Key Distribution Centres

Rangaradj Marc Paridimal
40479748@napier.ac.uk
Edinburgh Napier University - Applied Cryptopgraphy And Trust (CSN11131)

1 Introduction

The Key distribution is the promise of the data communication with an absolute security and independent from an actor with bad intentions.
The Key Distribution Centre is a trusted third party in charge of providing this level of security by performing different actions from generating some keys, performing encryption, bringing a secure access to different services based on a client profile.
Despite some improvements over the time the challenges are remaining: Performance, cost, security.
This paper will give a presentation of different KDC models starting from the most popular one within the enterprises: Kerberos.
This protocol will be explained as well as the attacks performed on this protocol. Some alternative version of Kerberos will also be presented in addition of solutions to remediate these weaknesses.
The other part will cover the other KDC implementation using the symmetric keys (IOT) and then an approach of KDC implementation using asymmetric keys. Both models are based on suggested innovative models.
The paper will then show that the different models are weak against the Man in the Middle attacks and will try to analyse what answers can be provided against this challenge.
The last section will try to do a projection on something that is not yet widely deployed but seen as the tomorrow's challenge: Quantum Key Distribution.

The implementation part will emulate Key distribution via the BB84 protocol based on QKD.

2 Literature Review

2.1 How does KDC using symmetric keys works (MIT version)

We are going here to explain in quick few high-level steps how the KDC [1] [13] concept works using the symmetric key. The idea is there then to introduce to Kerberos which is the protocol that put in place what is going to be explained.
1) KDC got Alice and Bob's keys.
2) Alice sends a request to the KDC with her ID and Bob's ID.
3) KDC understand then that Alice and Bob want to communicate so it generates a random session key. (Kses or Kab)
4) The generated session key is then encrypted by the KDC using AES256 using the key he shares with Alice (Ka) =¿ encrypted sessions key is EKsesalice EKsesbob
5) KDC encrypt the session key for Bob the same way. (using Bob's Key)
6) KDC sends then the 2 encrypted keys (encrypted session keys: EKsesalice EKsesbob) back to Alice who can now:
a. Decrypt her session key (because KDC encrypted the session key with Alice's key which is shared with the KDC)
b. Alice can't do anything with Bob's encrypted session (EKsesbob) Key BUT she managed to decrypt and get and her own session key (Kses) SO she can start encrypting the message using this session key.
c. The message X will be encrypted using the session Key. (Ksesalice)
d. This encrypted message is sent over the channel.
7) Bob gets the message and he's got Alice's identifier so he knows it's coming from Alice.

He tries to decrypt the message but doesn't have his sessions key. However, bob's encrypted session key (EKsesbob) has been sent over by Alice attached with the message.
Bob decrypts the encrypted session key (EKsesbob) using the key he shared with KDC and get the sessions key (Kses). (the same way Alice has been doing in step 6.b)
He uses the session key to decrypt the message.(statics key Ka, Kb = Key Encryption Keys)
The interesting points here are the following:
- The Keys from Alice and Bob are known to be long term keys.
- We are using these encryption key which is shared between Alice, Bob and the KDC to encrypt the session key.
- The encryption key is used to create the secure tunnel to communicate over a non-secure channel.
- The session key is an additional protection for Alice when she sends the message to Bob to be the only one to be able to decrypt the message as he will be the only one to be able to decrypt the "full packet" using his encryption key.
- The session Key have got a lifetime.

Without even speaking about the Kerberos, which is the protocol based on KDC potential weaknesses can be highlighted such as:
- KDC a central point of failure. [2] [6]
- What if the encryption keys are been compromised? [6]
- How to secure Alice and Bob for not being comprised themselves which would conduce to a data leak.
- How can Alice verify the authenticity of the encrypted keys sent by the KDC? This highlights the possibility of replay attack.[3][6] Indeed, the None of the parties verify if the session key has ever been used. If a bad third-party actor manages to get an old key, he will be able to impersonate the KDC and send old messages to Alice and Bob. As the bad actor got the old key it means that he somehow been

able to decrypt it and get the session key (using the key from Alice for example) and then be able to decipher the encrypted plaintext.
- This model doesn't seem to have any key confirmation mechanism. Indeed, maintaining the session key secret between 2 parties is a challenge as anyone can send a legitimate request to talk with Alice. This is what we call the man and the middle attack. [4] This third party that we are going to call Eve will then get all the different messages and Alice won't be able to tell if the messages (encrypted packet) that she is receiving are coming from Bob. The only way to verify that information would be to decrypt the packet using Bob's key:
o If Alice is able to decrypt it then it will prove that she is indeed communicating with Bob
o If not if means that someone else send the encrypted packet and she could drop it.
Some answers will be provided later on while we will be covering protections provided for Kerberos. Indeed, as Kerberos is using the same model, we are going to get the same problems.

2.2 Implementation of Kerberos

Kerberos is the implementation of the mechanism described above. It's been since then standardised in RFC4120 [10] updating the RFC1510 [9] which is used to be known as the oldest identity protocol in use as of today.
The following picture gives a detailed view of the Kerberos Authentication Protocol [5]:

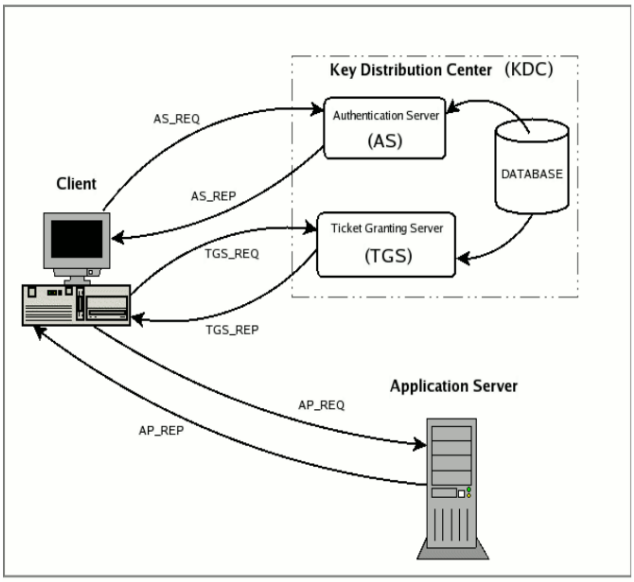

Figure 1: **Kerberos Authentication**

Here a description on how Kerberos work in detail, [8] [11] which illustrate the picture above:
In order to implement a Kerberos authentication, 3 entities need to be linked:
- The Client C who needs to obtain a ticket to access the Server S.
- The server S which is an instance of a service requested by C.
- The authentication service (AS) which authenticate the client regarding the request service ticketnew.
- The TGS (Ticket Granting Server) which authenticate the client on the final Service only if the authentication with the server has been successful.
- The KDC (Key Distribution Centre = AS + TGS) which represents the trusted third party.

The way Kerberos works can be compared to how the cinemas are working. The client C first buy his ticket to the checkout AS. The client C get a ticket in exchange of money. The client will then present the ticket to the checkpoint TGS to access to the cinema. Just before entering in the room there's a last control where the client will be controlled to see if the ticket if still valid. (ticket life time)

The encryption method follows the same logic than what has been explained in 2.1 but some terms can be defined [12]:
Long term key: key used between an Alice and the KDC for distributing the session key (this session key is used to encrypt information between Alice and Bob) This is a hashed version of Alice's password and also that corresponds to the encryption key described on the previous section.
In that way the password (which corresponds again to the long-term key) will never be sent over the network and will be used to encrypt the session key.
One other interesting point to highlight is the fact that the session key got a limited life time.

2.3 Attacks on Kerberos

Unfortunately, Kerberos is vulnerable to several attacks:

Replay attacks [14] [15]:
As mentioned previously, the server ensures that the client can access the service by validating only the last request sent: Application Server Request AP_REQ.
This replay attack requires the attacker to set up a Man-In-The-Middle between the client and the server. Thereafter, there are two possibilities:
- Either the attacker listens to the network and returns the AP_REQ request sent by the client in order to obtain access to the service.
- Either the attacker prevents the client from sending the AP_REQ request to the server and uses it to obtain access to the service instead of the client.

Several countermeasures have been proposed to reduce the impact of this vulnerability:
- Timestamp: The duration of use of the AP_REQ is limited to a certain time (generally 5 minutes).
- Cache: The server (V) stores in memory the requests (or more precisely the authenticators) made by the client during the authorized period of use. Thus, all duplicate requests are rejected.
- IP address: The ticket provided by the KDC may contain the list of IP addresses authorized to use this ticket. This information is kept in the AP_REQ request. Thus, the server is able to check whether the sender of the request has the right to use this ticket.

Pass the Ticket attack [15] [16] [17]: The original presentation of this attack was from 2008 Emmanuel Bouillon at the PacSec conference as well as during the Blackhat conference in 2009 and 2010.
This attack allows the local authentication on the client workstation, even if the Kerberos authentication has been completely completed (AS_REQ / $AS\_REP + TGS\_REQ$ / TGS_REP). On the attacking side, this requires to control the network between the client and the KDC, as well as physical access to the equipment.

- Session Key (SKservice) between the service and the user. This key is encrypted by SKtgs.
- The session key SKtgs encrypted with the user's secret key. This secret key is the hash of the password.

The attack takes place in two phases:
1. Listening and waiting for a legitimate Kerberos authentication.
During this phase, the attacker will record the Kerberos exchanges made during a legitimate connection by the victim. The goal is to get the TS service ticket.

2. Replay of a valid ticket
The attacker will then attempt to physically connect to the client workstation, using a valid user name, as well as any password (for example "password") as follows:
A) The client station will then generate an AS_REQ which will be intercepted by the attacker.

B) The attacker will respond with an AS_REP containing:
- A forged TGT ticket. This ticket will be encrypted with a key chosen by the attacker instead of the secret key of the TGS.
- An SKtgs session key chosen by the attacker and encrypted with a key derived from the password "password".

C) The client then generates a TGS_REQ request. This then contains the forged TGT previously received, as well as an authenticator encrypted with the session key SKtgs and chosen by the attacker. This TGS_REQ is also intercepted by the attacker.

D) The attacker then sends a TGS_REP to the client workstation, containing:
- The TS service ticket intercepted during step 1. This TS contains the SKservice session key used during the original exchange.
- The attacker also sends a forged session key (let's call the SKservice), encrypted with the SKtgs session key.

The checks carried out on the client workstation side for user authentication are:
- Verification that the legitimate service key of the client station is capable of decrypting the TS ticket.
- Verification that the SKtgs session key is able to decrypt SKservice.
- Checks the validity limits contained in the TS service ticket.
In our attack, all of these conditions are met.

KDC Spoofing [14] [15]: As its name suggests, this attack is based on the possibility of spoofing KDC responses. However, as mentioned previously, the Kerberos protocol is supposed to be protected against this type of attack.
Concretely, a user having physical access to the machine and being able to control the client's requests can obtain access to the equipment with the account of a specific user and any password. To do this, the attacker just needs to perform an authentication with a fixed password and block the sending of the AS_REQ request to the KDC. Indeed, the attacker will respond to this request himself by forging an AS_REQ request with the password previously used.

2.4 Modified proposals of Kerberos

Some researchers have proposed some simplified version of Kerberos which we are going to not study but highlight the key points.

The first one is a derivative model of authentication protocols based on Kerberos. Jayati Ghosh Dastidar [18] has proposed a single-sign-on protocol based on Kerberos.
How does that work: The entire system is based on a simplified form of Kerberos where the authentication Server and Ticket Granting Server are merged. But it doesn't mean that it is less secure. Indeed, Nonce and time stamps are used to prevent the replay attack which is a big weakness of the standard Kerberos protocol. The same logic is applicable to the reflection attacks.
This model can be summarised as followed:

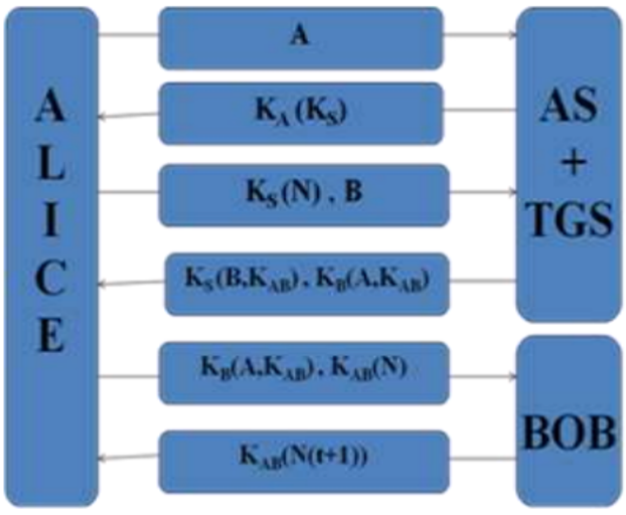

Figure 2: **Proposed Protocol**

The advantages of this protocol over the standard one is:
- The performance improved as using a single server (AS + TGS) and the extension of the shared session key. This concretely means that within a specific timestamp this key can be used as many times as possible without having to go back to an authentication process like it can be done in the

normal Kerberos protocol as described.
- This model is using the nonce rather than the timestamp which improves the security. (the nonce is used to be encrypted by session keys) The server generates a second random nonce ("challenge-response") and send it to the client (encrypted in the session key).
The client must decrypt, decrement and encrypt. (the bad actor can't do anything if he doesn't know the session key)
The weaknesses are the same one that we have previously mentioned: Single point of failure from KDC (Bottleneck and chances of compromising AG and TGS)

2) Other researchers have proposed an alternative way of implementing Kerberos and published their finding into the International Journal of Network Security [19].
They have modified the KDC behaviour where this one will save a profile in what is called a realm to generate a secret key which will be the result of: hashing the profile and as a consequence the long-term key will be independent of the user password.

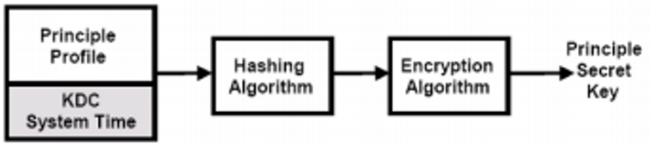

Figure 3: **Secret key generation block diagram**

What is called profile here is the type of data which will be accessed by a user and this can be: text data, image, video ...
The KDC will have a map of the principle ID and User profile which will be used to generate the principle secret key by applying a hash algorithm to the principle profile.
There is no need to go in details into the exchange but this is almost the same than the standard Kerberos. The interesting part here was the modification of the KDC in order to generate the Principle secret key.

The realm principles long-term secret keys not dependant of the password.
Session key : For any client and any server, if the TGS generates a symmetric session key for a client and a server, then bad actor can't get the session key.

2.5 Proposed solutions against Kerberos weaknesses and symmetric key distributions issues:

To fight against the different issues brought by Kerberos and the attacks against this protocol it is possible to put in place some solutions like Identity protection for a Windows environment. [33] The solutions are: Windows Defender Credential Guard (which can be enabled from the domain controller using a group policy) and also by migrating the Domain controller to Azure AD which use OAuth / OpenID Connect based protocols as authentication. The use of klist from the mimikatz tool shows that the Kerberos credentials are stored in LSASS. A proposed solution using LSASS purge allow the removal of List Kerberos credentials for all authenticated users (including services and computer account) klist, kerberos::list (List all user tickets (TGT and TGS) in user memory. No special privileges required since it only displays the current user's tickets. Similar to functionality of "klist") This being run as a regular interval can potentially mitigate the risk of attacks on Kerberos. [34] [35]

2.6 Distributed KDC using symmetric keys:

All of our daily objects called IOTs are network connected via the Internet.
This bring to "people" the same challenges that have been addressed to companies few years ago which have been partially solved by the implementation of Kerberos described in the previous sections.
A research paper is presenting a quite innovative, complex but also a secure way of implementing Kerberos applied for IOTs [31]

IOTs are using a system with multiple nodes. What does that concretely mean? if this model is compared to the "enterprise Kerberos", the KDC has been identified as a central point of failure, which means potentially vector of lots of attacks to compromise the Encryptions keys.
Here the model is different. Indeed, the AS and KDC have been segregated but their functions are remaining the same. In Kerberos (normal model) there is only one system that manages the Authentication and KDC functions. In the presented Kerberos model there are n-numbers of AS and n-number of KDC. They which keep changing their roles with the help of a random algorithm after specified period of time.
In concrete words if an attacker would like to attack the IOT device, one of the ways will be to guess and find the correct AS and KDC which add another layer of complexity. This version is no longer vulnerable to any password guessing attacks [4] as the secret key will be independent from the user password.

To clarify the description above is attached the proposed model:

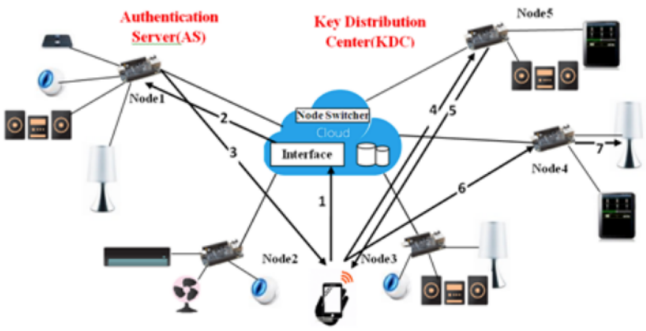

Figure 4: **Architecture of Proposed System**

The chances of AS and KDC being compromised by a bad actor are almost impossible as there are multiple nodes

within the network. (It will be tough to find AS and KDC) If an attacker is able to get information about AS or KDC, by the time he will try to compromise the other part the nodes performing roles will change.
On their implementation example the different authors have been using a login page with a secure authentication using a username and password. When the user gets authenticated, he will be prompted some choices which he will have to choose to be granted some kind of accesses. The proposed system authenticates, encrypts, decrypts, contacts cloud services and provides requested services to the user.

2.7 Distributed KDC and Asymmetric keys:

The previous section covered Kerberos with symmetric keys using N nodes of AS and KDC which has enhanced the security of the Kerberos protocol.
The presented paper here is using a different approach.

A. Mareeswari, S. Santhosh Kumar [32] have been presenting into the International Journal of Knowledge Based Computer Systems a system-based on decentralized KDC using RSA with a key size of 2048 to give access to clients some accreditations to give a time-based access to documents or any other files or resources. We can see this approach like using a phone card that we can recharge to perform further communication. Using this method, we can also repudiate access based on time to a resource. The Private Key is the mix of the client's accreditations.
The authors have been using an approach based on utilizing Attribute Based Encryption (ABE)

This guarantee:
Privacy: Unauthorized persons cannot access the information that means the data should be stored in confidential manner.
Reliability: Authorized persons can modify the information only.
Ease of Use: To access the information from any place at any time.
Please note that I have made a raw copy under quotes from the document as the goal is perfectly summarized.

This approach is in fact really interesting as it shows how things can be adapted in a highly cloud-based environment where most of people are using cloud-based storage like google drive, mega etc. . .

That being said we can see this approach is still weak to the main security issue of the Asymmetric keys which is the man in the middle attack: [1] (page 342: 13.3.1 Man-in-the-Middle Attack)
That's something applicable to all the public-key scheme like Diffie Hellman Key Exchange, RSA. . .
These attacks are possible because the public are non-authenticated.
Let's illustrate that again with Alice and Bob analogy: When Alice receives a key from Bob, she has no way to verify that it's Bob's key. That's also adaptable to the model we have previously studied.

2.8 Proposed models to mitigate the man in the Middle Attacks:

Unlike Kerberos where the Authentication Server (AS) grants a Ticket Granting Session key, which is used to encrypt the communication between the user and the TGS should there be a request to a service or resource (prevents MITM attacks), we have seen that the KDC models are generally weak to the man and the middle attacks. The public keys are not authenticated, When Alice receives a public key from Bob, she has no way of verifying if the communication is really from him.

Several papers have been written to explicitly trying to provide an answer to this issue: The first paper is using the Public Key cryptography enabled for Kerberos authentication. [37] It is bringing some additional protocols such as PKINIT, PKCROSS, and PKTAPP at different stages of the Kerberos framework.

2.9 Quantum Key Distribution:

The future will probably introduce the quantum computers, [27] and these will introduce new challenges. Like mentioned by Peter Shor [28] the quantum computer will be able to break RSA as they will be able to factor large prime numbers. It becomes then more and more important to bring an evolution into the way cryptology is being conducted to be able to resist to quantum computers. The main algorithm here is the QKD (Quantum Key Distribution) which is supposed to bring an unconditional security [20] even if we will see later that this is not necessarily the case [24] as it is not possible to guarantee to generate a perfect key using QKD. QKD is based on physics laws and is on theory the only key generation method that offers a strong security: it is not possible for a spy or an eavesdropper to keep a copy of the. Quantum signals in a QKD process thanks to the theorem of non-cloning theorem. [20][25]

How does it work? QKD take benefit of the photonic signal properties

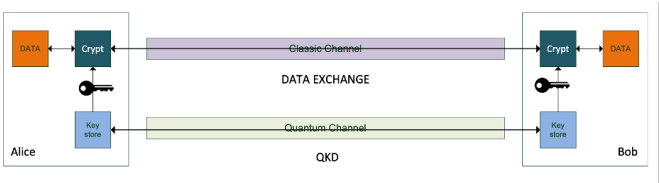

Figure 5:]
Basic QKD architecture [25]

As the picture above illustrate, Alice and Bob are exchanging a high number of signals using the quantum channel (optical fiber) and additional information are sent thought an authenticated public classical channel.
They are following a protocol that is generating a long random string of secret bits which is the key. The unconditional security is achieved by encrypting the message using the one-time-pad encryption scheme. [20]
We can distinguish the QKD protocols by the way the detections technics are used to extract the information on the encoded keys. (Using the light properties)

Challenges of performance and practical security: The implementation of such solution can bring challenges like the maximum distance within the fiber optic canal. Fibre has minimum loss of 0.2 dB/km at 1550 nm [30] (Light attenuation) the losses caused by propagation in the channel limit the range of QKD point-to-point links to a few hundreds of kilometers: it would take several years to generate a single bit of a secret key, even in using sources and detectors perfect light, which presents little practical interest.
The detectors. Based on SNSPD (Superconducting Nanowire Single Photon Detectors) are really promising for the quantum communication [20] because of their high efficiency and the low downtime and also because they are commercially available.
Their use has allowed to reach 72 dB of channel losses, which is equivalent to 360 km of standard fiber. [30] [20] The excess of noise is crucial here. Indeed, this noise needs to be lowest possible and this noise needs to be evaluated which is a challenge the more the distance is becoming higher.

The second challenge is maximal rate of secret key generation while using these noisy channels. The produced keys are dependent on the performance of the used detectors as we have previously seen.
The encryption keys produced by QKD can also be used for AES protocols which will hence be robust against a quantum computer attack.
A high rate (high bit rate to generate a secret key) will allow a more frequent update of the encryption key in the symmetric case.

Security challenges: In theory the QKD security aspect looks perfect by its implementation got some imperfections. One explanation has been given [24] by Horace P. Yuen in his paper "SIMPLE EXPLANATION ON WHY QKD KEYS HAVE NOT BEEN PROVED SECURE".
He is saying that the the generation of the shared secret K is perfect and can be used for One Time Pad is possible in theory but not in practice. Indeed, like it has been previously shown the distance "d" which is used to measure the deviation from the perfect key got a negative effect in practice. Indeed, the value of "d" is described as the "probability that Eve may estimate the whole K correctly and hence recover the entire plaintext sequence X from the ciphertext sequence Y, the latter always taken to be known to an attacker".

To address these challenges some new protocols have been developed such as the MDI [22] (measurement device independent) which have been used to demonstrate a QKD system secure against malicious devices. [22]
These protocols offer solutions based on quantum mechanics where the security relies on the violation of a Bell inequality.

Possible usage for future implementation: [23] [26] [25]
As a future implementation QKD protocol could also be used for the wider Internet and to improve its security by being integrated to IPsec and TLS.
IPsec and TLS are using the shared secret method that need to calculate and generates keys for encryption and integrity protection and the analogy with QKD is hence natural as it can be used as the shared secret, the key or as a One Time Pad. [23]
The quantum key can be used to create the shared secret in TLS or IKE SA or can also be used to protect the IKE SA (peers and traffic) and the IPsec or for replacing TLS encryption and MAC keys. Encryption and integrity protection won't need to calculate a shared secret and compute secret keys.

3 Implementation

3.1 Proposal

In this situation the proposed implementation [36] will be based on the QKD protocol and will simulate the qubit exchange via the BB84 protocol.
There will be different parameters to choose:
- Number of qubits
- The number of iterations (to simulate the cascade protocol)
- And whether or not Eve is present

The result of our different choices will be True and/or False with an associated percentage.
True correspond to Alice and Bob having the same key.
False correspond to Alice and Bob not having the same key.
The BB84 protocol has not been covered on the previous section to explain it via the proposed code and will be explained in the evaluation part.

3.2 Code

You can load segments of code from a file, or embed them directly.

Listing 1: Python simulation of protocol BB84 for QKD. - https://github.com/videlanicolas/QKD

```
#!/usr/bin/python
from numpy import matrix
from math import pow, sqrt
from random import randint
import sys, argparse

class qubit():
    def __init__(self,initial_state):
        if initial_state:
            self.__state = matrix([[0],[1]])
        else:
            self.__state = matrix([[1],[0]])
        self.__measured = False
        self.__H = (1/sqrt(2))*matrix([[1,1],[1,-1]])
        self.__X = matrix([[0,1],[1,0]])
    def show(self):
        aux = ""
        if round(matrix([1,0])*self.__state,2):
            aux += "{0}|0>".format(str(round(matrix([1,0])*self.__state,2)) if round(matrix([1,0])*self.__state,2) != 1.0 else '')
        if round(matrix([0,1])*self.__state,2):
            if aux:
                aux += " + "
            aux += "{0}|1>".format(str(round(matrix([0,1])*self.__state,2)) if round(matrix([0,1])*self.__state,2) != 1.0 else '')
        return aux
    def measure(self):
        if self.__measured:
            raise Exception("Qubit already measured!")
        M = 1000000
```

```
        m = randint(0,M)
        self.__measured = True
        if m < round(pow(matrix([1,0])*self.__state,2),2)*M:
            return 0
        else:
            return 1
    def hadamard(self):
        if self.__measured:
            raise Exception("Qubit already measured!")
        self.__state = self.__H*self.__state
    def X(self):
        if self.__measured:
            raise Exception("Qubit already measured!")
        self.__state = self.__X*self.__state

class quantum_user():
    def __init__(self,name):
        self.name = name
    def send(self,data,basis):
        """
        Uso base computacional |0> y |1> para los estados horizontal y vertical.
        Uso base Hadamard |0> + |1> y |0> - |1> para los estados diagonales.
        0 0 -> |0>
        0 1 -> |1>
        1 0 -> |0> + |1>
        1 1 -> |0> - |1>
        """
        assert len(data) == len(basis), "Basis and data must be the same length!"
        qubits = list()
        for i in range(len(data)):
            if not basis[i]:
                #Base computacional
                if not data[i]:
                    qubits.append(qubit(0))
                else:
                    qubits.append(qubit(1))
            else:
                #Base Hadamard
                if not data[i]:
                    aux = qubit(0)
                else:
                    aux = qubit(1)
                aux.hadamard()
                qubits.append(aux)
        return qubits
    def receive(self,data,basis):
        assert len(data) == len(basis), "Basis and data must be the same length!"
        bits = list()
        for i in range(len(data)):
            if not basis[i]:
                bits.append(data[i].measure())
            else:
                data[i].hadamard()
                bits.append(data[i].measure())
        return bits
def generate_random_bits(N):
    aux = list()
    for i in range(N):
        aux.append(randint(0,1))
    return aux

def QKD(N,verbose=False,eve_present=False):
    alice_basis = generate_random_bits(N)
    alice_bits = generate_random_bits(N)
    alice = quantum_user("Alice")
    alice_qubits = alice.send(data=alice_bits,basis=alice_basis)
    if eve_present:
        eve_basis = generate_random_bits(N)
        eve = quantum_user("Eve")
        eve_bits = eve.receive(data=alice_qubits,basis=eve_basis)
        alice_qubits = eve.send(data=eve_bits,basis=eve_basis)
    bob_basis = generate_random_bits(N)
    bob = quantum_user("Bob")
    bob_bits = bob.receive(data=alice_qubits,basis=bob_basis)
    alice_key = list()
    bob_key = list()
    for i in range(N):
        if alice_basis[i] == bob_basis[i]:
            alice_key.append(alice_bits[i])
            bob_key.append(bob_bits[i])
    if alice_key != bob_key:
        key = False
        length = None
        print "Encription key mismatch, eve is present."
    else:
        key = True
        length = len(bob_key)
        print "Successfully exchanged key!"
        print "Key Length: " + str(length)
    if verbose:
        print "Alice generates {0} random basis.".format(str(N))
        raw_input()
        print ''.join(str(e) for e in alice_basis)
        raw_input()
        print "Alice generates {0} random bits.".format(str(N))
        raw_input()
        print ''.join(str(e) for e in alice_bits)
        raw_input()
        print "Alice sends to Bob {0} encoded Qubits.".format(str(N))
        raw_input()
        aux = ""
        for q in alice_qubits:
            aux += q.show() + "   "
        print aux
        raw_input()
        if eve_present:
            print "Eve intercepts Qubits!"
            raw_input()
            print ''.join(str(e) for e in eve_basis)
            raw_input()
            print "Eve's bits."
            raw_input()
            print ''.join(str(e) for e in eve_bits)
            raw_input()
        print "Bob generates {0} random basis.".format(str(N))
        raw_input()
        print ''.join(str(e) for e in bob_basis)
        raw_input()
        print "Bob receives and decodes Alice's Qubits."
        raw_input()
        print ''.join(str(e) for e in bob_bits)
        raw_input()
        print "Alice and Bob interchange basis through Internet and compare their basis."
        raw_input()
    #print "Key obtained: " + ''.join(str(e) for e in bob_bits)
    #print "Efficiency: {0}%".format(str(round((float(len(key))/float(len(alice_bits)))*100.0)))
    return key

if __name__ == "__main__":
    parser = argparse.ArgumentParser(description='BB84 QKD demonstration with Python.')
    requiredNamed = parser.add_argument_group('Required arguments')
    optionalNamed = parser.add_argument_group('Optional arguments')
    requiredNamed.add_argument('-q','--qubits', required=True, help='Number of Qubits.')
    optionalNamed.add_argument('-i','--iterate',required=False, help='Number of iterations.')
    optionalNamed.add_argument('-e','--eve', action='store_true',default=False,required=False, help='Is EVE present?')
    optionalNamed.add_argument('-v','--verbose', action='store_true',default=False,required=False, help='Verbose logs.')
    args = parser.parse_args()
    assert int(args.qubits)
    ret = list()
    if args.iterate:
        assert int(args.iterate)
        N = int(args.iterate)
    else:
        N = 1
    for i in range(N):
        print "############## {0} ##############".format(str(i))
        ret.append(QKD(int(args.qubits),verbose=args.verbose,eve_present=args.eve))
        print "################################{0}################################".format(str(i))
    print "##############################"
```

```
178     print "↩
          ##############################"
179     t = "{0:.2f}".format(float(ret.count(True))*100.0/float(N))
180     u = "{0:.2f}".format(float(ret.count(False))*100.0/float(N))
181     print "True: {0} <{1}%>".format(ret.count(True),str(t))
182     print "False: {0} <{1}%>".format(ret.count(False),str(u))
```

4 Implementation

When Eve is not present and hence spying and altering the qubits, we can notice that whatever the iteration and the qubits the chances of Alice transmitting the qubits to Bob with a perfect Key matching is always true (in this code which doesn't take in consideration the channel noise etc. . .)

1) Let's take the simplest example which is: 1 quibit and 1 iteration :

```
C:\Users\Marc Paridimal\Desktop\QKD-master>python qkd.py -q 1 -i 1  -v
############# 0 #############
Successfully exchanged key!
Key Length: 0
Alice generates 1 random basis.

1

Alice generates 1 random bits.

0

Alice sends to Bob 1 encoded Qubits.

0.71|0> + 0.71|1>

Bob generates 1 random basis.

0

Bob receives and decodes Alice's Qubits.

1

Alice and Bob interchange basis through Internet and compare their basis.

##############################
##############################
##############################
True: 1 <100.00%>
False: 0 <0.00%>
```

Figure 6: **1 quibit and 1 iteration**

We can see that:
- Alice generates a private random bit.
- Then, for each bit she randomly chooses if she encodes it using the computational basis: $|0i, |1i$ or in the Hadamard basis: $|+i, |i$
$|+i = 1/2(|0i + |1i)$
$|i = 1/2(|0i|1i)$
- Alice sends the qubits to Bob via a quantum but not necessarily secure channel

On Bob's side:
- Each time Bob receives a qubit, he randomly decides about the measurement: $|0i, |1i$ basis or in the $|+i, |i$ basis
- He writes down the results and the basis he used:
o If he used $|0i, |1i$ he writes down 0 if he gets —0i and 1 if he gets —1i
o If he used $|+i, |i$ he writes down 0 if he gets —+i and 1 if he gets —i

2) Let's take another example this time:

```
C:\Users\Marc Paridimal\Desktop\QKD-master>python qkd.py -q 3 -i 3  -v
############# 0 #############
Successfully exchanged key!
Key Length: 2
Alice generates 3 random basis.

110

Alice generates 3 random bits.

111

Alice sends to Bob 3 encoded Qubits.

0.71|0> + -0.71|1>   |1>   |1>

Bob generates 3 random basis.

010

Bob receives and decodes Alice's Qubits.

011

Alice and Bob interchange basis through Internet and compare their basis.

##############################
############# 1 #############
Successfully exchanged key!
Key Length: 1
Alice generates 3 random basis.

010

Alice generates 3 random bits.

101

Alice sends to Bob 3 encoded Qubits.

0.71|0> + -0.71|1>   0.71|0> + 0.71|1>   |1>

Bob generates 3 random basis.

100
```

Figure 7: **3 quibits and 3 iterations - 1**

Here again we can see that the number of qubits exchanged are 3 with an iteration of 3. Without Eve they managed to exchange the Key perfectly.
We can also notice that the 3-qubit sent by Alice have been encoded based on the computational basis and in the Hadamard basis.

3) Let's now introduce Eve:

As we can see again Alice has chosen 5 qubit and 2 iterations.
Again, for each bits Alice choose she encode it the random basis.

After encoding it Alice sends the 5 encoded qubits to Bob.

This is where Alice come in the game and intercepts the qubits:
Eve is performing some number of measurements on what she intercepted. The non-Cloning Theorem prove that Eve and Bob can't have a copy of the same qubit. When Eve performs these measurements, she has to pick a basic to measure each qubit. Hence Eve got ½ to guess the good basis.
On the first part it looks like the key exchange was a success. The information sent via the public channel did show that they were both using the same basis. Eve did a choice that

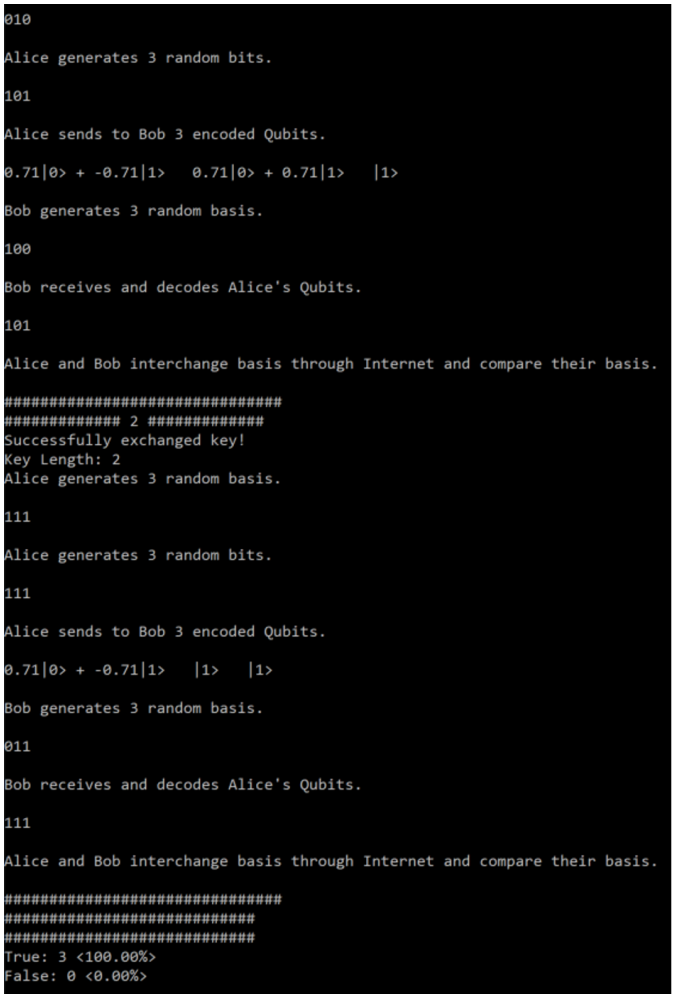

Figure 8: **3 quibits and 3 iterations - 2**

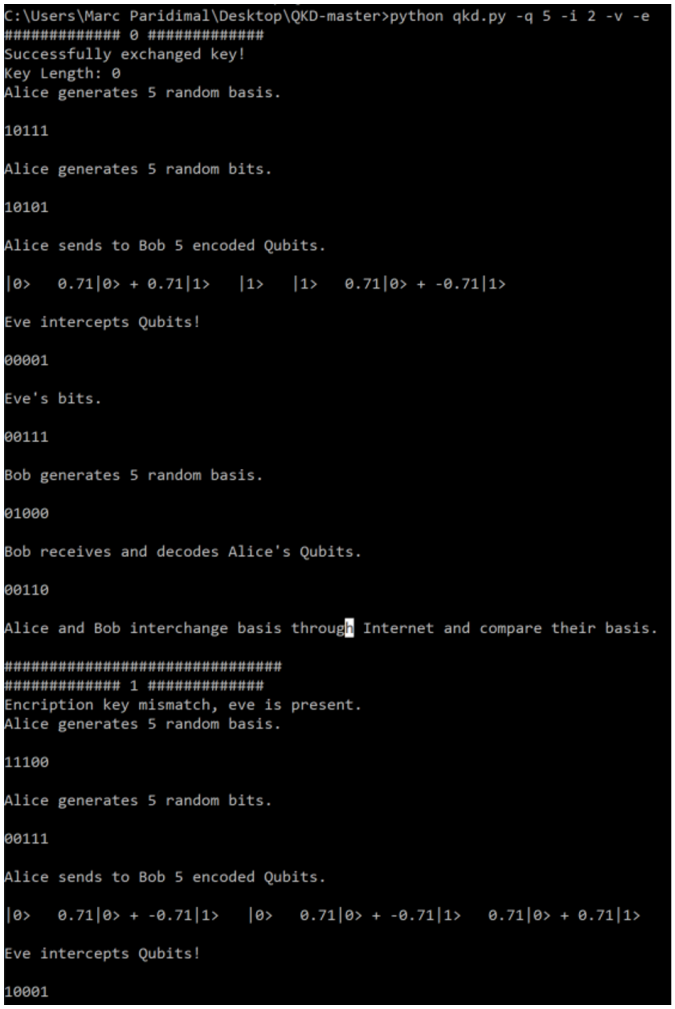

Figure 9: **Eve is spying - 1**

didn't generate enough error while transmitting the qubit and this error rate is acceptable for Alice and Bob.
On the opposite the second iteration failed. This because after dropping the bits where the basis is not matching, the length of the key was under K/2 suggesting Eve was listening and altering the information sent.
Alice sends to Bob an encrypted test message with that key. If Bob can't decrypt the message with his key this will prove the presence of Eve performing a Man In The Middle attack. Moreover, all the different tests performed are showing that the more the key length is big the more the probability of undetected Eve's dropping is low.

5 Conclusion

This paper as covered different models of KDC involving different challenges on different parameters: Key distribution, security performance.
As been shows all the models whether they are using symmetric or asymmetric, distributed or non-distributed model have their advantages and also inconvenient. The inconvenient are mainly for security reasons.
Some answers to the security issues have been brought.
A topic of future research would be to go more in depth on evaluating the security and solutions for enterprise KDC especially around Kerberos but maybe also probably on Azure AD which is using OAuth and doing some evaluation on security.
The other alternative would be around the QKD even if the implementation can be more complex to put in place but the idea would be to use the technics being used to compromise Kerberos and apply the same logic to QQ84 even if the work of working is different.

References

[1] Understanding Cryptography by Christof Paar · Jan Pelzl http://swarm.cs.pub.ro/ mbarbulescu/cripto/Understanding

[2] Decentralized KDC Scheme with Verifiable Outsourcing of Key Updates in Cloud Computing - International Journal of Knowledge Based Computer Systems Volume 5 Issue 2 December 2017 ISSN.: 2321-5623

[3] The Evolution of the Kerberos Authentication Service - John T. Kohl $https$: $//www.researchgate.net/publication/2662482_The_Evolution_of_the$

[4] Nan Zhang, Xiaoyu Wu, Cheng Yang, Yinghua Shen and Yingye Cheng, "A lightweight authentication and authoriza-

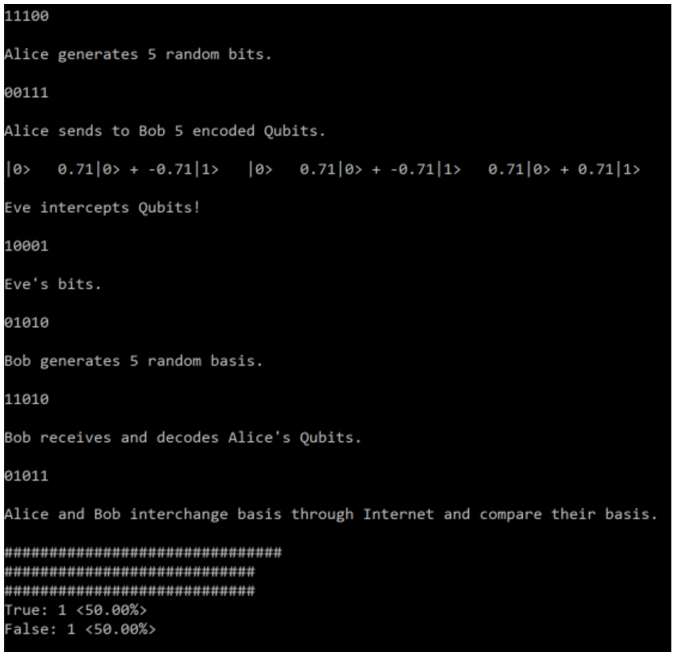

Figure 10: **Eve is spying - 2**

tion solution based on Kerberos," 2016 IEEE Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC), 2016, pp. 742-746, doi: 10.1109/IMCEC.2016.7867308.

[5] https://medium.com/@robert.broeckelmann/kerberos-and-windows-security-kerberos-v5-protocol-b9c804e06479

[6] Iliano Cervesato, Aaron D. Jaggard, Andre Scedrov, Joe-Kai Tsay, Christopher Walstad, Breaking and fixing public-key Kerberos, Information and Computation, Volume 206, Issues 2–4, 2008, file:///Users/mparidimal/Downloads/1-s2.0-S089054010700123X-main.pdf

[7] B. C. Neuman and T. Ts'o, "Kerberos: an authentication service for computer networks," in IEEE Communications Magazine, vol. 32, no. 9, pp. 33-38, Sept. 1994, doi: 10.1109/35.312841.

[8] Research Cell : An International Journal of Engineering Sciences, Issue December 2014, Vol. 1 ISSN: 2229-6913 (Print), ISSN: 2320-0332 (Online) -, Web Presence: http://www.ijoes.vidyapublications.com http://ijoes.vidyapublications.com/paper/Vol11/12-Vol11.pdf

[9] RFC1510: https://www.hjp.at/(st$_a$)/$doc/rfc/rfc1510.html$

[10] RFC4120: https://www.hjp.at/doc/rfc/rfc4120.html

[11] The Kerberos Protocol And Its Implementations - Fulvio Ricciardi (Fulvio.Ricciardi@le.infn.it) INFN – the National Institute of Nuclear Physics Computing and Network Services – LECCE (Italy) Document Version 1.0.3 (26 November 2006) https://zeroshell.org/kerberos/

[12] Five steps to using the Kerberos protocol: https://www.computerweekly.com/news/1280096067/Five-steps-to-using-the-Kerberos-protocol

[13] Meet the Three-headed Hell-hound ... Kerberos - https://medium.com/asecuritysite-when-bob-met-alice/meet-the-three-head-hell-hound-kerberos-149fc1763816

[14] 4th Australian Information Warfare and IT Security Conference 2003 Kerberos V Security: Replay Attacks Kimmo Kasslin , Antti Tikkanen and Teemupekka Virtanen https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.96.7578re

[15] Attacking and fixing the Microsoft Windows Kerberos login service Tommaso Malgherini and Riccardo Focardi Universit'a Ca' Foscari, Venezia http://secgroup.dais.unive.it/wp-content/uploads/2010/08/m0t-krb5-08-2010.pdf

[16] http://www.blackhat.com/presentations/bh-europe-09/Bouillon/BlackHat-Europe-09-Bouillon-Taming-the-Beast-Kerberous-whitepaper.pdf

[17] E. Bouillon: Gaining access through Kerberos, PacSec 2008

[18] An Authentication Protocol based on Kerberos Article in International Journal of Engineering Research and Applications · July 2017 DOI: 10.9790/9622-0707047074

[19] International Journal of Network Security, Vol.12, No.3, PP.159–170, May 2011 159 An Authentication Protocol Based on Kerberos 5

[20] Diamanti, E., Lo, HK., Qi, B. et al. Practical challenges in quantum key distribution. npj Quantum Inf 2, 16025 (2016). https://doi.org/10.1038/npjqi.2016.25 https://www.nature.com/articles/npjqi201625citeas

[21] Device-independent quantum key distribution secure against collective attacks - Stefano Pironio et al 2009 New J. Phys. 11 045021

[22] Experimental quantum key distribution secure against malicious devices arXiv:2006.12863 [quant-ph] - https://doi.org/10.1103/PhysRevApplied.15.034081

[23] arXiv:1004.0605 - Quantum Key Distribution (QKD) and Commodity Security Protocols: Introduction and Integration https://arxiv.org/pdf/1004.0605.pdf

[24] arXiv:1408.4780 - Quantum Physics (quant-ph); Cryptography and Security (cs.CR) Simple explanation on why QKD keys have not been proved secure

https://arxiv.org/pdf/1408.4780.pdf

[25] An approach to securing IPsec with Quantum Key Distribution (QKD) using the AIT QKD software - By: Stefan Marksteiner PID:1210320006

[26] arXiv:1712.02617v1 [cs.CR] 7 Dec 2017 - The Engineering of a Scalable Multi-Site Communications System Utilizing Quantum Key Distribution (QKD)

[27] Diamanti, E Kashefi, E 2017, 'Best of both worlds', Nature Physics, vol. 13, no. 3-4, pp. 3-4. https://doi.org/10.1038/nphys3972

[28] Quantum-computing pioneer warns of complacency over Internet security – Interview of Peter Shor In Nature.com

[29] arXiv:1408.0562 [quant-ph] - Quantum Physics (quant-ph); Superconductivity (cond-mat.supr-con); Cryptography and Security (cs.CR); Optics (physics.optics)

[30] Long-distance quantum key distribution in optical fibre P. Los Alamos National Laboratory, Los Alamos, New Mexico 87545 2 National Institute of Standards and Technology, Boulder, Colorado 80305 3 Albion College, Albion, Michigan 49224

[31] International Journal of Pure and Applied Mathematics Volume 118 No. 24 2018 ISSN: 1314-3395 (on-line version) - Modified Kerberos for IoT by Dynamic Expansion

[32] International Journal of Knowledge Based Computer Systems Volume 5 Issue 2 December 2017 ISSN.: 2321-5623 - Decentralized KDC Scheme with Verifiable Outsourcing of Key Updates in Cloud Computing

[33] Lakshmi V. (2019) Identity Protection. In: Beginning Security with Microsoft Technologies. Apress, Berkeley, CA.

[34] Offerman N and Roobol S. Gitlab repository for LSASS Purge tool. url: https://gitlab. os3.nl/sroobol/lsass-purge-tool (visited on 13/07/2019).

[35] Active Directory Security - https://adsecurity.org/?page_id=1821

[36] https://github.com/videlanicolas/QKD

[37] S. T. F. Al-Janabi and M. A. Rasheed, "Public-Key Cryptography Enabled Kerberos Authentication," 2011 Developments in E-systems Engineering, 2011, pp. 209-214, doi: 10.1109/DeSE.2011.16.