# Content

1. Introduction
   1. Aims
   2. Motivation
2. Background
   1. PAKE
      1. Need for PAKE
      2. What is PAKE
   2. EKE
      1. What is EKE
      2. EKE protocol design
3. Design
   1. Protocol
   2. Parameter selection
4. Implementation
   1. Challenges
      1. Large number operations
         1. Miller rabin test
         2. Modular Exponentiation
      2. Primitive root for large numbers
         1. Math behind it
      3. KDF scrypt
5. Evaluation
6. Conclusion

# 2

## (a)PAKE

### 2.a (i) Need for PAKE

The most common method employed for ensuring secure communications over the Internet is TLS (Transport Layer Security).

The TLS connection commences with a TLS handshake wherein the parties agree on the encryption algorithm and cryptographic parameters, generate the shared secret key, and authenticate each other. In simpler terms, an agreement is reached on the communication method, and mutual authentication is performed. Messages can then be encrypted and transmitted through a confidential TLS channel. The server can decrypt a message sent by the client and also verify its integrity. Subsequent to decryption, the server gains access to the original plaintext message.

When this mechanism is utilized for password authentication, the password is sent by the client to the server using the confidential TLS channel. The server decrypts the password and verifies it against a one-way hash image of the password (since the hash of the password and the salt to calculate the hash are stored in the database). This way, if access to the database is gained by an attacker, they only obtain the hash value. Consequently, they will need to execute an exhaustive offline dictionary attack to deduce the password.

However, this method possesses an Achilles' heel - the user transmits their cleartext password to the server as it is required for the server to conduct the check. If the TLS connection is established with a compromised server, an attacker might discover the user's password that appears in clear text.

PAKE provides a solution where the password does not have to be sent to the server in cleartext.

### 2.a(ii) What is PAKE

PAKE, an acronym for Password Authenticated Key Exchange, represents a protocol facilitating the establishment of a shared secret key between two parties for the purpose of secure communication. The key is formed over an untrusted network, with each party authenticating each other through a password, ensuring that the password is not transmitted in cleartext to the server. The key exchange is secure even when the messages exchanged are fully observable by a potential attacker.

In the context of PAKE, the authentication password is short and easily memorizable. The protocol's structure is designed in a manner that eliminates the necessity for the password to be uniformly distributed over a large space to ensure security. This property makes it more convenient than alternative authenticated key exchange protocols.

Nevertheless, numerous PAKE protocols remain susceptible to dictionary attacks, wherein an attacker tries all possible passwords to compromise security (brute-force method). Consequently, these protocols should account for the fact that the passwords used have low entropy.

## (b) EKE

### What is EKE

The encrypted key exchange (EKE) protocol of Bellovin and Merritt was the first successful PAKE protocol.

It is a combination of asymmetric (public key) and symmetric cryptography (private key). Two parties can use a shared secret (password) to exchange secret information like the session key. The password is protected from offline dictionary attacks. The protocol is also secure against active attacks.

### EKE protocol design

We assume that parties A and B, with public keys E\textsubscript{A} and E\textsubscript{B}, share the password P. \\

A then sends encryption of E\textsubscript{A} using P\\

i.e., \hspace{1in} P(E\textsubscript{A})\\

B decrypts it and generates a random secret key R. Subsequently, B sends the encryption of R using E\textsubscript{A} and P \\

i.e., \hspace{1in} P(E\textsubscript{A}(R)) \\

A decrypts this message to obtain the common secret key R.\\

A and B proceed to send each other challenges encrypted using R to validate R (check that they have arrived at the same secret key). Hence, A sends an encrypted challenge\textsubscript{A} \\

i.e., \hspace{1in} R(challenge\textsubscript{A}) \\

B decrypts this and reciprocates by sending its own challenge to A along with the decrypted challenge.\\

i.e., \hspace{1in} R(challenge\textsubscript{A}, challenge\textsubscript{B}) \\.

A then decrypts this. If the decrypted challenge\textsubscript{A} matches the original one, everything is okay. Then A sends back challenge\textsubscript{B} \\

i.e., \hspace{1in} R(challenge\textsubscript{B}) \\

B then decrypts it. If it matches the original challenge sent, both parties are confident that they have arrived at the same secret key, which they can now use for communication.

### EKE implementations

EKE can be implemented using RSA or using El Gamal encryption, which is an encrypted version of Diffie-Hellman key exchange.\\

I chose to implement EKE based on Diffie-Hellman.