Rupanshoo Saxena (2019096) | Saatvik Bhatnagar (2019097)

Network Security - Assignment 4
Report

Topic

Project 1 (A1 = 9096, A2 = 9097, k = (9096 + 9097) mod 3 = 1)

Project 1 - Digitally signed degree certificates:

This application relates to building a web server that responds with a degree-certificate and grade-card whenever someone requests for it. The request must contain the graduate's name & unique roll-number. The degree-certificate and grade-card (possibly in PDF format) is suitably digitally signed by the university authorities, together with the current (and correct) date & time.

1. How and where do you get the correct GMT date and time? Is the source reliable and the GMT date and time obtained in a secure manner?
2. How do you ensure that only the graduate is able to download it (by providing information beyond the roll no, such as date of birth, home pin code, etc.?
3. Should the graduate decide to share the document with others, how can one trace the origin of the document (could watermarks be useful?)?
4. Do we need to have access to public-keys, and if so how?

Bonus points if you address this issue: How do you get the document to be digitally signed by two persons (say the Registrar and the Director?)

1. Digitally signed degree certificates:

For the given project we are required to establish a web server that provides students with their digitally signed degree certificates and grade cards.

1.1 Requirements

Requirements to be fulfilled for the project:

1. **Integrity of timing** i.e. getting correct GMT date and time securely, making sure that the source is reliable → achieved using Cristian's algorithm
2. **Ensuring Confidentiality** i.e. only the graduate is able to download it → achieved using Symmetric private-key cryptosystem.
3. **Source/ origin of message** i.e. tracing origin of the document must be possible → achieved by digitally signing the documents i.e. the non-repudiation property of digital signatures

4. **Accessing public keys securely** → Public keys will be accessed securely using a KDC securely.

2. Web Server

2.1 Entities

1. The Client (Student side)
2. The Certificate Granting Server
3. The KDC
4. The Clock Server (provides synchronized process time)

2.2 Synchronized time - Cristian's Algorithm

It is essential to maintain integrity of time i.e. getting the correct GMT date and time securely and making sure that the source is reliable in message authentication. This requirement is fulfilled by synchronization of clock which is done using Cristian's algorithm.
Reference Link: https://www.geeksforgeeks.org/cristians-algorithm/

Cristian's Algorithm is used to synchronize time using a clock time server. The algorithm follows the following formula:

$$T_{client} = T_{server} + (T_1 - T_0) / 2$$

Here,
T_{client} → the final synchronized time
T_{server} → clock time returned by the clock server
T_0 → the time at which the client sends the clock server a request
T_1 → the time at which the client receives the response sent by the clock server

The time at the client-side differs from actual time by at most $(T_1 - T_0)/2$ seconds. Using the above statement we can draw a conclusion that the error in synchronization can be at most $(T_1 - T_0)/2$ seconds.
Hence,
latency or error belongs in the range $[-(T_1 - T_0)/2 , (T_1 - T_0)/2]$

For each request that is carried out from any party, a timestamp is always a part of the request. Hence, instead of using an unreliable source, we use the Clock Server and calculate the synchronized time which gives us the correct GMT time and date in a secure manner and without latency and this synchronized time is then used as the timestamps and sent in requests which gives the synchronized time as to when request was sent.

2.3 Establishing a connection between the degree-granting server and the student client

First and foremost, we need to establish a connection between the degree-granting server and the student requesting for the digitally signed degree certificate and grade card.

1. A key-pair <PU, PR> using RSA is generated for both the degree-granting server and the student client.
2. A handshake takes place between the degree-granting server and the Key Distribution Centre i.e. the KDC where-in the degree-granting server sends its ID (ID_{Server}) and its public key (PU_{Server}) to the KDC. The KDC stores public keys of all parties and sends them to other parties on request.
   So, after the handshake with the degree-granting server, the KDC has stored the public key of the server mapped with its ID.
   As a confirmation of the handshake, the KDC also sends its own ID_{KDC} and PU_{KDC} to the degree-granting server.
3. A similar handshake also takes place between the student client and the KDC where-in the student client sends its ID_{client} and PU_{client} to the KDC and the KDC stores it for future use. As a confirmation of the handshake, the KDC also sends its own ID_{KDC} and PU_{KDC} to the student client.

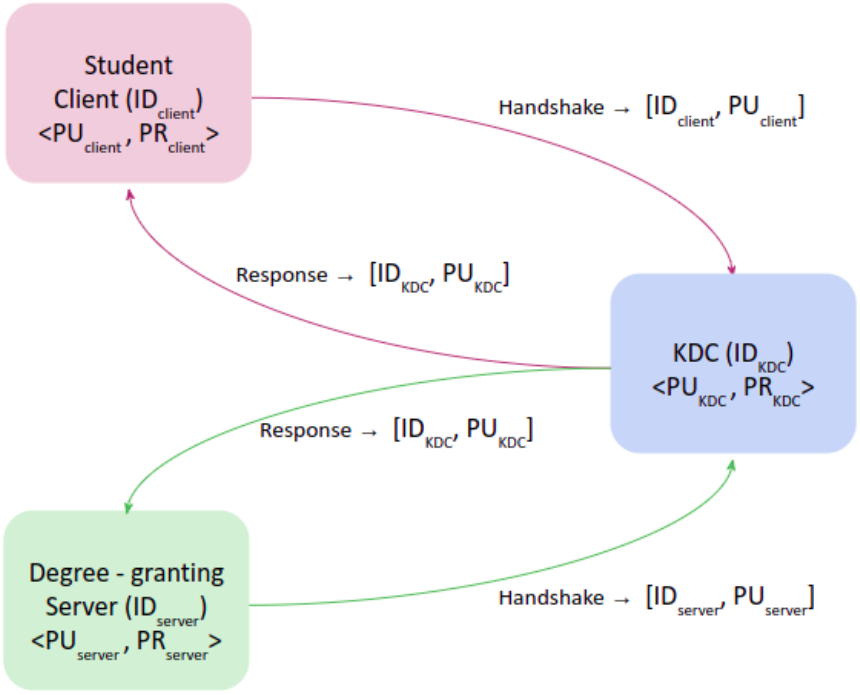

Fig 1: Both parties carry out a handshake with the KDC and receive confirmation on their public keys being received and stored by the KDC

4. Now, the KDC has the public keys of both the server and the client and keeps on listening for any other requests - which may be handshake requests or entities requesting for public keys of other parties through the KDC.

5. The student client sends a request to the KDC to ask for PU_{server} so that it is able to send the degree-granting server a request for obtaining their own digitally-signed degree certificate and grade card. The KDC responds with its own ID and encrypted requested information which consists of the ID of the requested party and the requested party's public key. This requested information is encrypted using the student client's public key which ensures confidentiality.

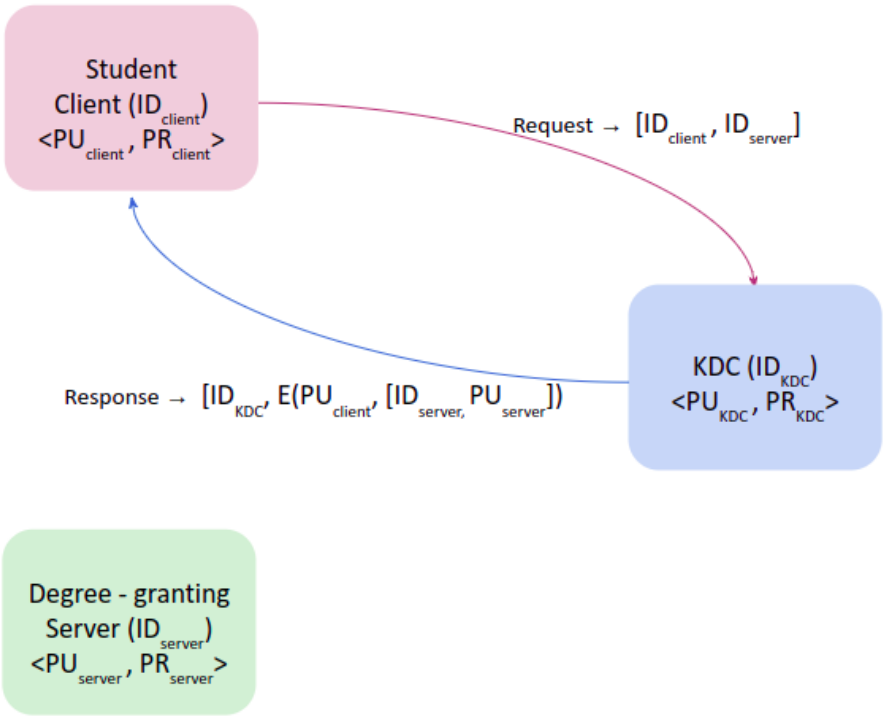

Fig 2. The student client requests the KDC for the degree server's public key and receives requested information confidentially.

6. The Student client decrypts the response received from the KDC using its own private key PR_{client} and is able to access the public key of the degree-granting server. It can now communicate with the degree-granting server.

We see from above that we require public keys. They are essential to establish a connection between the student client and the degree-granting server. We encrypt the messages being sent by them to each other using their public keys which ensures confidentiality of the messages being sent.

All the public keys are available with the KDC (Key Distribution Centre) and all parties get access to other's public keys by requesting the KDC. The KDC sends them the relevant information.

2.4 Requesting for Degree-certificate and Grade card

1. The degree-granting server receives an encrypted request from a client which they decrypt using its own PR_{server}. They obtain the following information from the decrypted request:

```
{
        name,
        rollno,
        password
}
```

name: name of the student whose degree we're requesting from the server
rollno: unique roll number of the student
password: secret password that is used to authenticate the student.

2. The server verifies if the password and roll number match the student's name who is requesting the degree-certificate. If the details don't match then we show an authentication error.
3. If the password matches, the server starts the digital signature process.

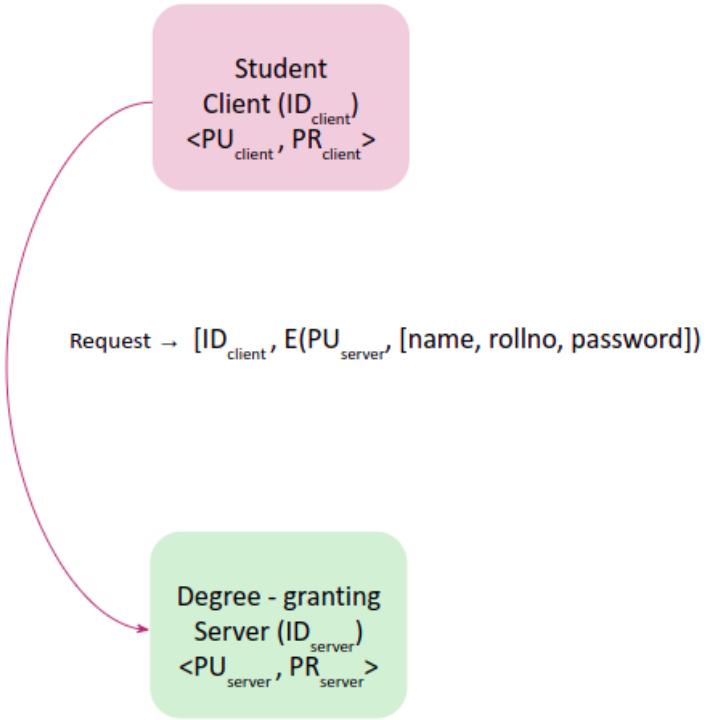

Fig 3. Degree server receives a request from the client and authenticates the student details

2.5 Creating Private Symmetric Key ‘K’

1. Now that the degree-granting server has verified the student details successfully and has the password, it generated a secret **symmetric private key ‘K’** using the secret password sent by the student securely.
   This private symmetric key (K) is shared between the degree-granting server and the student client. This private symmetric key is essential since it serves the following purpose:
   - The private symmetric key K ensures **message integrity.** The client receives data encrypted with the symmetric key and uses the same private symmetric key to decrypt the file. This ensures that the encrypted file has not been tampered with and is the same file that had been sent from the server.
   - The private symmetric key K also ensures **confidentiality**. Encrypting the data with the private symmetric key ensures that only the intended client who possesses the key will be able to decrypt it since the key is private and only shared between the 2 intended parties.

So, in case of our project, we ensure that only the intended student is able to download the degree certificate by encrypting the message using their private symmetric key K which ensures confidentiality and no one other than the student will be able to decrypt the file and hence won’t be able to access or download it.

2. So, now that the degree-granting server has generated the special key using the secret password, it sends this private symmetric key over to the student client securely.
3. To send it securely, the degree-granting server asks the KDC for the student client’s public key PU_{client}. The KDC responds to this request and sends the degree-granting server the requested student’s information in an encrypted manner.
4. The degree-granting server then decrypts the message received from the KDC to obtain the student’s PU_{client} , encrypts the private symmetric key K using the PU_{client} and sends it to the student client.
5. The student client receives the encrypted private symmetric key K, decrypts it using its own private key PR_{client} and obtains the private symmetric key. It sends an acknowledgement by encrypting it with ‘K’ so that the degree-granting server can decrypt it and verify that the private symmetric key was successfully delivered.

6. The degree-granting server on decrypting the acknowledgement with K verifies the name and rollno with the earlier obtained values and on successful verification, carries on to send the degree certificate and grade card.

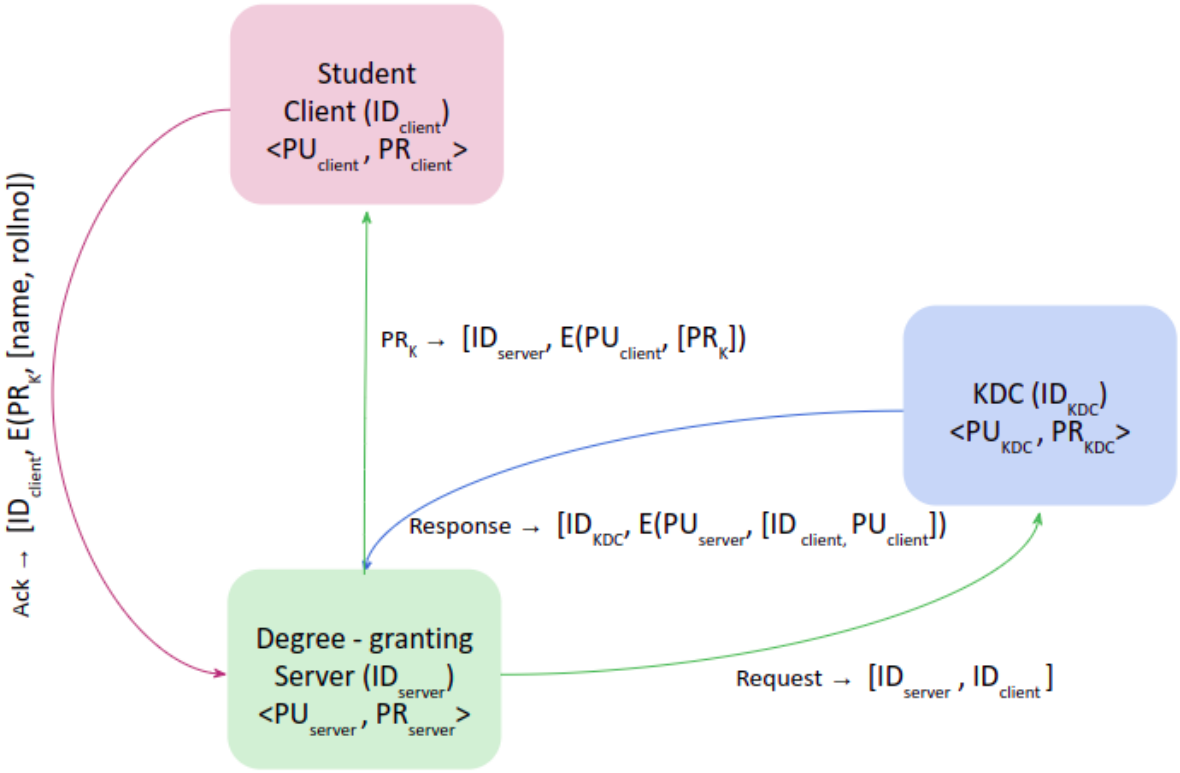

Fig 4. Degree server verifies student, shares private symmetric key and receives acknowledgement from student

2.6 The Digital Signature Process

We are dealing with 2 documents here which are - the degree-certificate and the grade card. Both these files are taken to be text files (.txt). To start the digital signature process, we first need to change the files into a usable format.

We first get the bytes of the text in the file since the hashing function we are using uses bytes to hash the message and not simple text or string input.

Later, when we send the original message also as part of the complete encrypted message, we encode the file in base-64. This is done because the bytes array is not json serializable and we need a string for the same.

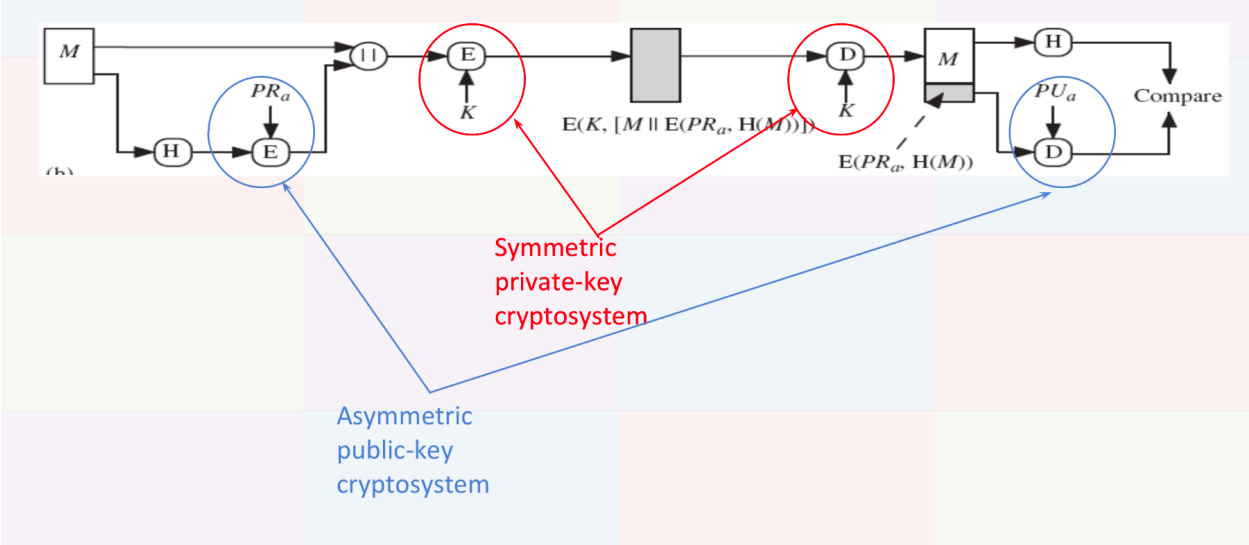

Fig 5. The Digital Signature

Following the above diagram, the digital signature process happens in the following steps:

1. The string message obtained from the text files of the grade card and the degree certificate is converted into bytes and is sent into a hashing function (SHA256). After the message is hashed, it is encrypted using the private key of the signing authority (the institute) i.e. PR_{server}. Let's call this the message digest.
2. The original message is then encoded into base 64 and along with the encrypted hashed message is together then encrypted by the private symmetric key K that is shared between the degree-granting server and the student client.
3. This final encrypted message encrypted with K is then sent from the degree-granting server to the student client.
   This is the format of the message being sent from the degree server to the student client:

```
{
        grade_card_file_digest,
        degree_certificate_file_digest,
        grade_card_file,
        degree_certificate_file
}
```

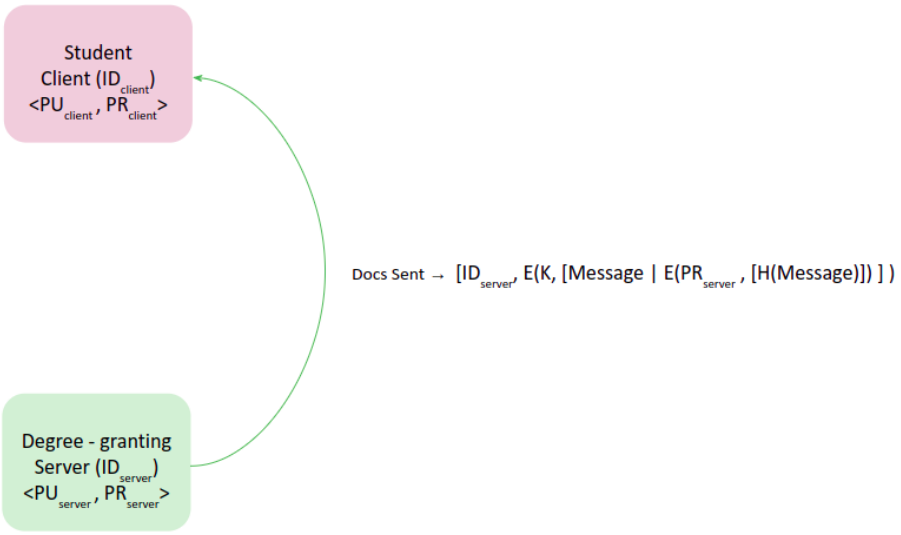

Fig 6. The degree-certificate and grade card are sent from the server to student client

4. The student client on receiving this, firstly, decrypts it using the private symmetric key K. Decrypting it successfully ensures that the message was not tampered with during transit since only these two parties possessed that private symmetric key. **This ensured that no other student or party was able to download the files sent to this particular student client who requested for it.**
5. After decrypting the message using K, the message consists of 2 parts - the actual encoded message file and the message digest i.e. the encrypted hash form of the message. To verify that the files received were actually sent by the degree-granting server, the student client decodes the original file and hashes it. It also decrypts the encrypted hashed message using the public key of the degree-granting server which also gives a hash.
6. To verify the authenticity, both the hashes should match, if they do then the message sent was authentic and it is verified that it hasn't been tampered with and has been sent by the degree-granting server itself.

Since after decrypting, both the hashes match on the student client's side, this means that the message received is authentic and the client carries on to download the files.

Successfully digitally signing the documents provides 4 strong security protocols :

- authentication
- message integrity
- non-repudiation
- confidentiality

Digital signature uses asymmetric public key cryptography which in turn ensures non-repudiation. When the sender creates a digital signature by signing using their private key, this can be verified by anyone else as it can be decrypted by the sender's public key which is available publicly. Hence, it ensures that the sender cannot deny having sent the message and hence traces every message sent back to the sender. In this way, even if the student further shares the obtained degree certificate and grade card, it can be traced back to the sender. Any sort of changes to the file like another party adding a watermark etc would lead to the original file being changed and hence couldn't be traced back to the sender, thus not being authentic.