Implementation of a key exchange protocol

with an online trusted third-party

Krissten J. Martínez F, Carlos A. Venencia S, Daniel G. Marquez A

Computer Science and Engineering Department

1fuenmayork@uninorte.edu.co

2cvenencia@uninorte.edu.co

3dgmarquez@uninorte.edu.co

Abstract— This document contains the investigative and developmental task of the implementation of a key exchange protocol with an online trusted third party for the Cryptography course, Universidad del Norte in 2022.

Keywords— Cryptography, Key exchange protocol, TLS, Public Key, Certificate Authority

1. Introduction

Cryptographic protocols and network security protocols are created by mathematicians and computer engineers to provide solutions to various problems, the one in question here being the ability to secure the communication between peers over an insecure network. The project’s objective is to develop a working key exchange protocol and a trusted third-party certificate authority for communication between two different users/peers.

1. Investigation
2. Authenticated Key Exchange protocols (AKE)

Authenticated Key Exchange protocols exist to provide security when two pairs wish to communicate over an insecure network. For this task, a secure channel is required with each of the pairs having a **shared key** and with the use of a protocol such as IPsec, which provides authenticated encryption of packets.

AKE protocols come into play when said pairs wish to establish this **shared key.** In this process each pair is allowed to establish a **session key** and know exactly which user they are talking to. A secure AKE protocol should ensure that keys are randomly generated for each paired connection.

1. Trusted Third-Party (TTP)

For the typical AKE protocol, the existence of a TTP has to be assumed. This **trusted third party** helps pairs who have no prior relationship which each other to communicate. For this to occur, each user must perform a registration protocol with the TTP, where a successful registration means the user is left with a **unique** and **offline** (for our project) **long term secret key**. Users do not share any private information with the TTP. These work like CA’s or Certificate Authorities, which bind the *identity* of a user to a *public key*.

Any given user may use AKE protocol many times, and while it has access to one **long term secret key**, each run of the AKE protocol should produce a fresh **session key**. This freshness guarantees security between different/multiple sessions.

*Basic AKE security (very informal)* [2]

***Three levels of (core) security:***

**- Static security:**

Suppose Pair1 successfully completes an AKE to obtain (k, Server)

If the Server is **not** corrupt, then:

**Authenticity** for Pair1: (similarly for Server)

* If Pair1’s key k is shared with anyone, it is only shared with Server

**Secrecy** for Pair1: (similarly for Server)

* To the adversary, Pair1’s key k is indistinguishable from random

  (even if adversary sees keys from other instances of Pair1 or Server)

**Consistency**: if Server completes AKE then it obtains (k, Pair1)

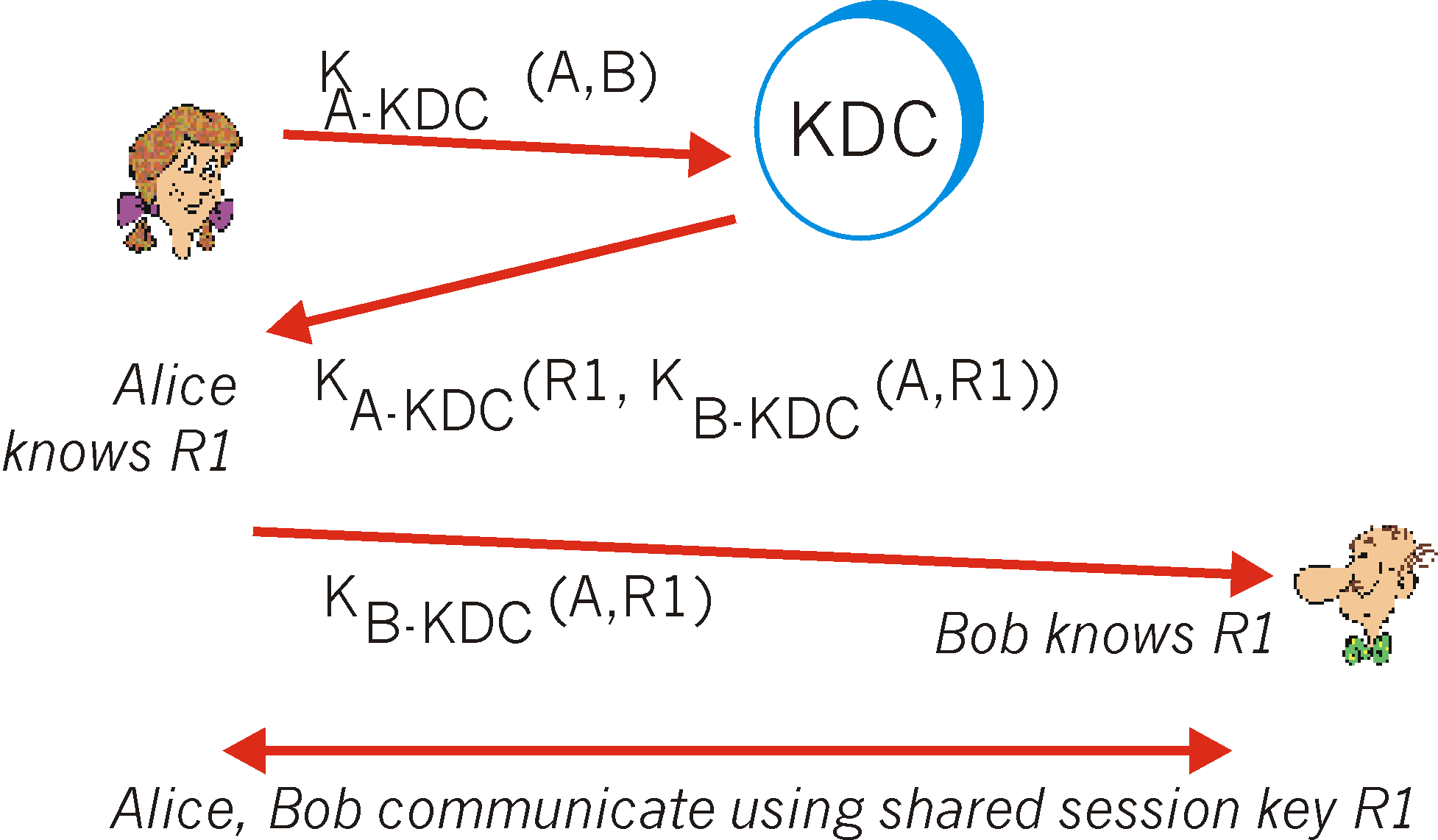
**-Forward secrecy:** static security, and if adversary learns the secret key from bank at time T

            then all sessions with Bank from time t<T remain secret.

**-HSM security:** if adversary queries an HSM holding the Servers’s secret key n times, then at most n sessions are compromised. Moreover, forward secrecy holds.

1. Design

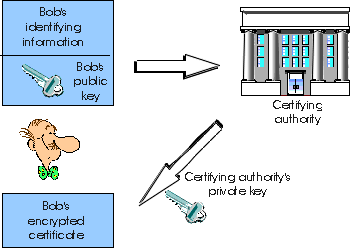
It is possible to create a protocol that uses a TTP not only on the registration phase, but also on the key exchange process. A TTP that plays an active role in both situations is called a **key distribution center (KDC)**.



*Figure 1.1:  Setting up a one-time session key using a Key Distribution Center*

While a key exchange protocol with an online TTP is very efficient on the client side, it can generate a tremendous load on the TTP. This is especially true in situations with large amounts of clients, like in a service allocated on the internet. This would be a very unreliable system since it would be very expensive to build and scale. Furthermore, an attack on the TTP would expose not only present and future key exchanges made, but also past ones. For these **security** and **scalability** concerns, it is difficult to implement it for the global internet.

However, this approach can be valid and practical on a smaller scale, for example, on a corporate network.



*Figure 1.2:  Bob obtains a certificate from the certification authority.*

This protocol uses a CPA-secure symmetric cipher and a secure MAC (Message Authentication Code) [1]. The MAC is generated using a hash function, which receives a key and the message to be checked. It returns the calculated code of fixed length.

**This protocol assumes the existence of a private channel between the user and the TTP**.

The process that a user *P* should follow to register with the TTP is as follows:

1. Send the message *idp*(identity of user *P*) to the TTP to indicate that the user is requesting to be registered.
2. The TTP generates a random long term secret key

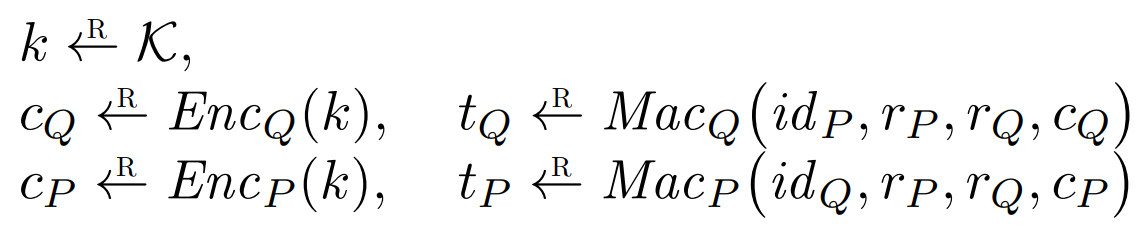
*kP ←R (kenc,P , kmac,P ) ∈ Ke × Km*

and stores the pair (*idp , kp)* in a private table.

1. Then, the TTP sends the message *kp* to *P*.

This process must be done by every user that wishes to communicate with other users through the TTP. After this point, if users *P* and *Q* wish to communicate with each other, they must complete the following steps:

1. *P* computes *rP*: a *nonce\**
2. *Q* computes *rQ*: a *nonce*
3. TTP checks if it has a secret key *kP* and *kQ* and aborts if not. Otherwise, computes:



1. TTP sends *(cQ , tQ)* to *Q* and sends *(cP , tP , idQ , rQ)* to *P*.
2. *P* and *Q* perform the following steps as *X* (self) and *Y* (other):

*a) X* verifies that *tX* is a valid MAC on the message *(idY , rP , rQ, cX)*, and aborts if not.

* 1. *X* decrypts the ciphertext *cX* and verifies that *cX* decrypts to a message *k*, which is a valid key from the space of keys, and aborts if not.
  2. *X* terminates successfully.

\* **Nonce:** arbitrary random number that is used once in a cryptographic communication.

1. Implementation

The solution to this problem was implemented using Python and the PyCryptodome library. The communication between the client script and the trusted third party script was achieved using sockets.

tty.py, acts as **key distribution center,** which means it handles registration of users and secure key distribution. New users are added to the user table and are sent back their own randomly generated key.

Inside this script the communication request handler listens for communication requests, when two clients connect, they send their ID’s and nonces, then it generates a secret key, encrypts it using each of the clients secret keys, calculates Tq and Tp and sends the information to the clients.



The client.py script connects via sockets to the trusted third-party and generates a nonce to later send them to the TTP. The script stays listening for a response and then verifies if the MAC keys are correct for each pair to then decrypt the key.



The full implementation of the forementioned key exchange protocol was coded using python, the python PyCryptodome cryptography library, sockets for the connection of the client and the TTP. The **code** and user-manual are accessible on Github on the following link:

[https://github.com/cvenencia/TTP-Implementation](https://github.com/cvenencia/TTP-Implementation.git)

1. Conclusions

The task of investigating and implementing a key exchange protocol with an online third-party was met for demonstration purposes.

References

[1] D. Boneh and V. Shoup, “A graduate course in applied cryptography - toc.cryptobook.us,” *A Graduate Course in Applied Cryptography*. [Online]. Available: https://toc.cryptobook.us/book.pdf. [Accessed: 14-Apr-2022].

[2] D. Boneh, “Auth. Key Exchange - Stanford University,” *Auth. Key Exchange*. [Online]. Available: https://crypto.stanford.edu/~dabo/cs255/lectures/key-exch.pdf. [Accessed: 14-Apr-2022].

[3] J. Kurose, “7.5 key distribution and Certification,” *Key Distribution*. [Online]. Available: https://userpages.umbc.edu/~dgorin1/451/security/dcomm/keydist.htm. [Accessed: 15-May-2022].