Bachelor Thesis

Password-Authenticated Key Exchange (PAKE)

Author	**Julien Béguin**
Supervisor	Prof. Alexandre Duc
Academic year	2021-2022

Yverdon-les-Bains, September 29, 2021

Specification

Context

Password-authenticated key exchange (PAKE) are very powerful cryptographic primitive. They allow a server to share a key with a client or to authenticate a client without having to know or to store his password. For this reason, they provide better security guarantees for initializing a secure connection using password than usual mechanism where the password is transmitted to the server and then compared to a hash. Despite its theoretical superiority, PAKEs are not implemented enough in the industry. Many old PAKEs were patented or got broken which might have hurt the adoption of this primitive.

Goals

1. Outline existing PAKE. This includes SRP, OPAQUE, KHAPE, EKE, OKE, EKE variants (PAK, PPK, PAK-X,), SNAPI and PEKEP. Also look for other less known PAKEs.

2. Study in detail the main PAKE — EKE, SRP, OPAQUE, KHAPE — and understand their differences.

3. Choose one of the modern PAKEs to implement. The choice is based on the properties of the PAKE, the existence of implementations for this PAKE and the existence of standards for this PAKE.

4. Design an interesting use case where using a PAKE is more appropriate than using a classical authentication method. The advantages of the PAKE are detailed in the report.

5. Implement the chosen PAKE and the use case using the desired programming language

Deliverables

- Implementation of the chosen PAKE with the use case

- Report containing :

    - PAKEs' state of the art,

    - Description of the use case,

    - Advantages of using a PAKE over a classical authentication method for this use case,

    - Implementation details

Contents

1 | Introduction

1.1 Problematic

1.1.1 Authentication

How to authenticate a user ? When a user want to connect itself to a online service, he send its username (or email) for identification. Then, he need a way to prove to the server that he is indeed the person he pretend to be. This is what we call authentication. Without it, anybody can impersonate the account of someone else.

Authentication can be based on multiple factors. Something that the user knows (e.g. password, PIN, ...), something that the user has (e.g. digital certificate, OTP token device, smartphone, ...) or something that the user is (e.g. fingerprint, iris, ...). Multiple factors can be combined to obtain a strong authentication.

Traditionally, the user send the authentication value to the server through a secure channel to avoid eavesdropping and then the server compare the value that he received to the value that he store for the specific user.

This means that the server has to knows and store this sensible value before authentication (generally during register).

Traditionally on websites and softwares, passwords are used as authentication value. They are the easier to implement and the most familiar to the users.

Attacks and mitigations This setup is not ideal and can lead to multiple attacks.

In case where the server get compromised, since the server store the passwords, the attacker immediately obtain access to all passwords. This means that he can impersonate every user.

To avoid this scenario, numerous technique has been developed. - memory-hard password hashing function (scrypt, Argon2, etc...) - salt - pepper

These techniques improve the security of storing password but they doesn't address a

deeper problem; When the user wants to login, he has to send its cleartext password to the server in order for the server to authenticate the user. This necessity void any password storing improvement if the server is ever persistently compromised or if password are accidentally logged or cached.

Why passwords are bad ?   Passwords are a problem. They are hard to remember and to manage for the user. They are generally low-entropy and users are reusing the same passwords too often. A password manager can help to manages password but there is a greater underlying problem. The problem is that "a password that leaves your possession is guaranteed to sacrifice security, no matter its complexity or how hard it may be to guess. Passwords are insecure by their very existence" [**?**]. Now-a-day, majority of password use require that the password is sent in cleartext.

Even if the channel between the client and the server is appropriatly secured, generally with TLS (Can also fail: PKI attack, cert missconfiguration, ... TODO), and even if on the server-side every secure password storing techniques are implemented, the password still has to be processed in cleartext. As stated before, there can be some software issue like accidental logging or caching of the password. But hardware vulnerabilities are not to forget. While the password in processed in clear, it reside on the memory. It use a shared bus between the CPU and the memory. Hardware attacks are less likely to occur but are no less severe (Spectre, Meltdown).

In a ideal world, the server should never see the user's password in cleartext at all.

Get rid of password   In summary, password are not ideal. They are difficult to remember, annoying to type and insecure. So why don't we try to get rid of them altogether ?

Promising initiatives to reduce or remove passwords are emerging and improving. (TODO examples: WebAuthn).

These solutions are a good replacement to passwords but they require a deep change. It will take time for them to grow mature and impose themself as industry standard.

This is also because password are so ubiquitous due in part the ease of implementation and the familiarity for the users.

If we cannot get rid of passwords for now, we need a way to make it "as secure as possible while they persist".

This is where PAKE become interesting. It allow password-based authentication without the password leaving the client.

1.1.2 Password-Authenticated Key Exchange

PAKEs at the rescue Password-Authenticated Key Exchange (PAKE) are cryptographic primitive. There is two types of PAKEs:

- Symmetric (also known as balanced) PAKE where the two party knows the password in clear

- Asymmetric (also known as augmented) PAKE designed for client-server scenarios. Only the client knows the password in clear

For the moment, we will focus on asymmetric PAKE (aPAKE) because it is the one that can solve our authentication problem.

aPAKE guarantee that the client's password is protected because it never leave the client's machine in cleartext.

2 | State of the art

3 | OPAQUE (or) KHAPE

4 | Use case: ...

5 | Implementation

6 | Conclusion

List of Figures

List of Tables