Lemme check the DB.

```
If inputPass == Hash(savedPass) {
  Allow();
} else {
   Deny()
}
```

DB

- “bunny” -> Hash(“foofoo”)

Try #2 (server-side hashing): Issues

- Server saves client’s password in plaintext
- Client sends their password in plaintext

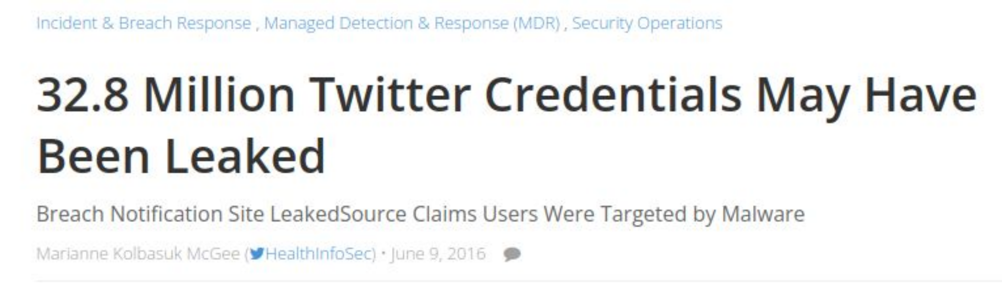
Try #2 (server-side hashing): Issues

- **RESOLVED** Server saves client’s password in plaintext
- **UNRESOLVED** Client sends their password in plaintext

Try #2 (server-side hashing): Issues

- **RESOLVED** Server saves client’s password in plaintext
- **UNRESOLVED** Client sends their password in plaintext

Why is this even an issue?

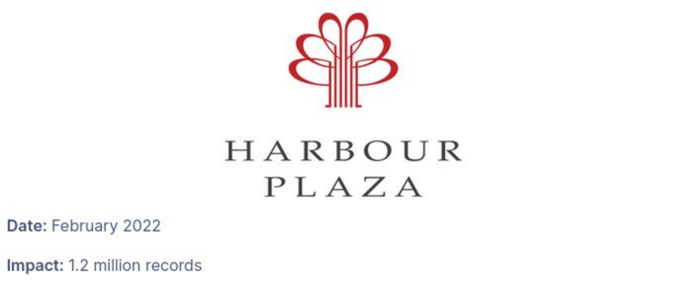


Incident & Breach Response , Managed Detection & Response (MDR) , Security Operations

32.8 Million Twitter Credentials May Have Been Leaked

Breach Notification Site LeakedSource Claims Users Were Targeted by Malware

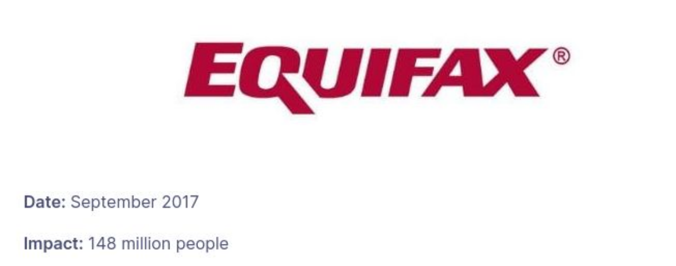
Marianne Kolbasuk McGee (HealthInfoSec) • June 9, 2016



HARBOUR PLAZA

Date: February 2022

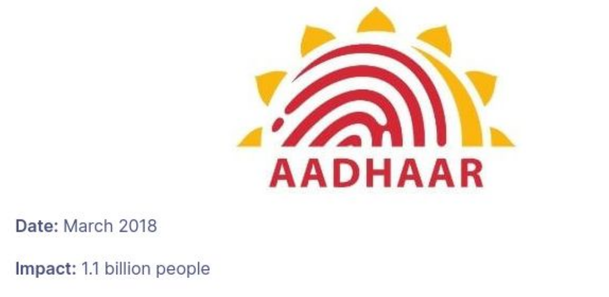
Impact: 1.2 million records



EQUIFAX®

Date: September 2017

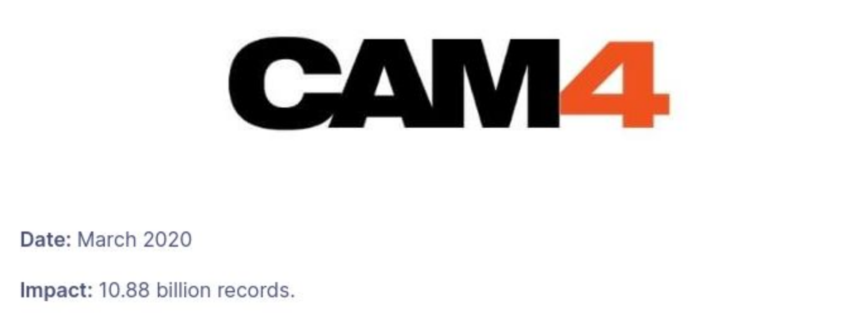
Impact: 148 million people



AADHAAR

Date: March 2018

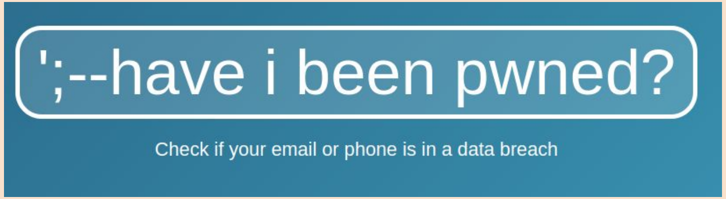
Impact: 1.1 billion people



CAM4

Date: March 2020

Impact: 10.88 billion records.



';--have i been pwned?

Check if your email or phone is in a data breach

The Problem with Typical Registrations

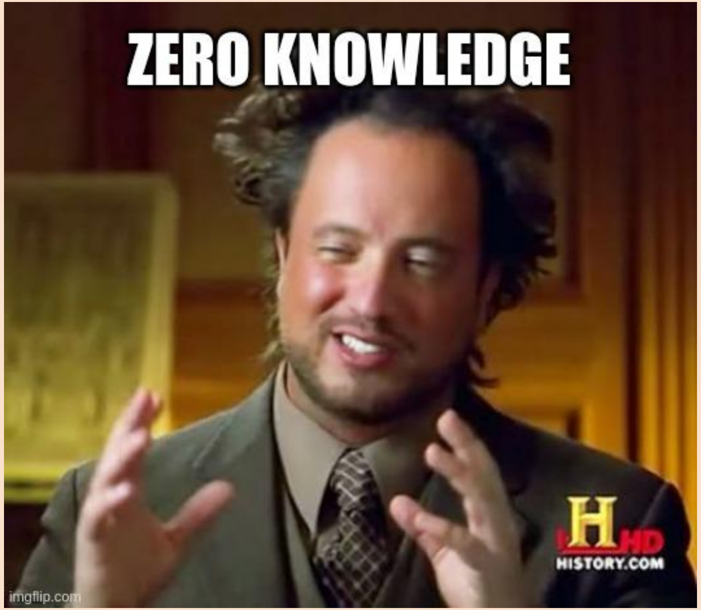
- Usernames and passwords are **always** sent in plaintext to the server
- *Hopefully*, the server will hash it before saving it

Most probably, they won’t

Demo: Login to HN

The Solution? **Half-life 3 And Cryptography**

Has Science Gone Too Far?
IS THIS IMAGE REAL OR FAKE?
REAL
FAKE
93% fail this quiz. would you? officialiqquiz.com



But mostly cryptography…

Let’s talk about OPRFs

(Oblivious Pseudorandom Functions)

Alice

Bob

Alice

Bob

Alice

Bob

Alice

OPRFs

Bob

They wanna compute a number **together** whereas only **one person** knows the result

Alice

OPRFs

Bob

As opposed to something like Diffie-Hellman, where **both** parties compute a number and **both** know the result

OPRFs

Alice

Bob

As opposed to something like Diffie-Hellman, where **both** parties compute a number and **both** know the result

Don’t tell this to a real cryptographer. They’ll chop off your legs.

OPRF Computation Overview

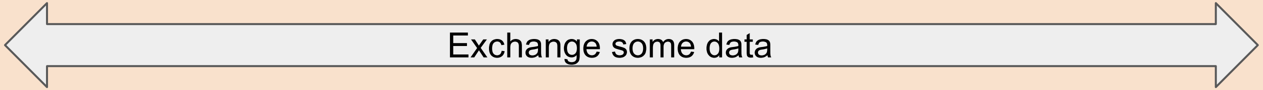
Alice

Bob

Step 1 alice_secret

bob_secret

Step 2



Exchange some data

Step 3 **oprf** = f(alice_secret, bob_secret)

Does not know result of product, but aids in the computation using his bob_secret

Alice

Has alice_secret

OPRF Computation Process

Step 0: Parameter Definitions
Step 1: Blinding
Step 2: Evaluation
Step 3: Unblinding

Bob

Has bob_secret

Alice

OPRF Computation Process

Step 0: Parameter Definitions

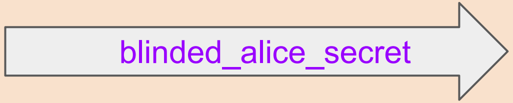
Step 1: Blinding

Step 2: Evaluation

Step 3: Unblinding

Bob

blinded_alice_secret = Blind(alice_secret)



blinded_alice_secret

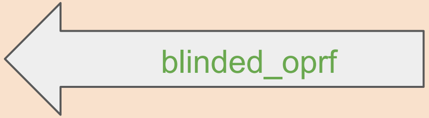
Alice

OPRF Computation Process

Step 0: Parameter Definitions
Step 1: Blinding
Step 2: Evaluation
Step 3: Unblinding

Bob

```
blinded_oprf =
  Evaluate(
     blinded_alice_secret,
     bob_secret,
  )
```



blinded_oprf

Alice

OPRF Computation Process

Step 0: Parameter Definitions
Step 1: Blinding
Step 2: Evaluation
Step 3: Unblinding (Finalization)

Bob

oprf = Unblind(blinded_oprf)

OPRF Computation Overview

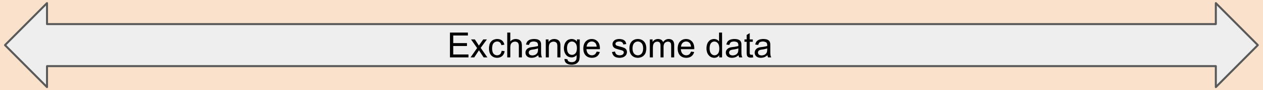
Alice

Bob

Step 1 alice_secret

bob_secret

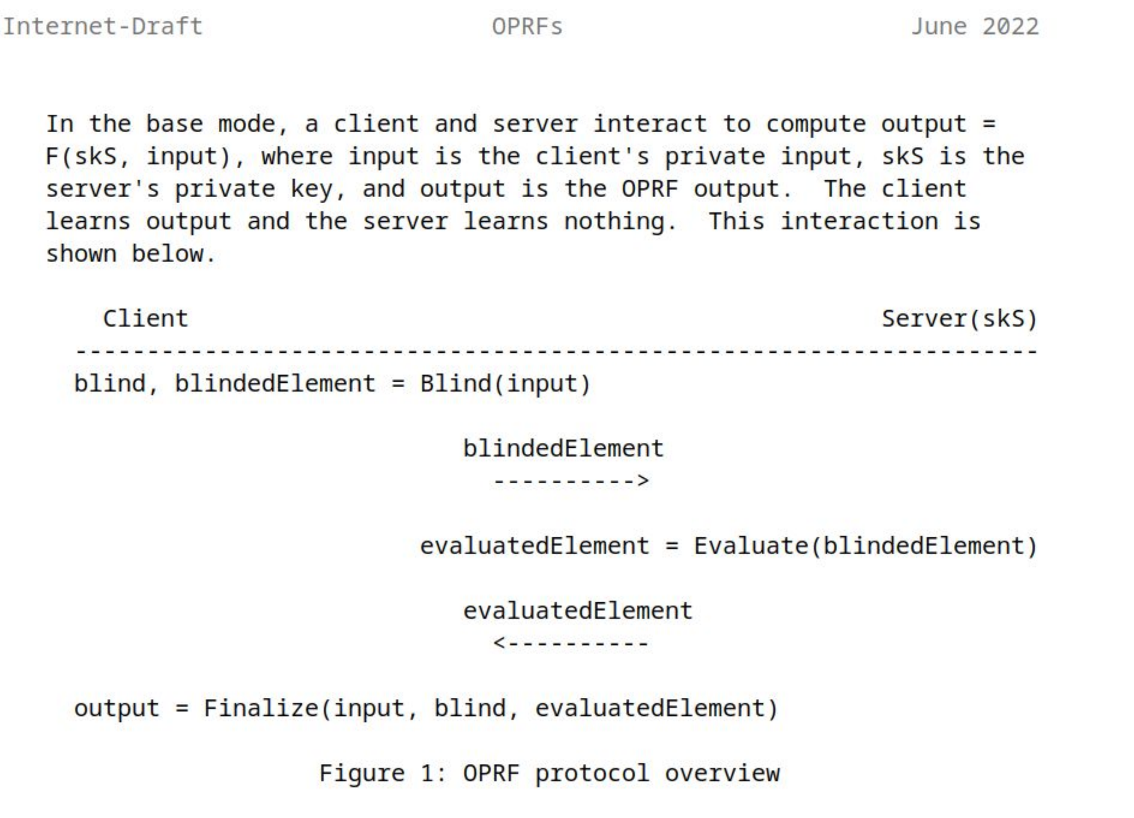
Step 2



Exchange some data

Step 3 **oprf** = f(alice_secret, bob_secret)

Does not know result of product, but aids in the computation using his bob_secret



Internet-Draft OPRFs June 2022

In the base mode, a client and server interact to compute output = F(skS, input), where input is the client's private input, skS is the server's private key, and output is the OPRF output. The client learns output and the server learns nothing. This interaction is shown below.

```
    Client                                                  Server(skS)
  -------------------------------------------------------------------
  blind, blindedElement = Blind(input)

                             blindedElement
                               ---------->

                        evaluatedElement = Evaluate(blindedElement)

                             evaluatedElement
                               <----------

  output = Finalize(input, blind, evaluatedElement)
```

Figure 1: OPRF protocol overview

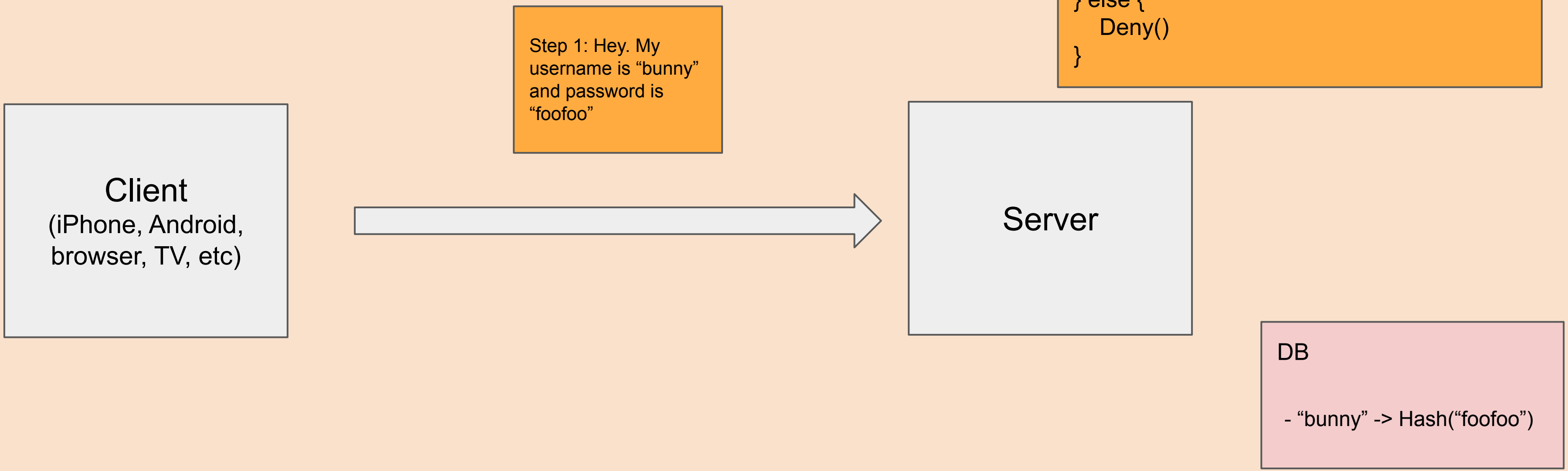
https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-voprf-10#section-3

Let’s revisit registrations/logins

Try #2 (server-side hashing): Login

Step 2: Lemme check the DB.

```
If inputPass == hash(savedPass) {
  Allow();
} else {
   Deny()
}
```



Step 1: Hey. My username is “bunny” and password is “foofoo”

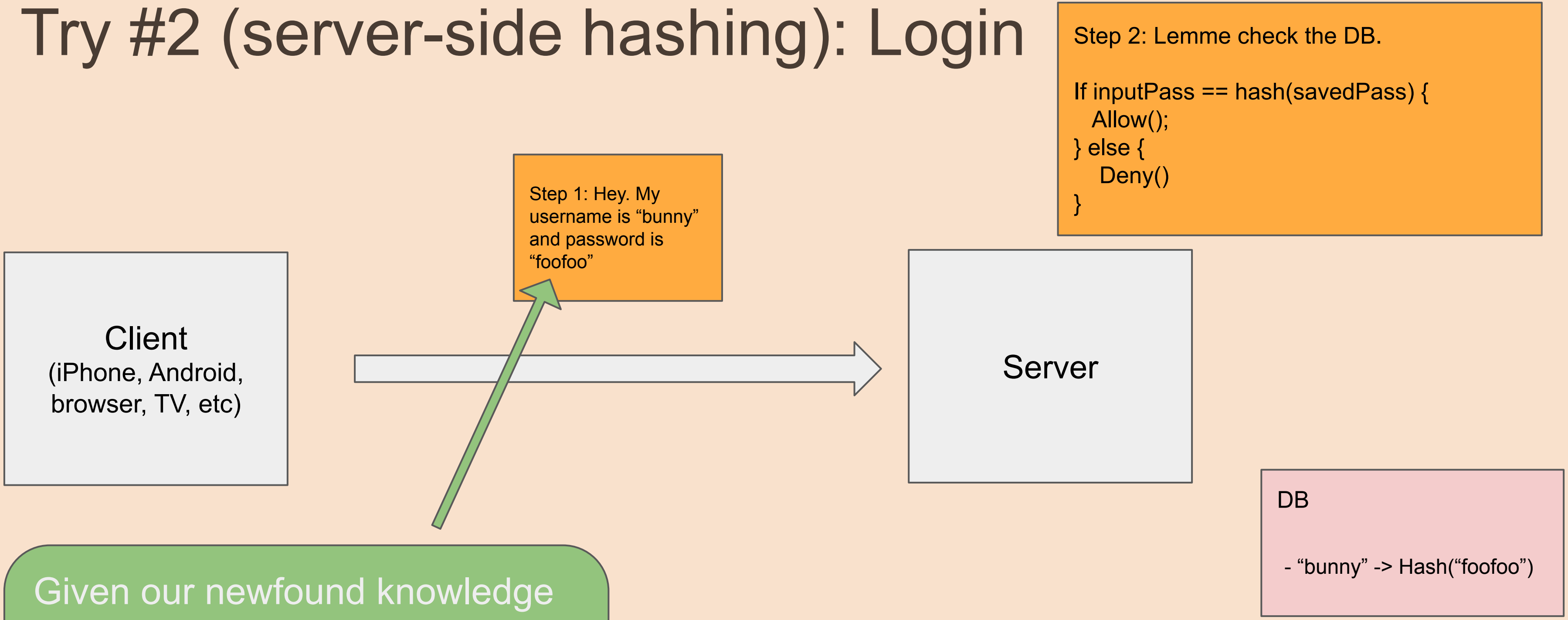
Client
(iPhone, Android, browser, TV, etc)

Server

DB

- “bunny” -> Hash(“foofoo”)

Try #2 (server-side hashing): Login



Step 2: Lemme check the DB.

```
If inputPass == hash(savedPass) {
  Allow();
} else {
   Deny()
}
```

Step 1: Hey. My username is “bunny” and password is “foofoo”

Client
(iPhone, Android, browser, TV, etc)

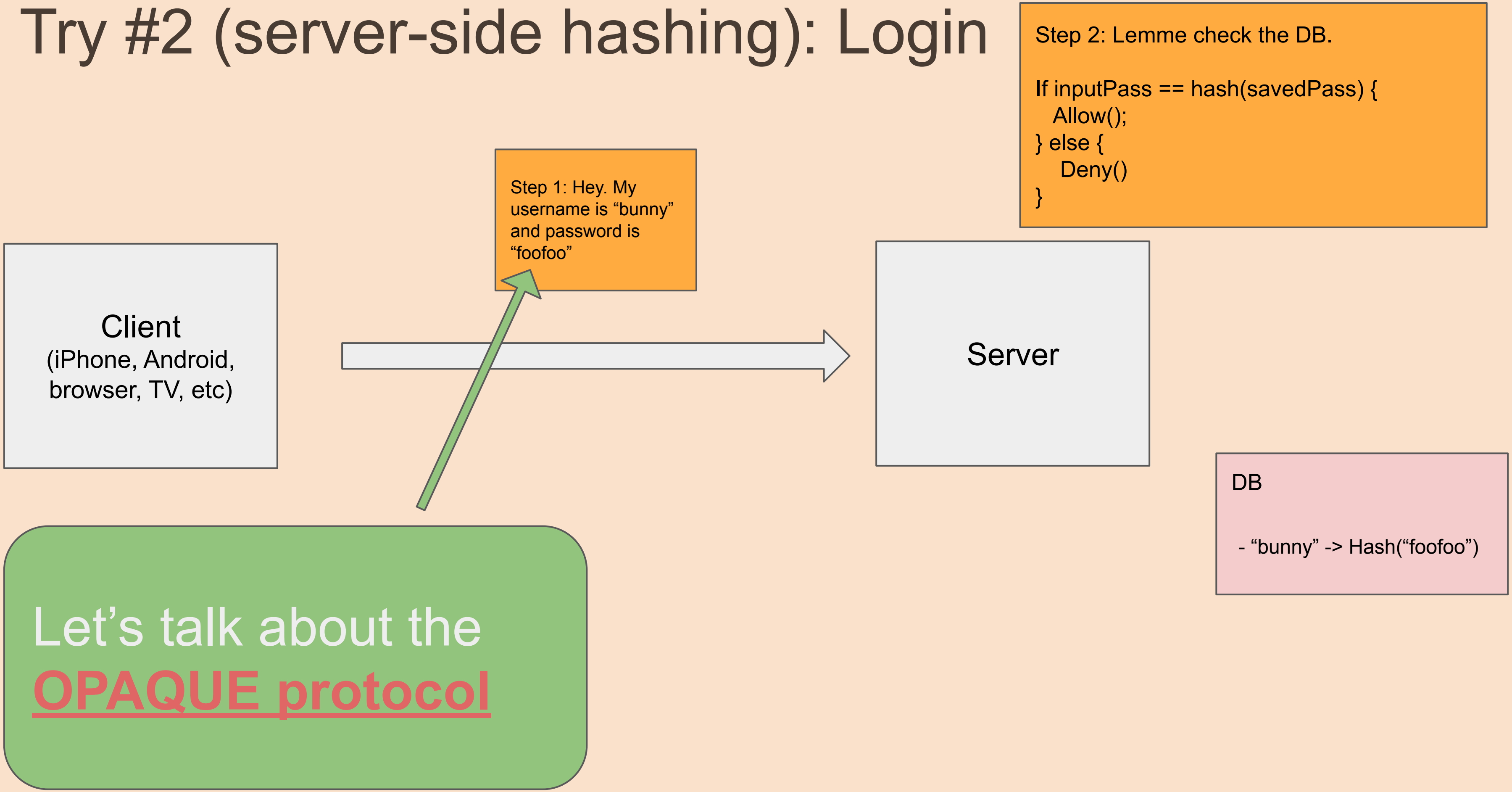
Server

DB

- “bunny” -> Hash(“foofoo”)

Given our newfound knowledge of OPRFs, **we no longer need to send the password** to the server. We can compute an OPRF instead

Try #2 (server-side hashing): Login



Client
(iPhone, Android, browser, TV, etc)

Step 1: Hey. My username is “bunny” and password is “foofoo”

Server

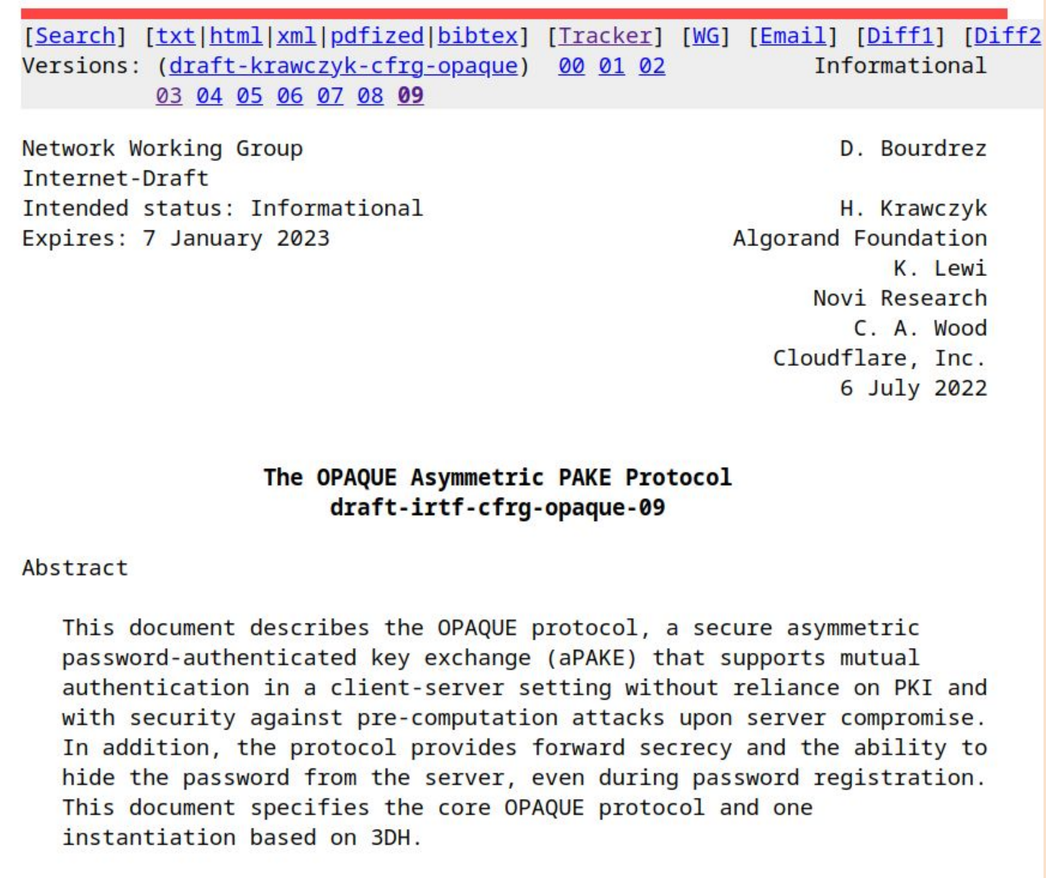
Step 2: Lemme check the DB.

```
If inputPass == hash(savedPass) {
  Allow();
} else {
   Deny()
}
```

DB

- “bunny” -> Hash(“foofoo”)

Let’s talk about the **OPAQUE protocol**



[Search] [txt|html|xml|pdfized|bibtex] [Tracker] [WG] [Email] [Diff1] [Diff2
Versions: (draft-krawczyk-cfrg-opaque) 00 01 02 Informational
03 04 05 06 07 08 09

Network Working Group D. Bourdrez
Internet-Draft
Intended status: Informational H. Krawczyk
Expires: 7 January 2023 Algorand Foundation
K. Lewi
Novi Research
C. A. Wood
Cloudflare, Inc.
6 July 2022

The OPAQUE Asymmetric PAKE Protocol
draft-irtf-cfrg-opaque-09

Abstract

This document describes the OPAQUE protocol, a secure asymmetric password-authenticated key exchange (aPAKE) that supports mutual authentication in a client-server setting without reliance on PKI and with security against pre-computation attacks upon server compromise. In addition, the protocol provides forward secrecy and the ability to hide the password from the server, even during password registration. This document specifies the core OPAQUE protocol and one instantiation based on 3DH.

https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-opaque-09

The OPAQUE Protocol

A fast and secure authentication protocol (for registrations and logins) where

- The client’s credentials **never leave their device**
- And the server only learns from the client as much as they can to do the authentication **and nothing more**.

OPAQUE is just one incarnation of privacy-preserving authentication schemes. There’re more like SPAKE2, J-PAKE, and EKE.

OPAQUE was the finalist among similar authentication schemes and the recommended protocol by the Crypto Forum Research Group: https://github.com/cfrg/pake-selection

Try #3 (OPAQUE): Registration

Alice (Client)

Step 0: Parameter Definitions
Step 1: OPRF computation
Step 2: Key generation
Step 3: Sealing an envelope

Bob (Server)

Step 0

```
alice_secret
```

```
bob_secret
```

Step 1

```
oprf = f(alice_secret, bob_secret)
```

Step 2

```
alice_priv, alice_pub = keygen()
```

```
bob_priv, bob_pub = keygen()
```

Step 3

```
alice_envelope = encrypt(key=oprf,
                         content=(alice_priv,
                                  alice_pub,
                                  bob_pub)
                         )
```

```
db.put("alice",
       alice_envelope,
       alice_pub)
```

Try #3 (OPAQUE): Login

Alice (Client)

Step 0: Parameter Definitions
Step 1: OPRF computation
Step 2: Decrypt the registration envelope
Step 3: Derive session key

Bob (Server)

Step 0	alice_secret	bob_secret
Step 1	oprf = f(alice_secret, bob_secret)	
Step 2	alice_priv, alice_pub, bob_pub = decrypt(key=oprf, content=alice_envelope)	alice_envelope, alice_pub = db.get(“alice”)
Step 3	**session_token** = dh(alice_priv, bob_pub)	**session_token** = dh(bob_priv, alice_pub)

Try #3 (OPAQUE): Login

Alice (Client)

Step 0: Parameter Definitions
Step 1: OPRF computation
Step 2: Decrypt the registration envelope
Step 3: Derive session key

Bob (Server)

Step 0	alice_secret	bob_secret
Step 1	oprf = f(alice_secret, bob_secret)	
Step 2	alice_priv, alice_pub, bob_pub = decrypt(key=oprf, content=alice_envelope)	alice_envelope, alice_pub = db.get(“alice”)
Step 3	**session_token** = dh(alice_priv, bob_pub)	**session_token** = dh(bob_priv, alice_pub)

This is a **shared, short-lived, single-use session_token**, computed by **both parties**, without ever sharing **alice_secret** over the wire

Try #3 (OPAQUE): Issues

- **RESOLVED** Server saves client's password in plaintext
- **RESOLVED** Client sends their password in plaintext

Try #3 (OPAQUE): Issues

- **RESOLVED** Server saves client’s password in plaintext
- **RESOLVED** Client sends their password in plaintext

Try

- word in plaintext

d in plaintext

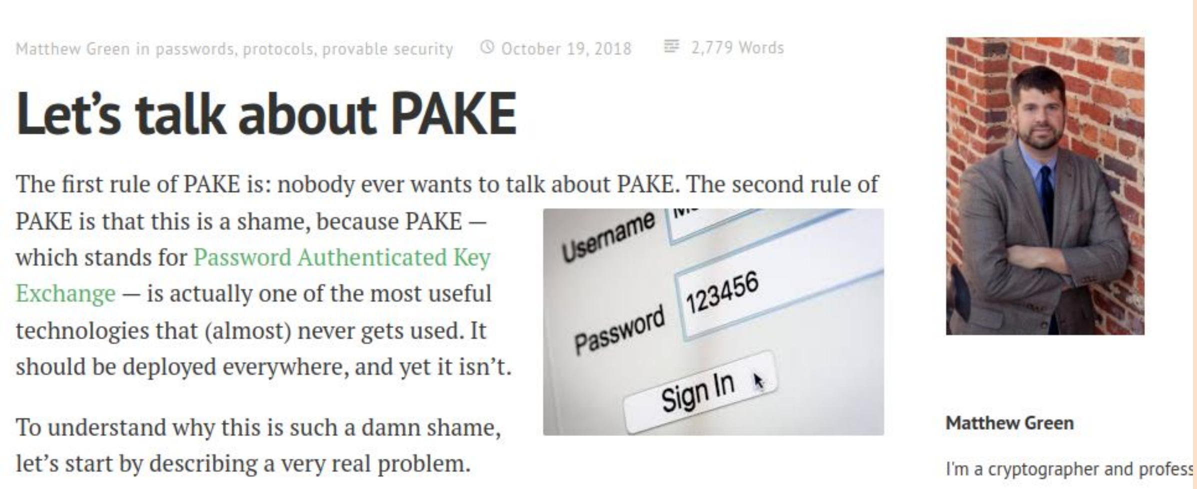
Okay, OPAQUE is cool.

Try

- wo

d

Okay, OPAQUE is cool. Why is this guy here?



Matthew Green in passwords, protocols, provable security October 19, 2018 2,779 Words

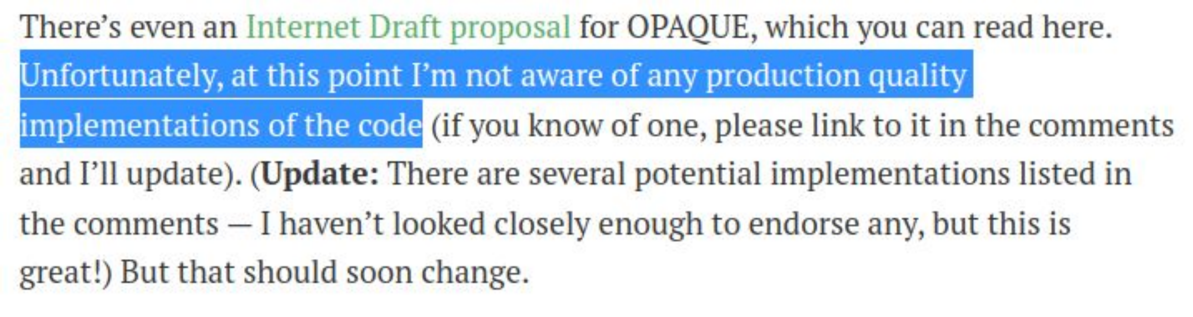
Let's talk about PAKE

The first rule of PAKE is: nobody ever wants to talk about PAKE. The second rule of PAKE is that this is a shame, because PAKE — which stands for Password Authenticated Key Exchange — is actually one of the most useful technologies that (almost) never gets used. It should be deployed everywhere, and yet it isn't.

To understand why this is such a damn shame, let's start by describing a very real problem.

Matthew Green

I'm a cryptographer and profess



There's even an Internet Draft proposal for OPAQUE, which you can read here. Unfortunately, at this point I'm not aware of any production quality implementations of the code (if you know of one, please link to it in the comments and I'll update). (**Update:** There are several potential implementations listed in the comments — I haven't looked closely enough to endorse any, but this is great!) But that should soon change.

https://blog.cryptographyengineering.com/2018/10/19/lets-talk-about-pake/

OPAQUE in the Wild

- I was working on a personal project where I needed a **privacy-first registration system**.
- Implementing cryptography is **hard**.
- I couldn’t find a **production-grade SDK** for easy use across multiple platforms

So, I wrote

OPAQUE in the Wild

So, I wrote an **SDK**

- I was working on a personal project where I needed a **privacy-first registration system**.
- Implementing cryptography is **hard**.
- I couldn’t find a **production-grade SDK** for easy use across multiple platforms

So, I wrote

Plissken: Privacy-First, Zero-Knowledge Password Authentication Suite

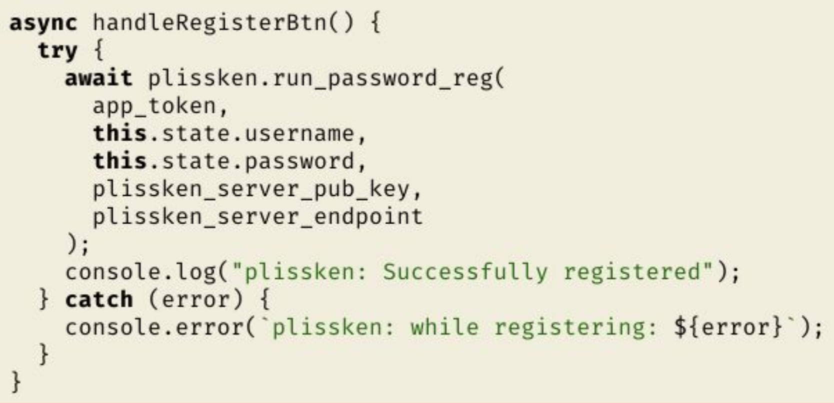
https://github.com/afjoseph/plissken

Plissken

- Open-source SDK for **Javascript, Android and iOS**.
- Provides backend and frontend components: deployment and usage should be **plug-and-play**
- Uses **security-audited** cryptographic libraries (Go’s stdlib, Cloudflare libs)
- Written in Go. Can be compiled to **WASM, JS, shared libraries** to use for **any** programming language and can produce tiny binaries for IoT devices

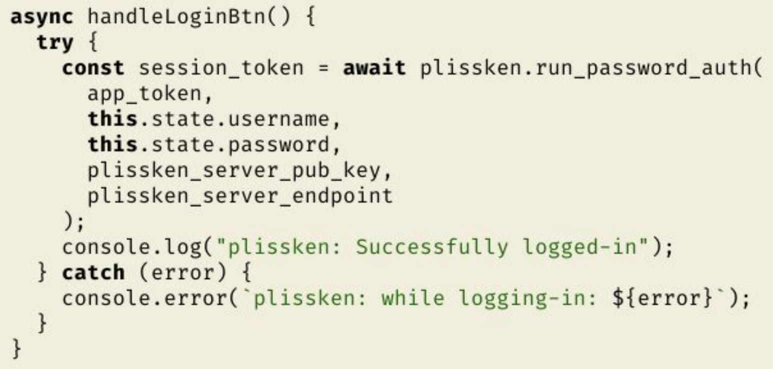
Client Login/Registration Code (JS)

Registration



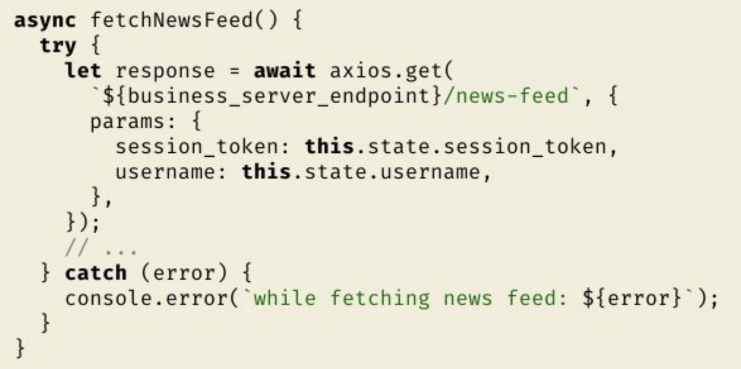
```
async handleRegisterBtn() {
  try {
    await plissken.run_password_reg(
      app_token,
      this.state.username,
      this.state.password,
      plissken_server_pub_key,
      plissken_server_endpoint
    );
    console.log("plissken: Successfully registered");
  } catch (error) {
    console.error(`plissken: while registering: ${error}`);
  }
}
```

Login



```
async handleLoginBtn() {
  try {
    const session_token = await plissken.run_password_auth(
      app_token,
      this.state.username,
      this.state.password,
      plissken_server_pub_key,
      plissken_server_endpoint
    );
    console.log("plissken: Successfully logged-in");
  } catch (error) {
    console.error(`plissken: while logging-in: ${error}`);
  }
}
```

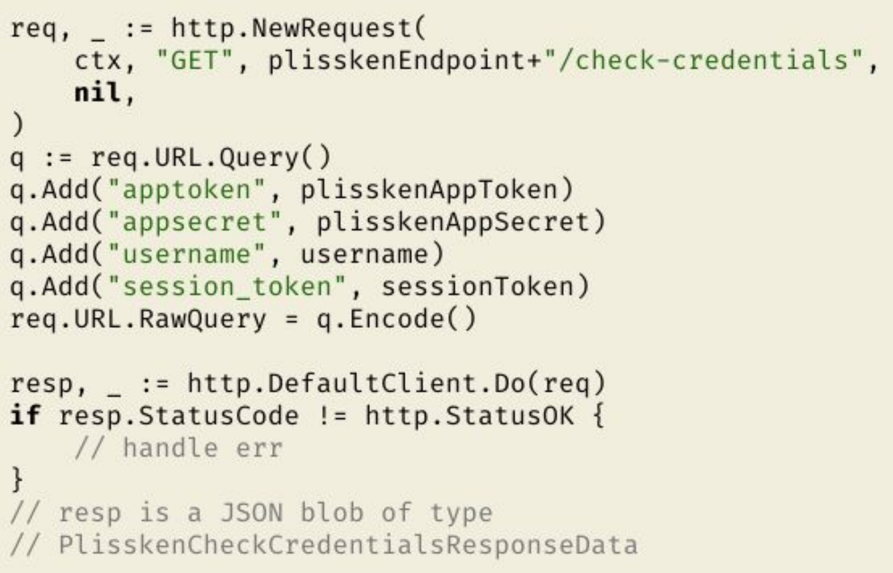
Using session tokens



```
async fetchNewsFeed() {
  try {
    let response = await axios.get(
      `${business_server_endpoint}/news-feed`, {
      params: {
        session_token: this.state.session_token,
        username: this.state.username,
      },
    });
    // ...
  } catch (error) {
    console.error(`while fetching news feed: ${error}`);
  }
}
```

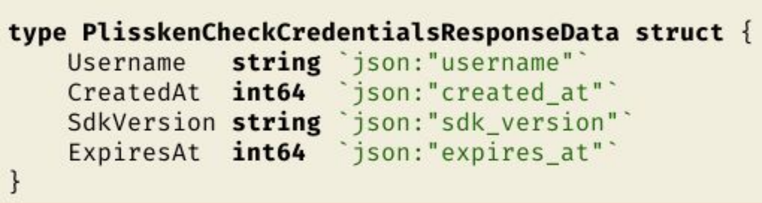
Backend Deployment/Usage Process

Check Session Tokens Through S2S Calls



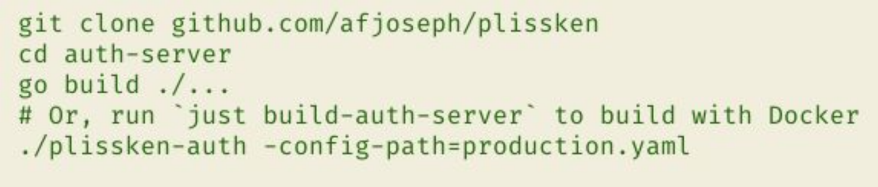
```go
req, _ := http.NewRequest(
    ctx, "GET", plisskenEndpoint+"/check-credentials",
    nil,
)
q := req.URL.Query()
q.Add("apptoken", plisskenAppToken)
q.Add("appsecret", plisskenAppSecret)
q.Add("username", username)
q.Add("session_token", sessionToken)
req.URL.RawQuery = q.Encode()

resp, _ := http.DefaultClient.Do(req)
if resp.StatusCode != http.StatusOK {
    // handle err
}
// resp is a JSON blob of type
// PlisskenCheckCredentialsResponseData
```



```go
type PlisskenCheckCredentialsResponseData struct {
    Username   string `json:"username"`
    CreatedAt  int64  `json:"created_at"`
    SdkVersion string `json:"sdk_version"`
    ExpiresAt  int64  `json:"expires_at"`
}
```

Plissken Auth Server Deployment



```
git clone github.com/afjoseph/plissken
cd auth-server
go build ./...
# Or, run `just build-auth-server` to build with Docker
./plissken-auth -config-path=production.yaml
```

Plissken Architecture: Registrations

Alice (Client)

Bob (Business Server)

1. Runs the registration protocol

Auth Server

2. Stores the password proofs

Plissken Architecture: Logins & Resource Fetching

Alice (Client)

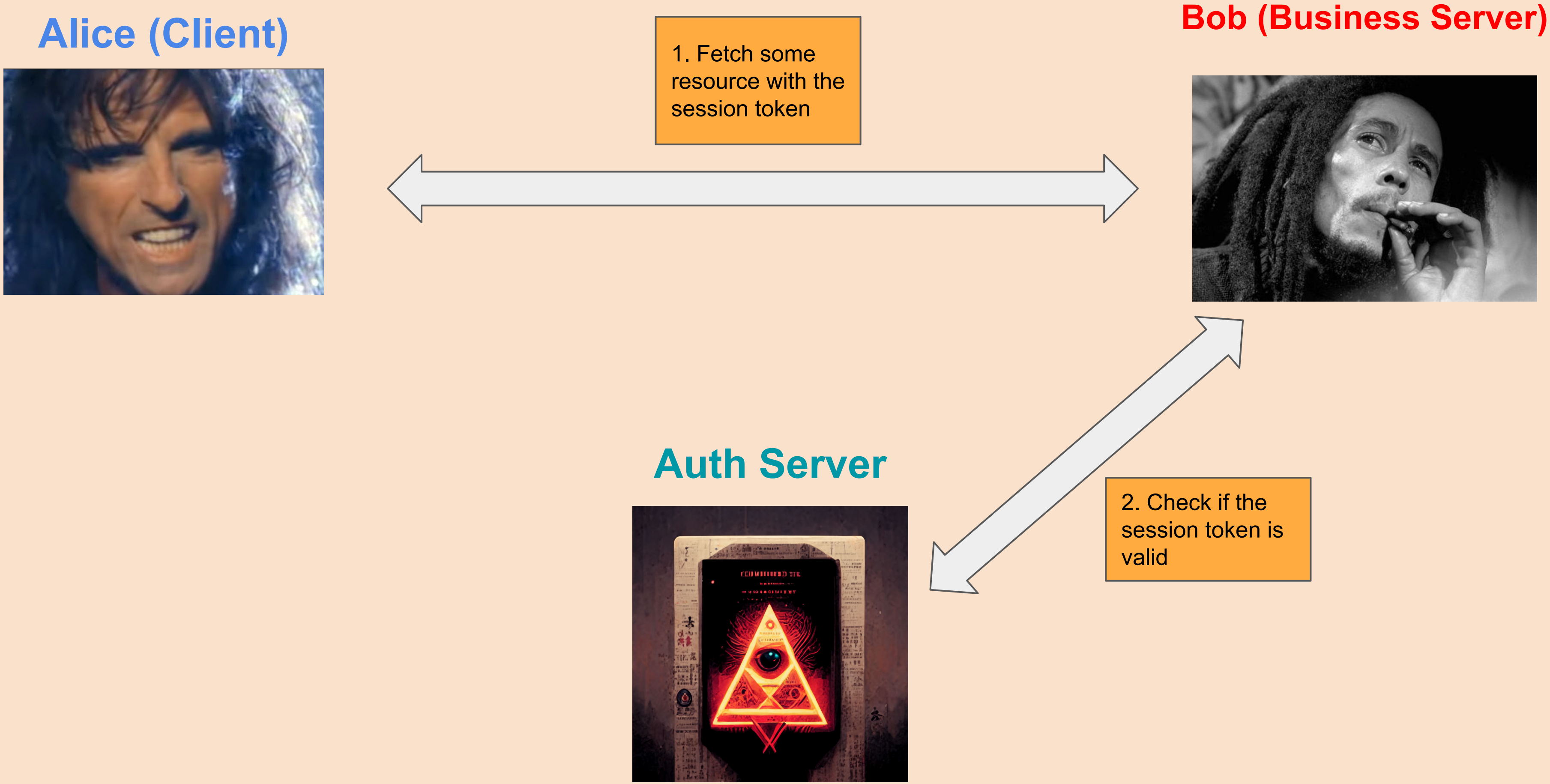
Bob (Business Server)

1. Runs the login protocol

Auth Server

2. Stores short-lived, single-use session tokens

Plissken Architecture: Resource Fetching



Alice (Client)

Bob (Business Server)

Auth Server

1. Fetch some resource with the session token

2. Check if the session token is valid

Demo

Next Steps

- Get a security audit
- More platforms and easier usage
- Use more cryptographic primitives (3DH, HMQV, etc.)

Next Steps

- Get a security audit
- More platforms and easier usage
- Use more cryptographic primitives (3DH, HMQV, etc.)

Contributions, stars and forks are welcome

Thank You!

@malwarecheese

https://github.com/afjoseph/plissken