SECURE AUTHENTICATION GATEWAY USING KERBEROS

INFORMATION & NETWORK SECURITY CO405
(COMPUTER SCIENCE AND ENGINEERING)

Submitted by:
Vagish Shanker Yagnik (2K18/CO/381)
Vishesh Jain (2K18/CO/391)

Under the Guidance of:
Miss Anshu Khurana
Phd Scholar

DELHI TECHNOLOGICAL UNIVERSITY
BAWANA ROAD, DELHI - 110042

INDEX

Introduction

Kerberos authentication is currently the default authorization technology used by Microsoft Windows, and implementations of Kerberos exist in Apple OS, FreeBSD, UNIX, and Linux. Microsoft introduced their version of Kerberos in Windows2000. It has also become a standard for websites and Single-Sign-On implementations across platforms. The Kerberos Consortium maintains Kerberos as an open-source project.

Kerberos is a vast improvement on previous authorization technologies. The strong cryptography and third-party ticket authorization make it much more difficult for cybercriminals to infiltrate your network. It is not totally without flaws, and in order to defend against those flaws, you need to first understand them.
Kerberos has made the internet and its denizens more secure, and enables users to do more work on the Internet and in the office without compromising safety.

So basically we are going to mimic the authentication functionality of kerberos and use it over a small distributed system containing some clients who wants to access certain functionality provided by 2-3 servers, and have tried to introduce an extra layer of security by introducing diffie-hellman in kerberos.

Literature Review

Our study started from some of the initial pacers which discussed the approach related to Kerberos, starting from the work of Paulson et al., with the Isabelle theorem prover. Their work included analysis of Kerberos 4 and proofs of various secrecy and authentication properties for that earlier version of Kerberos [2,3].

Our work was done by hand, but we intend to investigate the possibility of automating parts of it in the future. It may even be possible to automatically derive the authentication and confidentiality statements for specific classes of protocols.

Other analyses of Kerberos include one by Mitchell et al. [5] of a basic fragment of Kerberos 5 using the state exploration tool Mur. This is an example of the model-checking (rather than inductive) approach to protocol verification; see also, e.g., [4]. Mitchell et al., found an attack against a restricted fragment of Kerberos 5; this attack was shown not to be achievable in the full protocol, or even in the fragment examined here. Yu et al. [6] described a chosen-plaintext attack against Kerberos 4 and a related oracle-based attack on the original specification of Kerberos 5. These two attacks highlight a limitation of our approach, and of every approach based on a symbolic (Dolev–Yao) model of cryptography: our results here, which show that the design of the basic Kerberos 5 protocol does provide secrecy and authentication, rely on idealized cryptographic primitives, and therefore do not account for possible imperfections of implemented cryptosystems.

Working Of Kerberos

Here are the principal entities involved in the typical Kerberos workflow:

- **Client -** The client acts on behalf of the user and initiates communication for a service request
- **Server -** The server hosts the service the user wants to access
- **Authentication Server (AS) -** The AS performs the desired client authentication. If the authentication happens successfully, the AS issues the client a ticket called TGT (Ticket Granting Ticket). This ticket assures the other servers that the client is authenticated
- **Key Distribution Center (KDC)** - In a Kerberos environment, the authentication server logically separated into three parts: A database (db), the Authentication Server (AS), and the Ticket Granting Server (TGS). These three parts, in turn, exist in a single server called the Key Distribution Center
- **Ticket Granting Server (TGS) -** The TGS is an application server that issues service tickets as a service

Now let's break down the protocol flow.
First, there are three crucial secret keys involved in the Kerberos flow. There are unique secret keys for the client/user, the TGS, and the server shared with the AS.

- **Client/user.** Hash derived from the user's password
- **TGS secret key.** Hash of the password employed in determining the TGS
- **Server secret key.** Hash of the password used to determine the server providing the service.

The protocol flow consists of the following steps:

Step 1: Initial client authentication request. The user asks for a Ticket Granting Ticket (TGT) from the authentication server (AS). This request includes the client ID.

Step 2: KDC verifies the client's credentials. The AS checks the database for the client and TGS's availability. If the AS finds both values, it generates a client/user secret key, employing the user's password hash.

The AS then computes the TGS secret key and creates a session key (SK1) encrypted by the client/user secret key. The AS then generates a TGT containing the client ID, client network address, timestamp, lifetime, and SK1. The TGS secret key then encrypts the ticket.

Step 3: The client decrypts the message. The client uses the client/user secret key to decrypt the message and extract the SK1 and TGT, generating the authenticator that validates the client's TGS.

Step 4: The client uses TGT to request access. The client requests a ticket from the server offering the service by sending the extracted TGT and the created authenticator to TGS.

Step 5: The KDC creates a ticket for the file server. The TGS then uses the TGS secret key to decrypt the TGT received from the client and extracts the SK1. The TGS decrypts the authenticator and

checks to see if it matches the client ID and client network address. The TGS also uses the extracted timestamp to make sure the TGT hasn't expired.

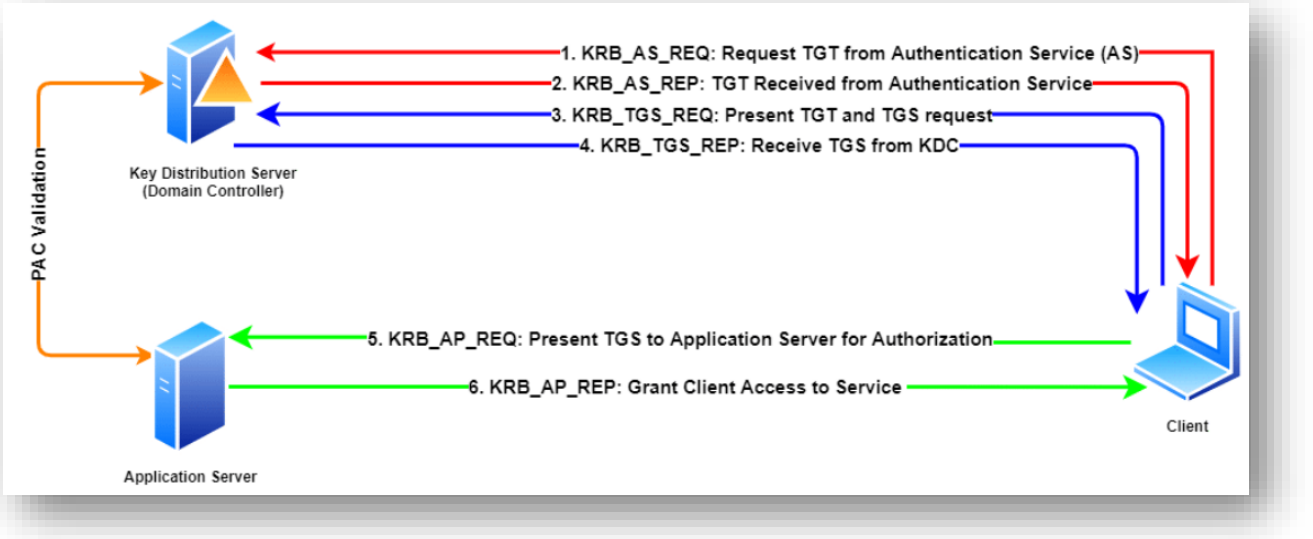
If the process conducts all the checks successfully, then the KDC generates a service session key (SK2) that is shared between the client and the target server.

Finally, the KDC creates a service ticket that includes the client id, client network address, timestamp, and SK2. This ticket is then encrypted with the server's secret key obtained from the db. The client receives a message containing the service ticket and the SK2, all encrypted with SK1.

Step 6: The client uses the file ticket to authenticate. The client decrypts the message using SK1 and extracts SK2. This process generates a new authenticator containing the client network address, client ID, and timestamp, encrypted with SK2, and sends it and the service ticket to the target server.

Step 7: The target server receives decryption and authentication. The target server uses the server's secret key to decrypt the service ticket and extract the SK2. The server uses SK2 to decrypt the authenticator, performing checks to make sure the client ID and client network address from the authenticator and the service ticket match. The server also checks the service ticket to see if it's expired.

Once the checks are met, the target server sends the client a message verifying that the client and the server have authenticated each other. The user can now engage in a secure session.



Limitations of Kerberos

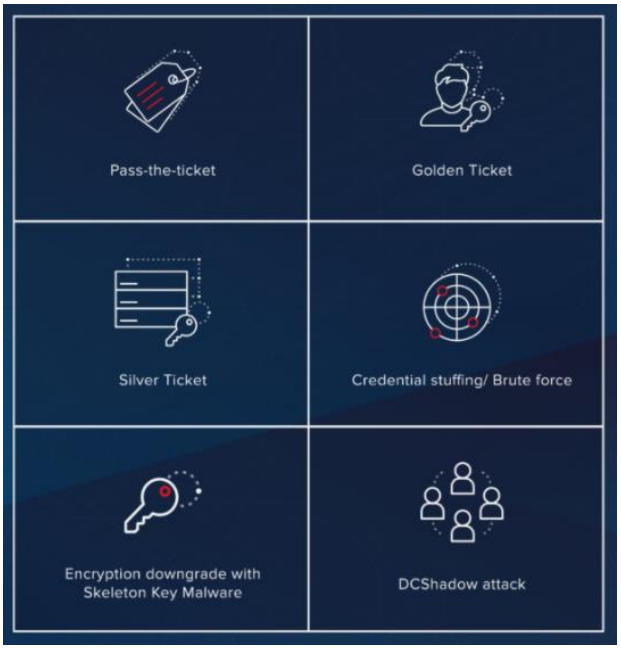
Yes. Because it is one of the most widely used authentication protocols, hackers have developed several ways to crack into Kerberos. Most of these hacks take advantage of a vulnerability, weak passwords, or malware – sometimes a combination of all three. Some of the more successful methods of hacking Kerberos include:

1) **The replay attack:** In order to avoid replay attacks, the protocol uses the timestamp mechanism in the tickets and the authentication operator. Time stamps require clock synchronization in the same net, and it is legal when the time difference of the time stamp verification is not longer than 5 minutes, or be regarded as a kind of replay attack. But in reality, the attackers can sniff the information in advance and modify it, and replay the information in 5 minutes when they have monitored that users request is authenticated, and the timestamp mechanism would be useless.

2) **Dictionary attack:** In step (2), the message sent from AS to C is encrypted by users private key which is produced with user password, and the users private key is dealt with user password by using one-way Hash function. Therefore, the attackers can collect a lot of data which is from AS back to C, and make dictionary attacks. When the user password is not strong, it cannot effectively prevent the dictionary attacks. Once the user password is obtained, the user identity can be pretended to be a random one.

3) **Key storage problem:** Traditional Kerberos uses a symmetric cryptosystem, so the shared keys between customers and KDC, between application server and KDC, between KDC and distant KDC have to be established and maintained. And in the expanding Internet using Kerberos protocol, keys management and maintenance become the most difficult problem to solve.

4) **Malware attack:** Kerberos authentication protocol depends on the absolutely trust in Kerberos software, and attackers can model Kerberos protocol to design client software which has built-in listeners. And then the attackers mislead users to install malware to listen to the users operations including password inputting, and what is worse, the attackers can directly invade KDC, and disguise as KDC to complete the middleman attack.



5) **The authentication forward problem:** Kerberos 5 adds a function authentication forwarding, and it is that when client is granted to access a server. It can let the server be client status to apply a different remote server service. In this way there exists the possibility of the springboard attack. Kerberos 4 does not support the authentication forwarding, so there is no such problems in it.

Innovation

In this project, we have tried to address 2 main limitations of the kerberos which are as follows:

- **Dictionary Attack**:
  In step (2), the message sent from AS to C is encrypted by users private key which is produced with user password, and the users private key is dealt with user password by using one-way Hash function. Therefore, the attackers can collect a lot of data which is from AS back to C, and make dictionary attacks. When the user password is not strong, it cannot effectively prevent the dictionary attacks.

So, to make the message passing secure from Authentication server to client we are introducing a way using diffie-hellman algorithm without adding more latency to the system.

Steps that we changed in Kerberos Authentication system:

1. Clients should have a secret private key (which would be generated from client's password using SHA256 algorithm) , say CPvtKey and one random key RKey would be generated at run time.

2. Using DH, a public key say CPubKey is created which is a combination of the client's secret key and random key.

3. Now when client sends request to authentication server, one more json object would be added containing both CPubKey and Rkey.

4. Authentication server on receiving the request first create a private Symmetric Key, say SymmKey that would be used for encryption.

```
    export interface KeyEx{
      publicKey: string,
      random: string
}
```

Results

Client to AS

req

```
Authentication request sent to Authenticator.....

Data Sent:

A : {
  username: 'vagish',
  serviceId: 1,
  userIpAddress: '::1',
  requestedLifeTimeForTGT: { value: 2, unit: 'min' }
}

 Key Exchange : {
  publicKey: '2f916e69b93065cea95d7c076c44bdb024bbd4fdb8df3aa9cce682896ae94656
FLFHDpTGmV',
  random: 'FLFHDpTGmV'
}
```

AS to client

```
Authenticated done by Authenticator Server :)

Encrypted Response from Authenticator Server:
 U2FsdGVkX1+JkvJqx2m2kRP5wUMYZsu1T/PuvErxk8Q+6e/s2OH5M0TDBWNAXWSDdMgZyKrJQ6qlU
gX+PW3DalUwiIAZAYDCvD7ttsPaUTWo/AdKdby5HzEiudpRmKhbnOwCvSuqS9LGUOAV0c6KgjR2Rnd
74o38SwrPcvigh3eHfp7/OAZIbJKQps5ATpgyDDpzpHk3eMx3APImwgzZlNxZd/6AbLCsypJ/SFXXj
zEjtHqFd8xdfAbgQB9scESwvRNRy8HZEgiGGI2agfT60Gic1I9crtLpH22QUB1bTag7N229Savy/jn
lMhWvcG7YNG/kvqo5DqulZWDEUnifCDiscjFmaSsLO5A7cMjKqsoxs+3nB2OqEgzs8SLviZx6sjSM/
MjbpNJQLA2xecW5LLypNRjxxl0lAdSsvRZwVk3tbeoNcoe9PjJg7Fey3ohIj0aOxI1lOfebw3oHl3q
Z9Nmx38ygE9dsQFyhVAVxHlyb6JQixDed2+CZyZNlVO5j8kbDGa0xCjR4GMwGid7cbZybTfzx2jn60
Cn/aD8av9zvFrq1nuEKPjtqNl+t5p+Xz2YWhrK79AAd6Kw4/xW1cR1IrKQ/lDzKZLuIkhJdi35nf9Z
ec3oG79FrP+gGJiq13s0AATOCebqT00MtCs2moA==

Decrypted packet using Diffie-Hellman:  {
  cipherB: 'U2FsdGVkX1/NEJdOTk/MBntsmCXzquH8DmP6nBLme9paRWnSIig/rc1h9YWUdVithF
gHjnpfLlf3t+lo8ZqqgCweYgp24edAXqlkImgqDXBMkg1u0dYOndK4+fDgMLUF2eIa4z2C0NIH5kfa
gH0KOP4aeRKQ52nBBE95p0K7e6sH4CRe9Z+p5yTR8MKyK/mA',
  cipherTGT: 'U2FsdGVkX18Bb/6KZ+dBsEVN9En4XK2mPPFP6OZy9NwPGKjhzYrJozyfCUJnIhP6
2FOV+Yo0jQOiZ8btPWUEGdB1xYNQQ43Gg4HRLumY6WHhJGwr929i0yiVKgGyL3fp+FEQKgV4cbxxZM
8DUK+WhfrQhp2kTh5QNTUWByMexWk8Tv7aSiz38QnIW5AaGTxDFgwL6AF06+9MhMGjG5CQeYT+e4d2
moB9XslD4tVDUwRv5BGsi61cJ3YRV7EJCWbS'
}
```

Authentication Server

```
Authencation request recieved by Authenticator at  1636983580228

A :  {
  username: 'vagish',
  serviceId: 1,
  userIpAddress: '::1',
  requestedLifeTimeForTGT: { value: 2, unit: 'min' }
}
Executing (default): SELECT `hashedPassword` FROM `customers` AS `customer` WH
ERE `customer`.`username` = 'vagish';

 Response to be send to client : {
  cipherB: 'U2FsdGVkX1+kMzL+zDevxP6LQqyK+A3wkFCKP/mCWwQZcEcc0GCVGMRj2KjaYNdcKU
5N4/7V4W9wTy/g04Oy008Z8HGks8OpIbzqzFd7LobV8OnWUDCKnp2ffHsxye8h/D4EWBoQIAAfA1ip
p3laO1ixTCqDJcvojdF1Yi7/vzmSr9xbAao2Sz5XlWGI8v+f',
  cipherTGT: 'U2FsdGVkX190xjQMqguv0n6C3EAvp4l7r4Q80OTDOg8zbea5whz1VYVcA5XLWJnj
IqnO7seh+Ylrg2Z5T7hMCyvQGvBgNJbVxgkt+yZ1aIrW3t5O5s7xzXhG5+ddzxRgIpbgI4JTIkHXHM
4bOcBR8b7pjTHsQL5vh+MWzsTrwoLkdULqJq2E4sjMj8PoxM5daF7nW9MyaBsFlOel84Hfo+Zzuv1Q
pQIjX8GbBFhucEx+88ywnlb0BqjWpeCWx/rz'
}
```

```
After encryption using Diffie-Hellman : U2FsdGVkX18baYQWZifZ4bWXMw4EG3fpJ0bFtW
GABJ5xSe5UUgjHjxsn+KQSAkUt4uPGuQo2Oay7h+zwb/aCGAPSQSkVqGMKVtmQLTp3JhKA+tWCsycH
kclrAAbZflV+L0z5gR1rvF0GN/jR4lGMFVLTKI0M9qslvGoghgn2FNRQ2kxtBigrtvdPXemNcFy6b3
Flk0CGFPSevwquxLrsH2/n6WbKQZgUhC4BxpuTj9tPBFjZG1FLXL+SQDKzc0evcPxf/n9LoA9tircT
e7Bd1h40FdIpR7n50zL8Mozlzi9t8DjRUBKQqOWHOH0b2lBHjVXcJBarc+Wou+8t2iRwspwVu7TWAf
pRCXo5IxS4tZRkohUWsFJNbB8vyhyQ9cItH+OC78DyaJENiAWHfvWJK13OYbHlV6onFA6Z8v5ESJrN
oz1nMumSxxZtIOdx8NZxI+DGYBjvnSPBkdM85+bgD5b2TUUVUrz4qa/g7y1UV8Bi8cQkKBCPZGVTPB
L8tx4/YvIoBtzUpB5ZYwrrjjxxLfm2b9dE7cqTomrz68j1xlBG9k2N0b9se4UoLYvnhKaQLdQ5T2//
vfcyppwlwZX14rKpyNgnyBbIVh7AYbrDbnhsgLW6NxzhO5AjR23usCXQ2yfzldZlpN7iGwqnuiu8YA
==

Client verified from user authenticator....
```

Client to TGS

```
Authentication request sent to Ticket Granting Server.....

Data Send to TGS : {
  TGT: 'U2FsdGVkX18Bb/6KZ+dBsEVN9En4XK2mPPFP6OZy9NwPGKjhzYrJozyfCUJnIhP62FOV+Y
o0jQOiZ8btPWUEGdB1xYNQQ43Gg4HRLumY6WHhJGwr929i0yiVKgGyL3fp+FEQKgV4cbxxZM8DUK+W
hfrQhp2kTh5QNTUWByMexWk8Tv7aSiz38QnIW5AaGTxDFgwL6AF06+9MhMGjG5CQeYT+e4d2moB9Xs
lD4tVDUwRv5BGsi61cJ3YRV7EJCWbS',
  D: '{"serviceId":1,"requestedLifeTimeForTGT":{"value":2,"unit":"min"}}',
  UserAuthenticator: 'U2FsdGVkX1/qsez7cEynBSSVXyfN02fgTaN7gTns46TDd7FWDIGGVh6y
fuAybLKp+fLh2C8UeRNIbdNeg4853evtpC+1QGY8SAVSU6oHbok='
}

Authenticated done by TGS :)

Response from TGS :  {
  cipherF: 'U2FsdGVkX1/7TKjHHyfWpvloPc9RGDexfhpqiLdGGT+Mk7RyFVM6JFYtg1T6Ed4orr
PGTxAMUyUdHh/XG61C8/63nzO5uhkLnpn1DCCuZftHmNTyQcebRoP6toTCrULSCrfb3McRPnLf/dw/
h/qllxv4fgG88nf2dkfLiCo04IO6kp/WQzpzXEPAnGWJ7pZU',
  cipherServiceTicket: 'U2FsdGVkX18reNB0jYyDQKJOl88lJIDCRSxfn2VF/EIsmESR/Q+jUB
M9/1lpkex/Fy84BZo3k/migdxa+2cwzcRX33SMQZKiybCOAScLMtBTAMHJxHOBltHI4UIRpGKZvh26
snbKv7rJbjoLfhnwFQy0+gUTTdP1e0QL6E7B/BXdPbiUd9eH9t5Kn+sX5aACALMDi32ycoUt4jwaRC
IH2G8PABWqth2PC40AAcLTiv1wNL/T1tKynLYCU382UiCo'
```

TGS

```
Authencation request recieved by TGS at  1636983876860

Encrypted recieved data:

D :  {"serviceId":1,"requestedLifeTimeForTGT":{"value":2,"unit":"min"}}

TGT :  U2FsdGVkX18Bb/6KZ+dBsEVN9En4XK2mPPFP6OZy9NwPGKjhzYrJozyfCUJnIhP62FOV+Yo0jQOi
Z8btPWUEGdB1xYNQQ43Gg4HRLumY6WHhJGwr929i0yiVKgGyL3fp+FEQKgV4cbxxZM8DUK+WhfrQhp2kTh5
QNTUWByMexWk8Tv7aSiz38QnIW5AaGTxDFgwL6AF06+9MhMGjG5CQeYT+e4d2moB9XslD4tVDUwRv5BGsi6
1cJ3YRV7EJCWbS

User Authenticator :  U2FsdGVkX1/qsez7cEynBSSVXyfN02fgTaN7gTns46TDd7FWDIGGVh6yfuAyb
LKp+fLh2C8UeRNIbdNeg4853evtpC+1QGY8SAVSU6oHbok=

Decrypted recieved data:

TGT :  {
  username: 'vagish',
  TGSid: 128,
  timestamp: '2021-11-15T13:44:36.764Z',
  userIpAddress: '::1',
  lifetimeForTGT: { value: 2, unit: 'min' },
  TGSsk: 'KEJFRYFj5M3qtYAO'
}
```

```
User Authenticator :  { username: 'vagish', timestamp: '2021-11-15T13:44:36.835Z' }szfxgfg
hgffssasfv---------------------- ::ffff:127.0.0.1 ::1
Executing (default): SELECT `id`, `serviceId`, `serviceSecretKey`, `createdAt`, `updatedAt
` FROM `services` AS `service` WHERE `service`.`serviceId` = 1;

client verified from tgs....
```

Client to Server

```
Authentication request sent to Server.....

Data Send to Server : {
  serviceticket: 'U2FsdGVkX18reNB0jYyDQKJOl88lJIDCRSxfn2VF/EIsmESR/Q+jUBM9/1lp
kex/Fy84BZo3k/migdxa+2cwzcRX33SMQZKiybCOAScLMtBTAMHJxHOBltHI4UIRpGKZvh26snbKv7
rJbjoLfhnwFQy0+gUTTdP1e0QL6E7B/BXdPbiUd9eH9t5Kn+sX5aACALMDi32ycoUt4jwaRCIH2G8P
ABWqth2PC40AAcLTiv1wNL/T1tKynLYCU382UiCo',
  userauthenticator: 'U2FsdGVkX1/BvpjhL1l9icmgDHUPfuqMy/uVhRvC7ryj5GvGaXDXfhcK
JXIM9JcUq/pdc4x9cs3Bi8gBnc8o6gT1b4OE0CdU87mXiWALTQ8='
}

Authentication done by Server :)

Response from Server :  { timestamp: '2021-11-15T13:44:36.947Z' }
```

Server

```
Authencation request recieved by Server at  1636983876934

Encrypted recieved data:

User Authenticator :  U2FsdGVkX1/BvpjhL1l9icmgDHUPfuqMy/uVhRvC7ryj5GvGaXDXfhcKJXIM9
JcUq/pdc4x9cs3Bi8gBnc8o6gT1b4OE0CdU87mXiWALTQ8=

Service Ticket :  U2FsdGVkX18reNB0jYyDQKJOl88lJIDCRSxfn2VF/EIsmESR/Q+jUBM9/1lpkex/F
y84BZo3k/migdxa+2cwzcRX33SMQZKiybCOAScLMtBTAMHJxHOBltHI4UIRpGKZvh26snbKv7rJbjoLfhnw
FQy0+gUTTdP1e0QL6E7B/BXdPbiUd9eH9t5Kn+sX5aACALMDi32ycoUt4jwaRCIH2G8PABWqth2PC40AAcL
Tiv1wNL/T1tKynLYCU382UiCo

Decrypted recieved data:

User Authenticator :  { username: 'vagish', timestamp: '2021-11-15T13:44:36.895Z' }

Service Ticket :  {
  username: 'vagish',
  timestamp: '2021-11-15T13:44:36.874Z',
  userIpAddress: '::1',
  lifeTimeForServiceTicket: { value: 2, unit: 'min' },
  serviceSessionKey: 'Uvc8ZM0KpwknkaZE'
}

Client verified from server....
```

References

[1] Formal analysis of Kerberos 5 Frederick Butlera, Iliano Cervesatob,∗,1, Aaron D. Jaggardc,2, Andre Scedrovd,3, Christopher Walstadd,3

[2] G. Bella, L.C. Paulson, Using Isabelle to prove properties of the Kerberos authentication system, in: H. Orman, C. Meadows (Eds.), Proc. DIMACS'97, Workshop on Design and Formal Verification of Security Protocols (CD-ROM), 1997,

[3] G. Bella, L.C. Paulson, Kerberos version IV: inductive analysis of the secrecy goals, in: Proc. ESORICS'98, Lecture Notes in Computer Science, Vol. 1485, Springer, Berlin, 1998, pp. 361–375.

[4] E.M. Clarke, S. Jha, W.R. Marrero, Using state space exploration and a natural deduction style message derivation engine to verify security protocols, in: Proc. IFIP Working Conference on Programming Concepts and Methods (PROCOMET), 1998, pp. 87–106.

[5] J.C. Mitchell, M. Mitchell, U. Stern, Automated analysis of cryptographic protocols using Murϕ, in: Proc. IEEE Symp. Security and Privacy, IEEE Computer Society Press, Silver Spring, MD, 1997, pp. 141–153.

[6] T. Yu, S. Hartman, K. Raeburn, The perils of unauthenticated encryption: Kerberos version 4, in: Proc. NDSS'04, 2004

[7] Authors Notes by MIT https://web.mit.edu/kerberos/

[8] Kerberos as used by Microsoft https://docs.microsoft.com/en-us/windows-server/security/kerberos/kerberos-authentication-overview

[9] GitHub Repository by MIT https://medium.com/@robert.broeckelmann/kerberos-and-windows-security-kerberos-on-windows-3bc021bc9630

[10] Comparitive Study https://medium.com/@robert.broeckelmann/kerberos-and-windows-security-kerberos-on-windows-3bc021bc9630

[11] Secure Key Distribution Prototype Based on Kerberos https://sci-hub.ee/10.1007/978-3-030-48149-0