Networks and Internet Applications

Continuous Assessment Test – CA3

Nicolas D'Alessandro Calderon

Answers

1. **Polyalphabetic encryption**

C1 - Caesar cipher (offset +7)

Plaintext Letter:	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
Ciphertext Letter:	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g

C2 - Monoalphabetic cipher

Plaintext Letter:	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
Ciphertext Letter:	p	g	i	r	k	u	m	f	b	t	y	v	o	z	q	x	h	l	e	j	a	c	w	n	s	d

Polyalphabetic cipher using C1 and C2 with repetition pattern [C1, C1, C1, C2, C2, C1, C2]

Plaintext Letter:	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
Alphabet:	C1	C1	C1	C2	C2	C1	C2	C1	C1	C1	C2	C2	C1	C2	C1	C1	C1	C2	C2	C1	C2	C1	C1	C1	C2	C2
Ciphertext Letter:	h	i	j	r	k	m	m	o	p	q	y	v	t	z	v	w	x	l	e	a	a	c	d	e	s	d

Plaintext:	n	i	c	o	l	a	s		d	a	l	e	s	s	a	n	d	r	o
Alphabet:	C1	C1	C1	C2	C2	C1	C2		C1	C1	C1	C2	C2	C1	C2	C1	C1	C1	C2
Ciphertext:	u	p	j	q	v	h	e		k	h	s	k	e	z	p	u	k	y	q

Plaintext:	n	i	c	o	l	a	s	d	a	l	e	s	s	a	n	d	r	o
Ciphertext:	u	p	j	q	v	h	e	k	h	s	k	e	z	p	u	k	y	q

Source: Computer Networking: A Top-Down Approach, Kurose & Ross, 8th edition, Chapter 8: Security in Computer Networks, Section 8.2.1 Symmetric Key Cryptography,

2.

a) In ~~stream encryption~~, the data is split and encrypted into blocks.

In **a block cipher**, the data is split and encrypted into blocks.

The learning material mentions that "In a block cipher, the message to be encrypted is processed in blocks of k bits."
<u>Source</u>: Computer Networking: A Top-Down Approach, Kurose & Ross, 8th edition, Chapter 8: Security in Computer Networks, Section 8.2.1 Symmetric Key Cryptography, p. 644.

b) The best way to get source authentication of a message is to use ~~symmetric encryption~~.

The best way to get source authentication of a message is to use **digital signatures with asymmetric encryption.**

The learning material mentions that "[…] digital signatures also provide message integrity, allowing the receiver to verify that the message was unaltered as well as the source of the message."
<u>Source</u>: Computer Networking: A Top-Down Approach, Kurose & Ross, 8th edition, Chapter 8: Security in Computer Networks, Section 8.3.3 Digital Signatures, p. 660.

c) Between Alice and Bob, the session keys mechanism uses ~~the same~~ symmetric key each time (connection/session/message).

Between Alice and Bob, the session keys mechanism uses **a new** symmetric key each time (connection/session/message).

*"First Alice chooses a key that will be used to encode the data itself; this key is referred to as a **session key** […]"*
<u>Source</u>: Computer Networking: A Top-Down Approach, Kurose & Ross, 8th edition, Chapter 8: Security in Computer Networks, Section 8.2.2 Public Key Encryption, p. 652.

d) The length of the hash of a message is ~~always approximately equal to~~ the length of the original message.

The length of the hash of a message is **fixed regardless of** the length of the original message.

<u>Source</u>: "[…] a hash function takes an input, m, and computes a fixed-size string H(m) known as a hash."
Computer Networking: A Top-Down Approach, Kurose & Ross, 8th edition, Chapter 8: Security in Computer Networks, Section 8.3.1 – Cryptographic Hash Functions, page 655.

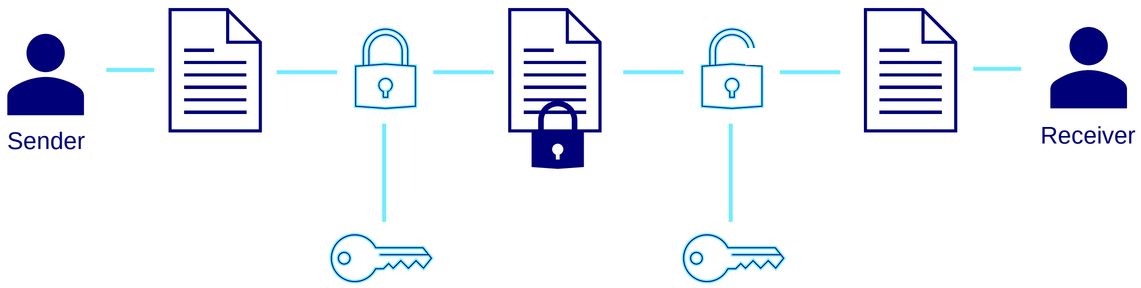
3.

a) RSA is an encryption algorithm that uses a pair of keys, a public key and a private key. The public key can be openly shared while the private key should be kept secret.
These keys are mathematically related, meaning that the message encrypted by the public key can only be decrypted using the corresponding private key and also the other way around.

In simple words, this is achieved by using modular exponential operations with very big numbers (the larger the numbers the more difficult to break). A summary of the mathematical operations and steps described in the learning material could be:

1. Choose two large prime numbers $\mathbf{p}$ and $\mathbf{q}$
2. Calculate $\mathbf{n = p * q}$ and a function called $\boldsymbol{\phi(n)}$
3. Choose a number **e** (public exponent) small and coprime with $\boldsymbol{\phi(n)}$
4. Calculate **d**, which is the inverse of **e** module $\boldsymbol{\phi(n)}$
5. Finally, the **public key** is $(\mathbf{e}, \mathbf{n})$ and the **private key** is $(\mathbf{d}, \mathbf{n})$.

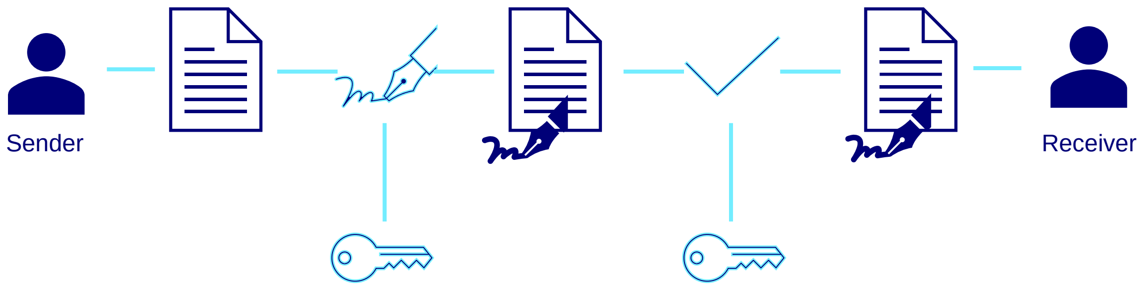
To encrypt a message $\mathbf{m}$ is used $\mathbf{c = m^e \bmod n}$ (with public key)
To decrypt $\mathbf{m = c^d \bmod n}$ (with private key)



Receiver **public key** is used to encrypt the message

Receiver uses its **private key** to decrypt the message

Additionally, a message $\mathbf{m}$ can be digitally signed using $\mathbf{f = m^d \bmod n}$ (with private key)
Then, to verify the sign $\mathbf{m = f^e \bmod n}$ (with public key)

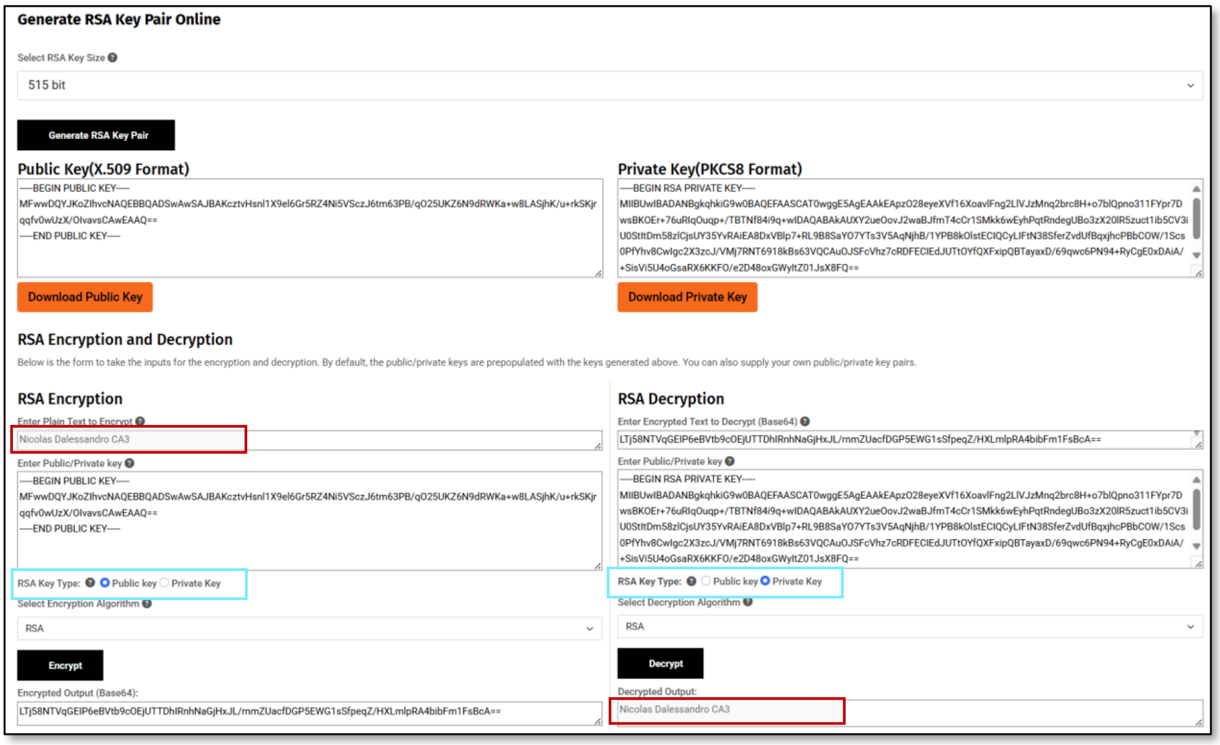


Sender uses its **private key** to sign the message

Sender **public key** is used to verify the authenticity of the message

RSA is powerful because it enables both **secure communication (confidentiality)** and secure **authentication and integrity (digital signatures),** depending on how the key pair is used.

b)



Generate RSA Key Pair Online

Select RSA Key Size: 515 bit

Public Key(X.509 Format)

```
-----BEGIN PUBLIC KEY-----
MFwwDQYJKoZIhvcNAQEBBQADSwAwSAJBAKcztvHsnl1X9el6Gr5RZ4Ni5VSczJ6tm63PB/qO25UKZ6N9dRWKa+w8LASjhK/u+rkSKjr
qqfv0wUzX/OlvavsCAwEAAQ==
-----END PUBLIC KEY-----
```

Private Key(PKCS8 Format)

```
-----BEGIN RSA PRIVATE KEY-----
MIIBUwIBADANBgkqhkiG9w0BAQEFAASCAT0wggE5AgEAAkEApzO28eyeXVf16XoavlFng2LlVJzMnq2brc8H+o7blQpno311FYpr7D
wsBKOEr+76uRIqOuqp+/TBTNf84i9q+wIDAQABAkAUXY2ueOovJ2waBJfmT4cCr1SMkk6wEyhPqtRndegUBo3zX20lR5zuct1ib5CV3i
U0StItDm58zlCjsUY35YvRAiEA8DxVBlp7+RL9B8SaYO7YTs3V5AqNjhB/1YPB8kOlstECIQCyLIFtN38SferZvdUfBqxjhcPBbCOW/1Scs
0PfYhv8CwIgc2X3zcJ/VMj7RNT6918kBs63VQCAuOJSFcVhz7cRDFECIEdJUTtOYfQXFxipQBTayaxD/69qwc6PN94+RyCgE0xDAiA/
+SisVi5U4oGsaRX6KKFO/e2D48oxGWyltZ01JsX8FQ==
```

RSA Encryption and Decryption

Below is the form to take the inputs for the encryption and decryption. By default, the public/private keys are prepopulated with the keys generated above. You can also supply your own public/private key pairs.

RSA Encryption

Enter Plain Text to Encrypt: Nicolas Dalessandro CA3

Enter Public/Private key:

```
-----BEGIN PUBLIC KEY-----
MFwwDQYJKoZIhvcNAQEBBQADSwAwSAJBAKcztvHsnl1X9el6Gr5RZ4Ni5VSczJ6tm63PB/qO25UKZ6N9dRWKa+w8LASjhK/u+rkSKjr
qqfv0wUzX/OlvavsCAwEAAQ==
-----END PUBLIC KEY-----
```

RSA Key Type: [✓ Public key] Private Key

Select Encryption Algorithm: RSA

Encrypted Output (Base64):

```
LTj58NTVqGEIP6eBVtb9cOEjUTTDhIRnhNaGjHxJL/rnmZUacfDGP5EWG1sSfpeqZ/HXLmlpRA4bibFm1FsBcA==
```

RSA Decryption

Enter Encrypted Text to Decrypt (Base64):

```
LTj58NTVqGEIP6eBVtb9cOEjUTTDhIRnhNaGjHxJL/rnmZUacfDGP5EWG1sSfpeqZ/HXLmlpRA4bibFm1FsBcA==
```

Enter Public/Private key:

```
-----BEGIN RSA PRIVATE KEY-----
MIIBUwIBADANBgkqhkiG9w0BAQEFAASCAT0wggE5AgEAAkEApzO28eyeXVf16XoavlFng2LlVJzMnq2brc8H+o7blQpno311FYpr7D
wsBKOEr+76uRIqOuqp+/TBTNf84i9q+wIDAQABAkAUXY2ueOovJ2waBJfmT4cCr1SMkk6wEyhPqtRndegUBo3zX20lR5zuct1ib5CV3i
U0StItDm58zlCjsUY35YvRAiEA8DxVBlp7+RL9B8SaYO7YTs3V5AqNjhB/1YPB8kOlstECIQCyLIFtN38SferZvdUfBqxjhcPBbCOW/1Scs
0PfYhv8CwIgc2X3zcJ/VMj7RNT6918kBs63VQCAuOJSFcVhz7cRDFECIEdJUTtOYfQXFxipQBTayaxD/69qwc6PN94+RyCgE0xDAiA/
+SisVi5U4oGsaRX6KKFO/e2D48oxGWyltZ01JsX8FQ==
```

RSA Key Type: Public key [✓ Private Key]

Select Decryption Algorithm: RSA

Decrypted Output: Nicolas Dalessandro CA3

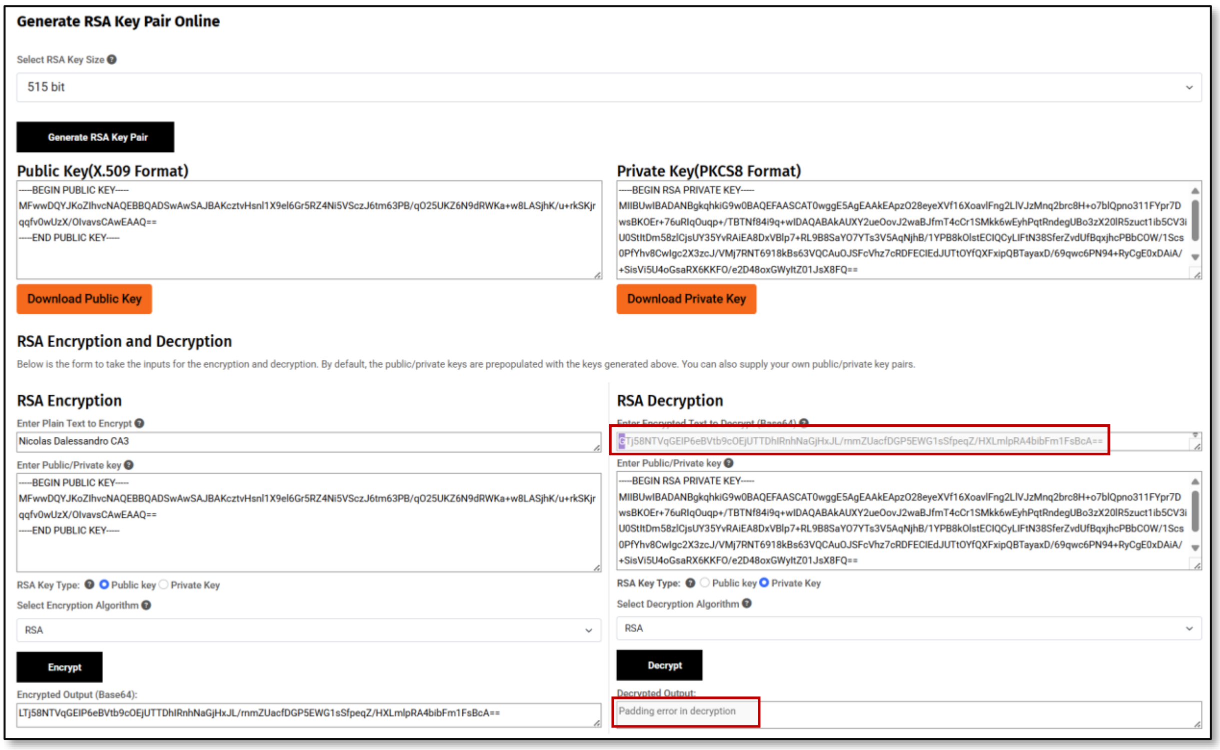
c) As we explain in the section a), there are **two valid** (secure) options for using the RSA keys, each offering different security options.

1. In the image above I encrypted the message using the *public key* and then decrypted it using the *private key*. This ensures **Confidentiality**, since anyone can encrypt the message with the openly shared public key, but only the receiver (who have the private key) can decrypt the message.

2. Additionally, as we also explained in section a), the sender could sign a message using *it private key*, so anyone can verify that signature using the corresponding and openly available *public key* . This ensures **Authenticity and Integrity** because only the owner of the private key could have signed the message, and if a single bit is altered after it has been signed, the public key will no longer be able to correctly verify the signature.

d) Because the RSA output size always matches the size of the key and not the size of the input. If we use the formula described above, $\mathbf{c = m^e \bmod n}$, the result **c** is a number less than $\mathbf{n}$. If the RSA key has a length of 512 bits as in our example, the output will always be 512 bits, no matter the size of the input message.
In our case, even the input message was short, the encrypted output is always 512 bits (88 characters in base64, with 2 '=' at the end).

e)



Generate RSA Key Pair Online

Select RSA Key Size: 515 bit

Generate RSA Key Pair

Public Key(X.509 Format)

```
-----BEGIN PUBLIC KEY-----
MFwwDQYJKoZIhvcNAQEBBQADSwAwSAJBAKcztvHsnl1X9el6Gr5RZ4Ni5VSczJ6tm63PB/qO25UKZ6N9dRWKa+w8LASjhK/u+rkSKjr
qqfv0wUzX/OlvavsCAwEAAQ==
-----END PUBLIC KEY-----
```

Download Public Key

Private Key(PKCS8 Format)

```
-----BEGIN RSA PRIVATE KEY-----
MIIBUwIBADANBgkqhkiG9w0BAQEFAASCAT0wggE5AgEAAkEApzO28eyeXVf16XoavlFng2LlVJzMnq2brc8H+o7blQpno311FYpr7D
wsBKOEr+76uRIqOuqp+/TBTNf84i9q+wIDAQABAkAUXY2ueOovJ2waBJfmT4cCr1SMkk6wEyhPqtRndegUBo3zX20lR5zuct1ib5CV3i
U0StltDm58zlCjsUY35YvRAiEA8DxVBlp7+RL9B8SaYO7YTs3V5AqNjhB/1YPB8kOlstECIQCyLIFtN38SferZvdUfBqxjhcPBbCOW/1Scs
0PfYhv8Cwlgc2X3zcJ/VMj7RNT6918kBs63VQCAuOJSFcVhz7cRDFECIEdJUTtOYfQXFxipQBTayaxD/69qwc6PN94+RyCgE0xDAiA/
+SisVi5U4oGsaRX6KKFO/e2D48oxGWyltZ01JsX8FQ==
```

Download Private Key

RSA Encryption and Decryption

Below is the form to take the inputs for the encryption and decryption. By default, the public/private keys are prepopulated with the keys generated above. You can also supply your own public/private key pairs.

RSA Encryption

Enter Plain Text to Encrypt: Nicolas Dalessandro CA3

Enter Public/Private key:

```
-----BEGIN PUBLIC KEY-----
MFwwDQYJKoZIhvcNAQEBBQADSwAwSAJBAKcztvHsnl1X9el6Gr5RZ4Ni5VSczJ6tm63PB/qO25UKZ6N9dRWKa+w8LASjhK/u+rkSKjr
qqfv0wUzX/OlvavsCAwEAAQ==
-----END PUBLIC KEY-----
```

RSA Key Type: [✓ Public key] Private Key

Select Encryption Algorithm: RSA

Encrypt

Encrypted Output (Base64):

```
LTj58NTVqGEIP6eBVtb9cOEjUTTDhIRnhNaGjHxJL/rnmZUacfDGP5EWG1sSfpeqZ/HXLmlpRA4bibFm1FsBcA==
```

RSA Decryption

Enter Encrypted Text to Decrypt (Base64):

```
GTj58NTVqGEIP6eBVtb9cOEjUTTDhIRnhNaGjHxJL/rnmZUacfDGP5EWG1sSfpeqZ/HXLmlpRA4bibFm1FsBcA==
```

Enter Public/Private key:

```
-----BEGIN RSA PRIVATE KEY-----
MIIBUwIBADANBgkqhkiG9w0BAQEFAASCAT0wggE5AgEAAkEApzO28eyeXVf16XoavlFng2LlVJzMnq2brc8H+o7blQpno311FYpr7D
wsBKOEr+76uRIqOuqp+/TBTNf84i9q+wIDAQABAkAUXY2ueOovJ2waBJfmT4cCr1SMkk6wEyhPqtRndegUBo3zX20lR5zuct1ib5CV3i
U0StltDm58zlCjsUY35YvRAiEA8DxVBlp7+RL9B8SaYO7YTs3V5AqNjhB/1YPB8kOlstECIQCyLIFtN38SferZvdUfBqxjhcPBbCOW/1Scs
0PfYhv8Cwlgc2X3zcJ/VMj7RNT6918kBs63VQCAuOJSFcVhz7cRDFECIEdJUTtOYfQXFxipQBTayaxD/69qwc6PN94+RyCgE0xDAiA/
+SisVi5U4oGsaRX6KKFO/e2D48oxGWyltZ01JsX8FQ==
```

RSA Key Type: Public key [✓ Private Key]

Select Decryption Algorithm: RSA

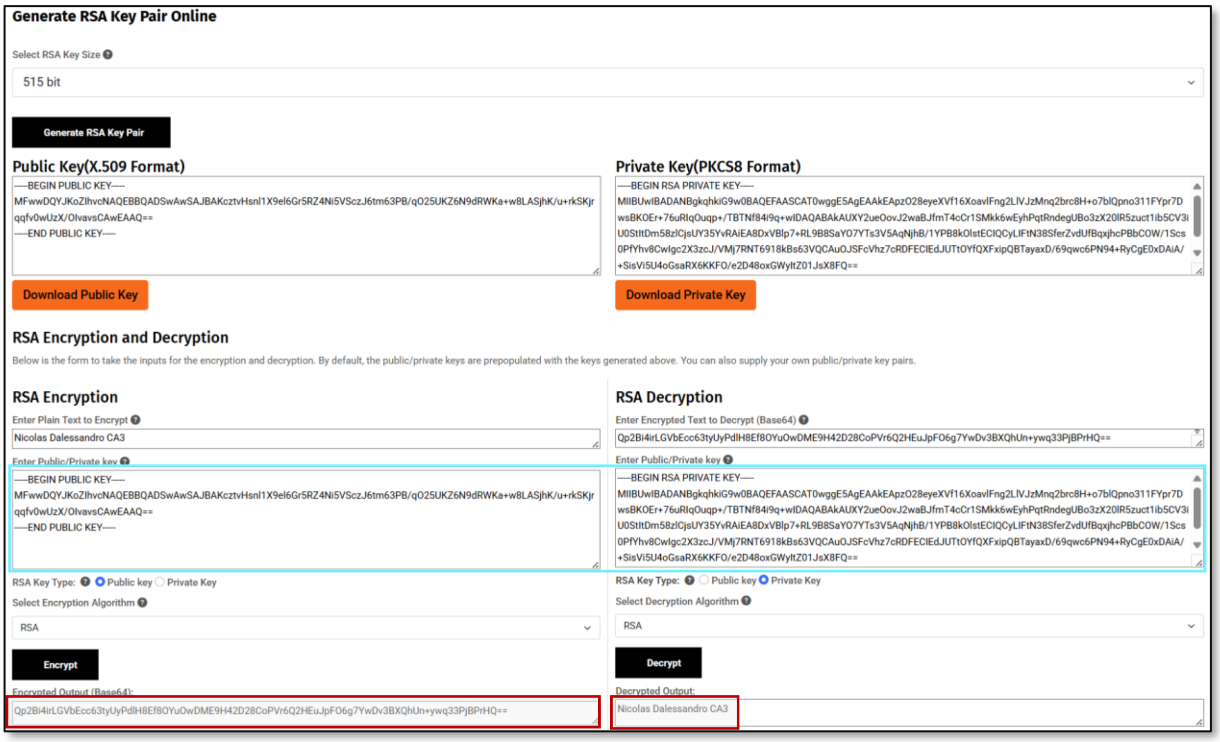
Decrypt

Decrypted Output:

```
Padding error in decryption
```

I changed the first character of the requested input field, and as we can see in the image above, the error obtained is "**Padding error in decryption**". This demonstrates that the message cannot be altered, if someone modifies the ciphertext, even just one character as I did, the decryption process fails, ensuring that the message integrity is preserved.

f)



Generate RSA Key Pair Online

Select RSA Key Size: 515 bit

Generate RSA Key Pair

Public Key(X.509 Format)

```
-----BEGIN PUBLIC KEY-----
MFwwDQYJKoZIhvcNAQEBBQADSwAwSAJBAKcztvHsnl1X9el6Gr5RZ4Ni5VSczJ6tm63PB/qO25UKZ6N9dRWKa+w8LASjhK/u+rkSKjr
qqfv0wUzX/OIvavsCAwEAAQ==
-----END PUBLIC KEY-----
```

Download Public Key

Private Key(PKCS8 Format)

```
-----BEGIN RSA PRIVATE KEY-----
MIIBUwIBADANBgkqhkiG9w0BAQEFAASCAT0wggE5AgEAAkEApzO28eyeXVf16XoavlFng2LlVJzMnq2brc8H+o7blQpno311FYpr7D
wsBKOEr+76uRIqOuqp+/TBTNf84i9q+wIDAQABAkAUXY2ueOovJ2waBJfmT4cCr1SMkk6wEyhPqtRndegUBo3zX20lR5zuct1ib5CV3i
U0StItDm58zlCjsUY35YvRAiEA8DxVBlp7+RL9B8SaYO7YTs3V5AqNjhB/1YPB8kOlstECIQCyLIFtN38SferZvdUfBqxjhcPBbCOW/1Scs
0PfYhv8CwIgc2X3zcJ/VMj7RNT6918kBs63VQCAuOJSFcVhz7cRDFECIEdJUTtOYfQXFxipQBTayaxD/69qwc6PN94+RyCgE0xDAiA/
+SisVi5U4oGsaRX6KKFO/e2D48oxGWyltZ01JsX8FQ==
```

Download Private Key

RSA Encryption and Decryption

Below is the form to take the inputs for the encryption and decryption. By default, the public/private keys are prepopulated with the keys generated above. You can also supply your own public/private key pairs.

RSA Encryption

Enter Plain Text to Encrypt: Nicolas Dalessandro CA3

Enter Public/Private key:

```
-----BEGIN PUBLIC KEY-----
MFwwDQYJKoZIhvcNAQEBBQADSwAwSAJBAKcztvHsnl1X9el6Gr5RZ4Ni5VSczJ6tm63PB/qO25UKZ6N9dRWKa+w8LASjhK/u+rkSKjr
qqfv0wUzX/OIvavsCAwEAAQ==
-----END PUBLIC KEY-----
```

RSA Key Type: [✓ Public key] Private Key

Select Encryption Algorithm: RSA

Encrypt

Encrypted Output (Base64):

```
Qp2Bi4irLGVbEcc63tyUyPdlH8Ef8OYuOwDME9H42D28CoPVr6Q2HEuJpFO6g7YwDv3BXQhUn+ywq33PjBPrHQ==
```

RSA Decryption

Enter Encrypted Text to Decrypt (Base64):

```
Qp2Bi4irLGVbEcc63tyUyPdlH8Ef8OYuOwDME9H42D28CoPVr6Q2HEuJpFO6g7YwDv3BXQhUn+ywq33PjBPrHQ==
```

Enter Public/Private key:

```
-----BEGIN RSA PRIVATE KEY-----
MIIBUwIBADANBgkqhkiG9w0BAQEFAASCAT0wggE5AgEAAkEApzO28eyeXVf16XoavlFng2LlVJzMnq2brc8H+o7blQpno311FYpr7D
wsBKOEr+76uRIqOuqp+/TBTNf84i9q+wIDAQABAkAUXY2ueOovJ2waBJfmT4cCr1SMkk6wEyhPqtRndegUBo3zX20lR5zuct1ib5CV3i
U0StItDm58zlCjsUY35YvRAiEA8DxVBlp7+RL9B8SaYO7YTs3V5AqNjhB/1YPB8kOlstECIQCyLIFtN38SferZvdUfBqxjhcPBbCOW/1Scs
0PfYhv8CwIgc2X3zcJ/VMj7RNT6918kBs63VQCAuOJSFcVhz7cRDFECIEdJUTtOYfQXFxipQBTayaxD/69qwc6PN94+RyCgE0xDAiA/
+SisVi5U4oGsaRX6KKFO/e2D48oxGWyltZ01JsX8FQ==
```

RSA Key Type: Public key [✓ Private Key]

Select Decryption Algorithm: RSA

Decrypt

Decrypted Output: Nicolas Dalessandro CA3

As we can see, when we click **Encrypt** again using the same *public key*, the encrypted output changes but the message can be still decrypted successfully using the same *private key*.
This happens because the RSA encryption process uses a padding schema that includes some random bytes, for example:

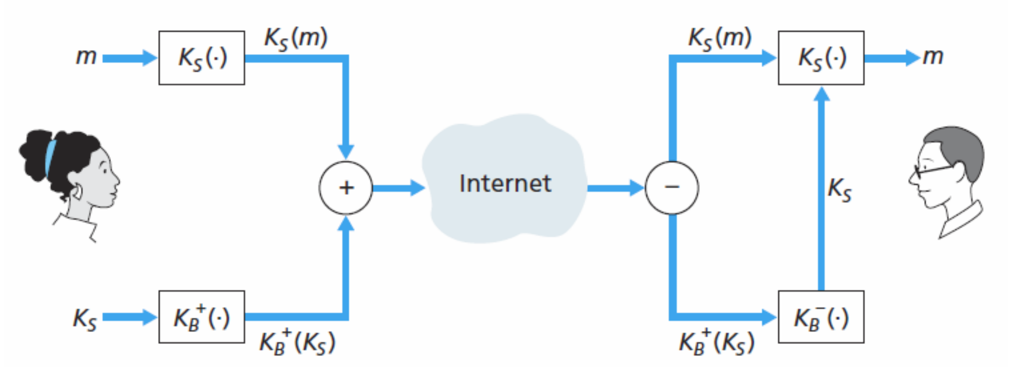
0x00 0x02 [some non-zero bytes] 0x00 [message bytes]

This new "message with padding" is what it gets encrypted. When we use the private key to decrypt the message, the system knows how to identify and remove this padding (because is a standard structure), so that we can recover the original message.

Sources:

- *Computer Networking: A Top-Down Approach, Kurose & Ross, 8th edition, Chapter 8: Security in Computer Networks, Section 8.2.2 – Public Key Encryption, page 650-652.*
- *https://medium.com/asecuritysite-when-bob-met-alice/so-how-does-padding-work-in-rsa-6b34a123ca1f*
- *https://www.splunk.com/en_us/blog/learn/rsa-algorithm-cryptography.html*

4.



This chapter covers different approaches in developing a secure email communication between Alice and Bob, applying the different cryptographic principles described in previous units (also covered in previous questions of this practice).

This figure 8.19 shows how Alice can send Bob a secure e-mail using a combination of symmetric encryption (secret key **K_S**) and asymmetric encryption (public key **K_B^+**):

Alice point of view (sender):

1. Choose a random secret key **K_s** that will be only used for this message.
2. Encrypts the message **m** using that key **=> K_s(m)**.
3. Encrypts the secret key **K_s** with Bob´s public key **K_B^+ => K_B^+(K_s)**
4. Concatenates both results **=> K_s(m) + K_B^+(K_s)**.
5. Send the packet trough internet to Bob.

Bob point of view (recipient):

1. Receives **K_S(m)** and **K_B+(K_S).**
2. Decrypts **K_B^+(K_S)** with his private key **K_B^- => gets the session key K_S.**
3. Use the obtained secret key **K_s** to decrypt **K_s(m) => <u>gets the message m.</u>**

As explained in the book, this figure **provides only confidentiality**. Using a combination of symmetric encryption **K_S(m)** and asymmetric encryption to protect the session key **(K_B+(K_S)),** ensures **confidentiality** because only Bob with his private key can recover **K_S** and read the message **m**, but it **does not guarantee integrity** because there is no verification that the message has not been modified along the way and also **there is no guarantee that it was Alice who send the message (authenticity).**

To achieve this, the book introduces two new concepts covered in figures 8.20 and 8.21. This last one combines the concepts explained in figures 8.19 + 8.20 to achieve the three security services required.

Figures 8.20 uses a Hash of the message **H(m)** and a digital signature with Alice private key **=> K_A^-(H(m)).** Bob can verify the **authenticity** and **integrity** of the message using Alice public key **(K_A^+)**, but as explained in the book, this process **does not provide confidentiality**.

Finally, figure 8.21 combines these two approaches to provide the complete security services Confidentiality, Authenticity and Integrity:

1. Alice creates a **m + K_A-(H(m))** to **guarantee authenticity and integrity** as in figure 8.20**.**
2. Encrypts everything with the symmetric secret key **K_s** to **provide confidentiality.**
3. Additionally, **K_s** is protected with **K_B^+** (Bob public key) as in figure 8.19.

5. Main fields in a X.509 certificate from the section 8.4 of the learning material:

Field Name	**Functionality and Justification**
Version	Specifies the version or standard of the X509 used. Ensures compatibility with the recipient software.
Serial number	Unique identifier assigned by the CA that helps to reference or revoke the certificate.
Signature	Indicates the algorithm used by the CA to digitally sign the certificate
Issuer name	DN of the CA that signed the certificate.
Validity period	Start and end date that indicates how long the certificate is considered valid and trusted.
Subject name	DN of the certificate owner (person, organization, server, etc.).
Subject public key	Public key that its being certificated and the associated algorithm.

**CA: Certification Authority*
**DN: Distinguished Name*

A **digital signature** is a cryptographic technique where the CA signs a hash of the certificate data using its private key.
The signature is added to the certificate, and anyone with the CA public key can verifies it.
This signature is used to ensure authenticity and integrity and is what turns a public key into a **trusted certificate**.

To generate the **digital signature** (in the case of a certificate), the subject sends its public key and identification data to a Certification Authority. The CA generates a hash (summary) of the certificate content and then encrypts the hash using its private key (this is the digital signature).
The resulting certificate contains the subject original information, the digital signature and metadata signed by the CA.

The CA digital signature is what transforms a document with public key into **a trusted certificate**. Anyone with the CA public key can verify the signature. If the signature is valid, we can trust that the public key belongs to the claimed proprietary (a web server, a person, etc.).

Sources:

- *Computer Networking: A Top-Down Approach, Kurose & Ross, 8th edition, Chapter 8: Security in Computer Networks, Section 8.4 – End-Point Authentication, page 664.*
- *https://www.geeksforgeeks.org/x-509-authentication-service/*

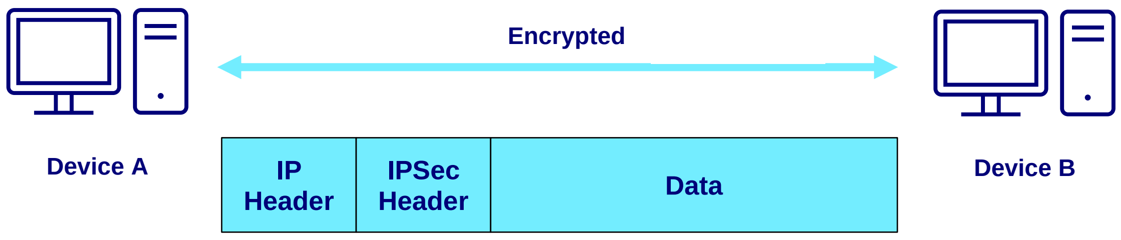
6. IPsec mechanism:

a) TLS/SSL and IPSec are both used to protect data, but they work at different layers of the network.
   1. **TLS** operates at the transport layer. It protects data in applications such as web browsers, emails, chats, etc. For example, when visiting a website with HTTPS, TLS is being used.
   2. **IPSec** operates at the IP network layer, and it is aimed to protect traffic between two devices. As the book explain, it is often used to create VPNs.

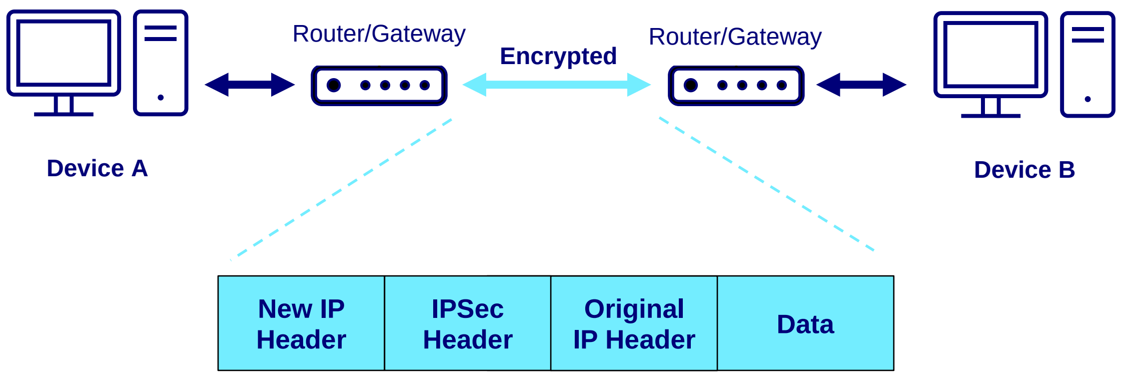
   In summary, **TLS works at the transport layer and protects specific apps** (like browsers). IPsec **works at the network layer and protects all data between two IP addresses**.

b) IPsec Modes:

   1. **Trasport Mode**: In transport mode the security is provided to the traffic end to end. Only the data part of the IP packet is encrypted. The original IP address is visible. Commonly used when the sender and receiver are the final devices.



   2. **Tunnel Mode**: In tunnel mode the protection is provided from the router or gateway of one network, to the router or gateway of another network. The whole IP packet (also the IP address) is encrypted.



Source: https://www.omscs-notes.com/information-security/ipsec-and-tls/

c) Selected IPSec mode:

1. **Sender <=IPSec=> Receiver**
   Similar to our example, the best choice is **Transport Mode** because both ends are the final devices.

2. **Sender <-IP-> Router 1 <=IPSec=> Router 2 <-IP-> Receiver**
   Also similar to the above diagram, the best choice here is Tunnel Mode because the encryption is between two routers.

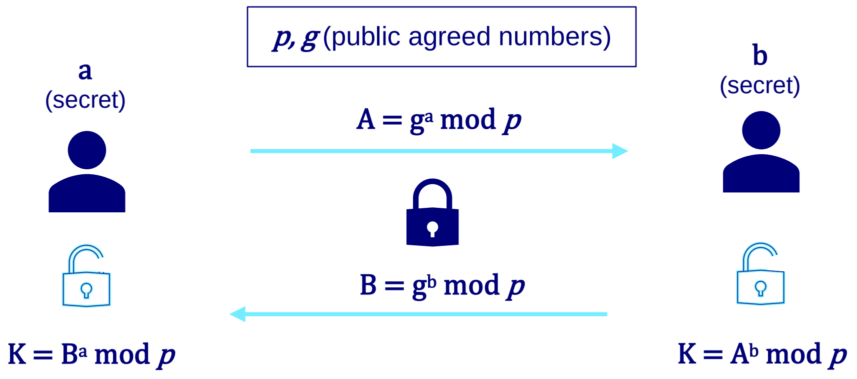
3. **Sender <=IPSec=> Router <-IP-> Receiver**
   The best choice here is Tunnel Mode because the encryption is only between the sender and a router.

7. "Diffie–Hellman" mechanism:

a) This algorithm is a protocol for exchanging keys, allowing two parts to communicate in a public channel (such as the internet) to establish a mutual secret.
In networking, this **allows two devices to agree on a secret key** without the need to send it directly. A mathematical technique is used to derive the key, that will be then used to encrypt messages.

b) How does this work:

1. Two public keys (numbers) are agreed, p (a prime number) and g (generator number).
2. Each part chooses a secret number (private key) a and b (integers and less than p).
3. Each part sends a computed value using the formula:
   $A = g^a \bmod p$
   $B = g^b \bmod p$
4. Both parts calculate the shared secret key using the formula:
   $K = B^a \bmod p$
   $K = A^b \bmod p$
5. Both get the same result K which is the shared key.



c) **Ephemeral Diffie-Hellman** is when the key exchange is done only once per session and then discarded to make the connection safer.
In **HTTPS** this method is used to create **session keys** that change every time, so if one key is stolen, other sessions are safe.

d) Example of the calculation:

- Public numbers: p (17) and g (3).
- Each part secret number (private key) a (12) and b (6).

Step 1: Values to be sent:

$$A = g^a \bmod p = 3^{12} \bmod 17 = 4$$
$$B = g^b \bmod p = 3^6 \bmod 17 = 16$$

Step 2: Calculate the shared key

$$K = B^a \bmod p = 16^{12} \bmod 17 = 1$$
$$K = A^b \bmod p = 4^6 \bmod 17 = 1$$

Both get the same result $K=1$ which is the shared key.

Sources:

- *https://community.ibm.com/community/user/ibmz-and-linuxone/blogs/subhasish-sarkar1/2020/07/09/understanding-diffiehellman-key-exchange-method?communityKey=8bcc5745-f7d5-43e3-a7ef-a4801c68c752*
- https://security.stackexchange.com/questions/45963/diffie-hellman-key-exchange-in-plain-english

8. **JWT and OAuth 2.0 practice**

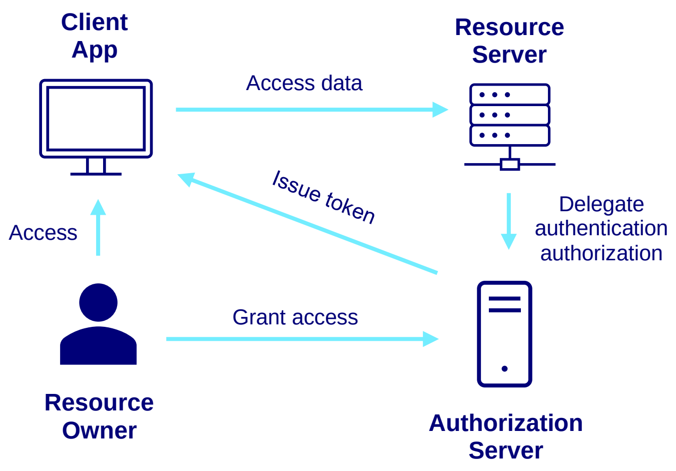
a)

OAuth 2.0 is an authorization protocol that allows to a third-party application to be able to access protected resources on behalf of a user but without sharing the password. This protocol defines roles and flows to grant secure access tokens that can be used to interact with APIs.

Main Roles in OAuth 2.0

Role	Description
Resource Owner	The user who owns the data or resource
Client	The app that wants to access the resource
Authorization Server	Verify the identity and generate tokens
Resource Server	Have the protected resource

OAuth 2.0 Typical flow (simplified for learning purposes in this memory)



A **JWT (Json Web Token)** is a digitally signed token that contains information such as user info, permissions, expiration, etc. It is structured like this:

<HEADER>.<PAYLOAD>.<SIGNATURE>

How it is used

The JWT is included in the header of the HTTP request, this allows the server to identify the user, check the permissions and confirm that the token is not expired or modified:

Authorization: Bearer <JWT>

How OAuth 2.0 and JWT work together

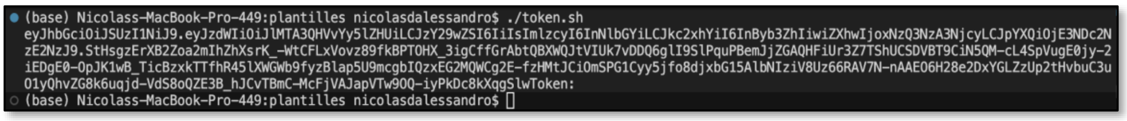
Step	Operation	Description
1	**POST / token**	Client requests the token using **Basic Auth (username:password)**
2	**Token emission**	Server returns a JWT as a Bearer token.
3	**Access with token**	The client uses the token in the **Authorization: Bearer** header

OAuth 2.0 provides a secure way to authorize the clients using access tokens and JWT is the format (compact) for those tokens. This system combination is widely used nowadays because it **ensures security, and easy verification of each request without the need of repeated login.**

b) To obtain the token following the statement instruction, I prepared the following curl command. This call sends a POST request to the token endpoint adding the header, the user and password requested:

```
curl -X POST http://labxarxes.techlab.uoc.edu:8095/dslab-api/token \
  -H "X-TenantID: prova" \
  -u e107@uoc.edu:TfM2023,
```

After executing the script, we can see the obtained token:



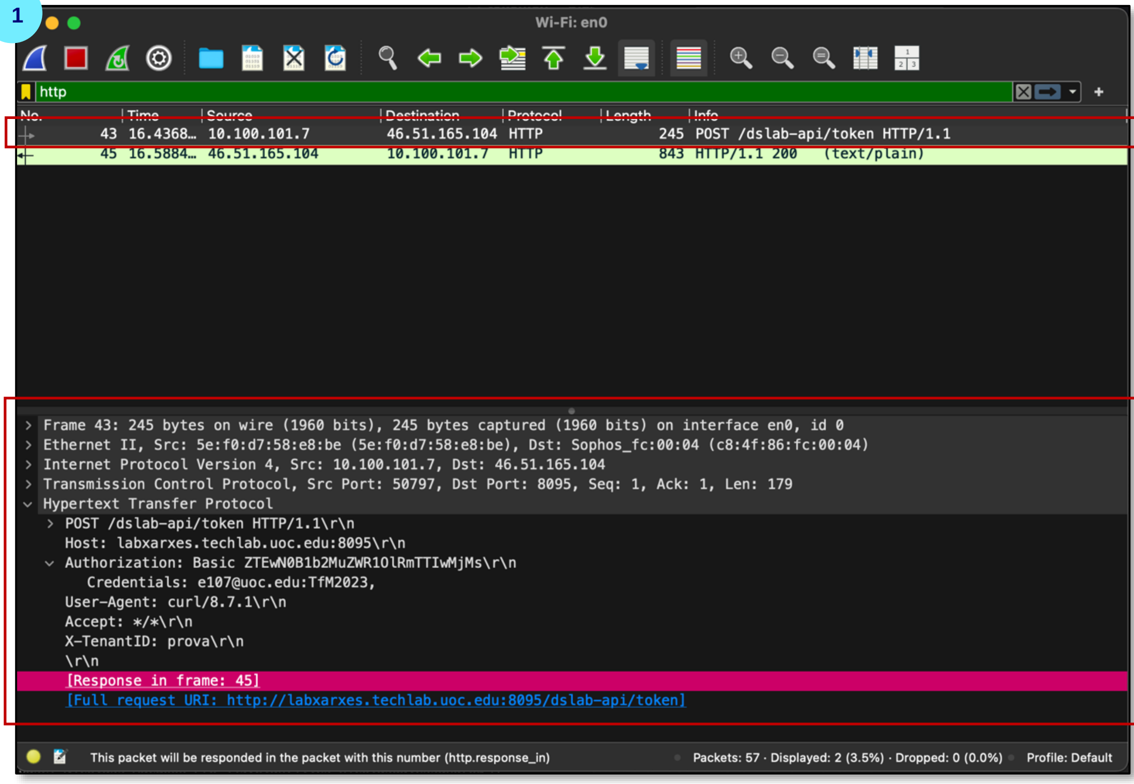
```
● (base) Nicolass-MacBook-Pro-449:plantilles nicolasdalessandro$ ./token.sh
  eyJhbGciOiJSUzI1NiJ9.eyJzdWIiOiJlMTA3QHVvYy5lZHUiLCJzY29wZSI6IiIsImlzcyI6InNlbGYiLCJkc2xhYiI6InByb3ZhIiwiZXhwIjoxNzQ3NzA3NjcyLCJpYXQiOjE3NDc2N
  zE2NzJ9.StHsgzErXB2Zoa2mIhZhXsrK_-WtCFLxVovz89fkBPTOHX_3igCffGrAbtQBXWQJtVIUk7vDDQ6glI9SlPquPBemJjZGAQHFiUr3Z7TShUCSDVBT9CiN5QM-cL4SpVugE0jy-2
  iEDgE0-OpJK1wB_TicBzxkTTfhR45lXWGWb9fyzBlap5U9mcgbIQzxEG2MQWCg2E-fzHMtJCiOmSPG1Cyy5jfo8djxbG15AlbNIziV8Uz66RAV7N-nAAEO6H28e2DxYGLZzUp2tHvbuC3u
  O1yQhvZG8k6uqjd-VdS8oQZE3B_hJCvTBmC-McFjVAJapVTw90Q-iyPkDc8kXqgSlwToken:
○ (base) Nicolass-MacBook-Pro-449:plantilles nicolasdalessandro$ 
```

Finally, we can see the request and response packets captured in Wireshark for this operation.

1. The first packet (43) shows the **POST** request to the **/dslab-api/token** endpoint with all the expected headers, including **Authorization**, **X-TenantID** and **Host**.

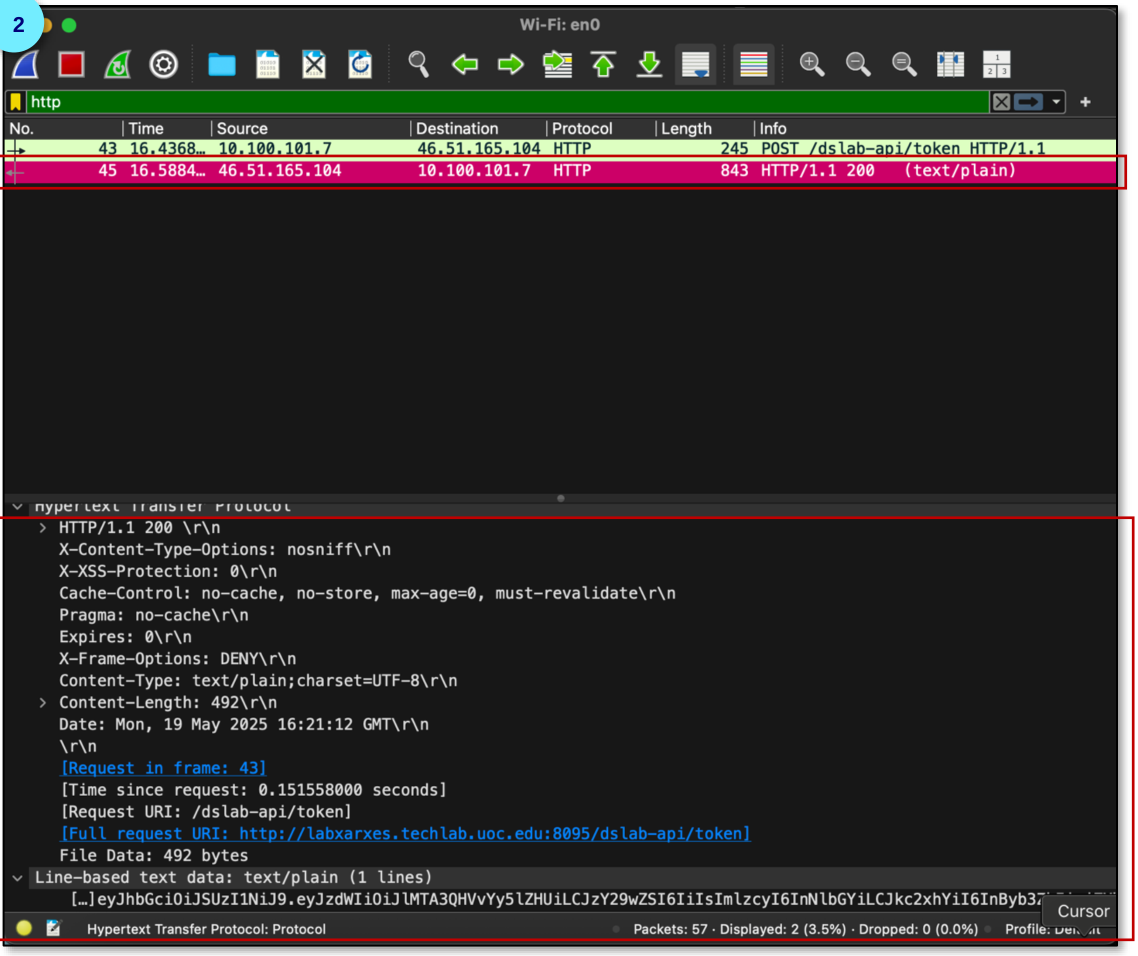
2. The second packet (45) contains HTTP response including the token in plain text.

1



```
Wi-Fi: en0
http
No.   Time       Source         Destination    Protocol  Length  Info
  43  16.4368…  10.100.101.7   46.51.165.104  HTTP       245  POST /dslab-api/token HTTP/1.1
  45  16.5884…  46.51.165.104  10.100.101.7   HTTP       843  HTTP/1.1 200   (text/plain)

> Frame 43: 245 bytes on wire (1960 bits), 245 bytes captured (1960 bits) on interface en0, id 0
> Ethernet II, Src: 5e:f0:d7:58:e8:be (5e:f0:d7:58:e8:be), Dst: Sophos_fc:00:04 (c8:4f:86:fc:00:04)
> Internet Protocol Version 4, Src: 10.100.101.7, Dst: 46.51.165.104
> Transmission Control Protocol, Src Port: 50797, Dst Port: 8095, Seq: 1, Ack: 1, Len: 179
v Hypertext Transfer Protocol
  > POST /dslab-api/token HTTP/1.1\r\n
    Host: labxarxes.techlab.uoc.edu:8095\r\n
  v Authorization: Basic ZTEwN0B1b2MuZWR1OlRmTTIwMjMs\r\n
      Credentials: e107@uoc.edu:TfM2023,
    User-Agent: curl/8.7.1\r\n
    Accept: */*\r\n
    X-TenantID: prova\r\n
    \r\n
    [Response in frame: 45]
    [Full request URI: http://labxarxes.techlab.uoc.edu:8095/dslab-api/token]

This packet will be responded in the packet with this number (http.response_in)   Packets: 57 · Displayed: 2 (3.5%) · Dropped: 0 (0.0%)   Profile: Default
```

2

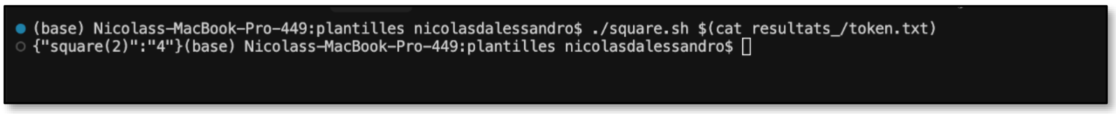


```
Wi-Fi: en0
http
No.   Time       Source         Destination    Protocol  Length  Info
  43  16.4368…  10.100.101.7   46.51.165.104  HTTP       245  POST /dslab-api/token HTTP/1.1
  45  16.5884…  46.51.165.104  10.100.101.7   HTTP       843  HTTP/1.1 200   (text/plain)

v Hypertext Transfer Protocol
  > HTTP/1.1 200 \r\n
    X-Content-Type-Options: nosniff\r\n
    X-XSS-Protection: 0\r\n
    Cache-Control: no-cache, no-store, max-age=0, must-revalidate\r\n
    Pragma: no-cache\r\n
    Expires: 0\r\n
    X-Frame-Options: DENY\r\n
    Content-Type: text/plain;charset=UTF-8\r\n
  > Content-Length: 492\r\n
    Date: Mon, 19 May 2025 16:21:12 GMT\r\n
    \r\n
    [Request in frame: 43]
    [Time since request: 0.151558000 seconds]
    [Request URI: /dslab-api/token]
    [Full request URI: http://labxarxes.techlab.uoc.edu:8095/dslab-api/token]
    File Data: 492 bytes
v Line-based text data: text/plain (1 lines)
    […]eyJhbGciOiJSUzI1NiJ9.eyJzdWIiOiJlMTA3QHVvYy5lZHUiLCJzY29wZSI6IiIsImlzcyI6InNlbGYiLCJkc2xhYiI6InByb3Z
Cursor
Hypertext Transfer Protocol: Protocol   Packets: 57 · Displayed: 2 (3.5%) · Dropped: 0 (0.0%)   Profile: Default
```

c) To call the square operation I created the following script with the described HTTP GET operation, using the JWT token obtained in the previous step and stored in **resultats_/token.txt**:

```bash
#!/bin/bash
TOKEN=$1

curl -X GET http://labxarxes.techlab.uoc.edu:8095/dslab-api/xai/square/2 \
  -H "Authorization: Bearer $TOKEN"
```

Testing this in my computer, we obtained the expected JSON:

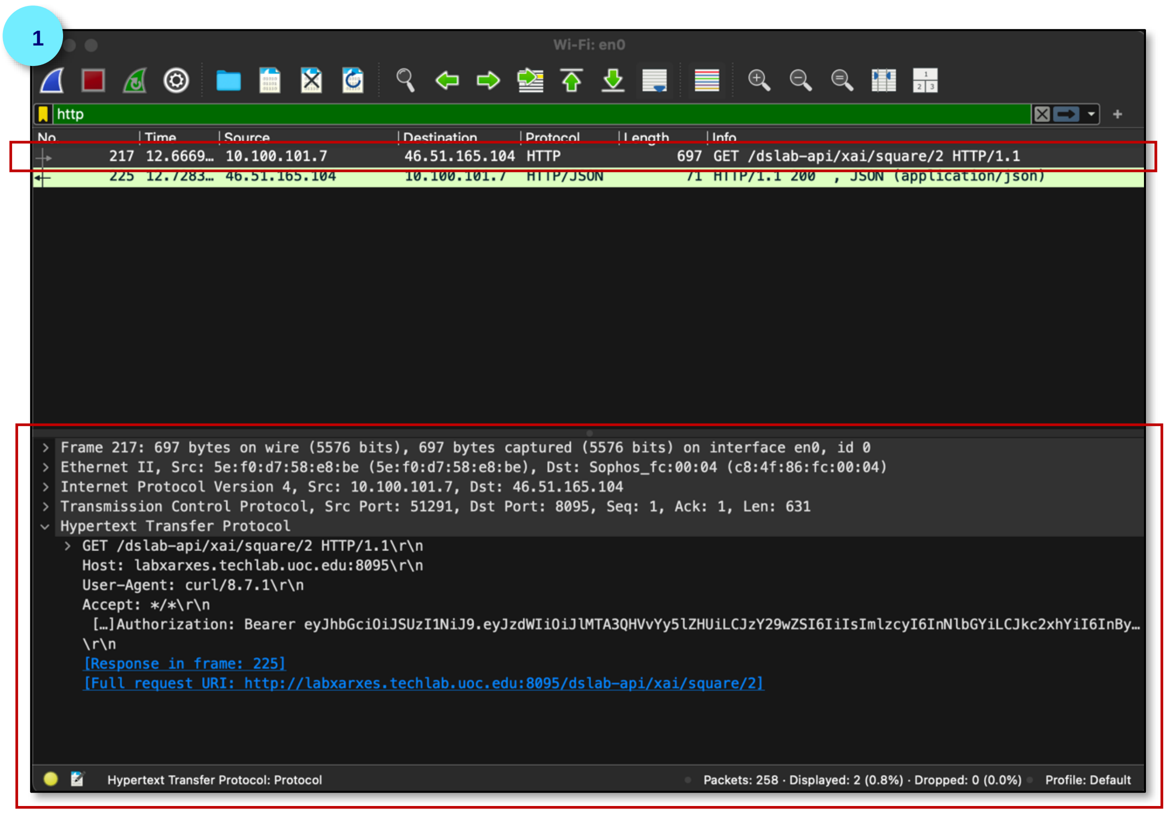


```
● (base) Nicolass-MacBook-Pro-449:plantilles nicolasdalessandro$ ./square.sh $(cat resultats_/token.txt)
○ {"square(2)":"4"}(base) Nicolass-MacBook-Pro-449:plantilles nicolasdalessandro$ 
```

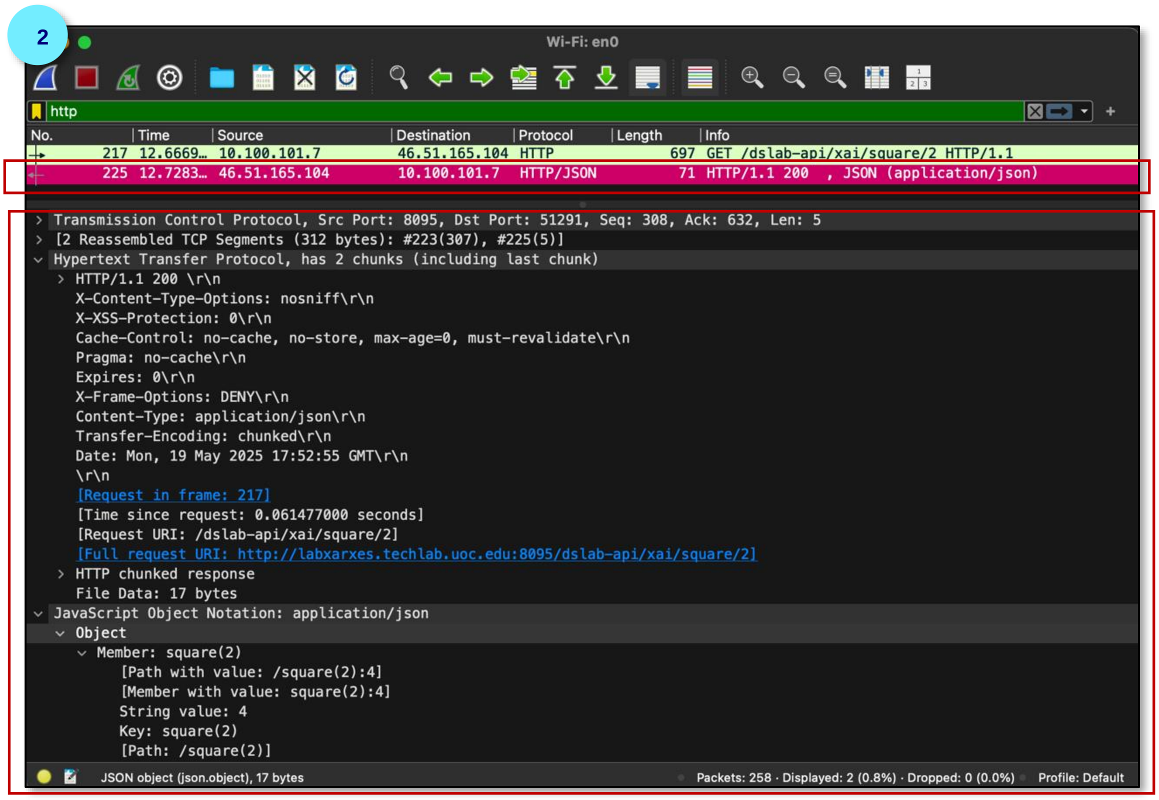
Finally, I captured the request and response using Wireshark.

1. The first packet (217) shows the **GET** request to the **/dslab-api/square/2** endpoint with all the expected headers, including **Authorization** Bearer with the previously obtained token.

2. The second packet (225) contains HTTP response including the JSON object with the expected response to the number we wanted to calculate the square (in this example 2, as requested for DSlab test).

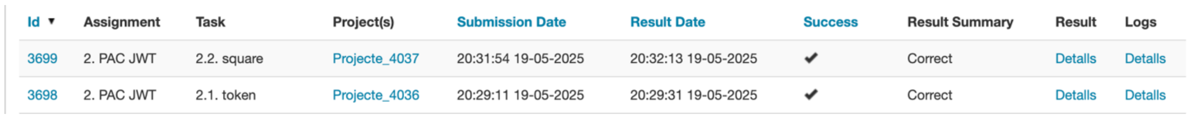


```
Wi-Fi: en0
http
No.   Time       Source          Destination     Protocol    Length  Info
217   12.6669…   10.100.101.7    46.51.165.104   HTTP        697     GET /dslab-api/xai/square/2 HTTP/1.1
225   12.7283…   46.51.165.104   10.100.101.7    HTTP/JSON   71      HTTP/1.1 200  , JSON (application/json)

> Frame 217: 697 bytes on wire (5576 bits), 697 bytes captured (5576 bits) on interface en0, id 0
> Ethernet II, Src: 5e:f0:d7:58:e8:be (5e:f0:d7:58:e8:be), Dst: Sophos_fc:00:04 (c8:4f:86:fc:00:04)
> Internet Protocol Version 4, Src: 10.100.101.7, Dst: 46.51.165.104
> Transmission Control Protocol, Src Port: 51291, Dst Port: 8095, Seq: 1, Ack: 1, Len: 631
v Hypertext Transfer Protocol
  > GET /dslab-api/xai/square/2 HTTP/1.1\r\n
    Host: labxarxes.techlab.uoc.edu:8095\r\n
    User-Agent: curl/8.7.1\r\n
    Accept: */*\r\n
     […]Authorization: Bearer eyJhbGciOiJSUzI1NiJ9.eyJzdWIiOiJlMTA3QHVvYy5lZHUiLCJzY29wZSI6IiIsImlzcyI6InNlbGYiLCJkc2xhYiI6InBy…
    \r\n
    [Response in frame: 225]
    [Full request URI: http://labxarxes.techlab.uoc.edu:8095/dslab-api/xai/square/2]

Hypertext Transfer Protocol: Protocol     Packets: 258 · Displayed: 2 (0.8%) · Dropped: 0 (0.0%)     Profile: Default
```



```
No.      Time       Source          Destination     Protocol   Length  Info
   217  12.6669…  10.100.101.7    46.51.165.104   HTTP         697  GET /dslab-api/xai/square/2 HTTP/1.1
   225  12.7283…  46.51.165.104   10.100.101.7    HTTP/JSON     71  HTTP/1.1 200  , JSON (application/json)

> Transmission Control Protocol, Src Port: 8095, Dst Port: 51291, Seq: 308, Ack: 632, Len: 5
> [2 Reassembled TCP Segments (312 bytes): #223(307), #225(5)]
v Hypertext Transfer Protocol, has 2 chunks (including last chunk)
  > HTTP/1.1 200 \r\n
    X-Content-Type-Options: nosniff\r\n
    X-XSS-Protection: 0\r\n
    Cache-Control: no-cache, no-store, max-age=0, must-revalidate\r\n
    Pragma: no-cache\r\n
    Expires: 0\r\n
    X-Frame-Options: DENY\r\n
    Content-Type: application/json\r\n
    Transfer-Encoding: chunked\r\n
    Date: Mon, 19 May 2025 17:52:55 GMT\r\n
    \r\n
    [Request in frame: 217]
    [Time since request: 0.061477000 seconds]
    [Request URI: /dslab-api/xai/square/2]
    [Full request URI: http://labxarxes.techlab.uoc.edu:8095/dslab-api/xai/square/2]
  > HTTP chunked response
    File Data: 17 bytes
v JavaScript Object Notation: application/json
  v Object
    v Member: square(2)
        [Path with value: /square(2):4]
        [Member with value: square(2):4]
        String value: 4
        Key: square(2)
        [Path: /square(2)]

JSON object (json.object), 17 bytes      Packets: 258 · Displayed: 2 (0.8%) · Dropped: 0 (0.0%)   Profile: Default
```

d) Submission to DS lab:



Id ▾	Assignment	Task	Project(s)	Submission Date	Result Date	Success	Result Summary	Result	Logs
3699	2. PAC JWT	2.2. square	Projecte_4037	20:31:54 19-05-2025	20:32:13 19-05-2025	✔	Correct	Detalls	Detalls
3698	2. PAC JWT	2.1. token	Projecte_4036	20:29:11 19-05-2025	20:29:31 19-05-2025	✔	Correct	Detalls	Detalls

e) If we compare the process with the brief explanation of the section, we can first understand who represent each role described:

Role	**Description**	**Role in our example**
Resource Owner	The user who owns the data or resource	We as authenticated users. **e107@uoc.edu, TfM2023,**
Client	The app that wants to access the resource	Our script with the curl operations (**token.sh** and **square.sh**)
Authorization Server	Verify the identity and generate tokens	**labxarxes.techlab.uoc.edu:8095, endpoint /token**
Resource Server	Have the protected resource	**labxarxes.techlab.uoc.edu:8095, endpoint /square/2**

Regarding the HTTP request and response obtained while capturing packets with Wireshark and comparing with the expected headers:

1. In the **POST /dslab-api/token**

   Expected headers:

   - Authorization: Basic <base64(username:password)>
   - Custom tenant ID header (application-specific)
   - Host and User-Agent (standard HTTP)

   Captured headers:

   - Authorization: Basic ….
     Credentials: e107@uoc.edu: TfM2023,
   - X-TenantID: prova
   - Host: labxarxes.techlab.uoc.edu:8095
   - User-Agent: curl/...

2. In the **GET /dslab-api/xai/square/2**

   Expected headers:

   - Authorization: Bearer <access_token>

   Captured headers:

   - Authorization: Bearer JWT…

Sources:

- *https://www.geeksforgeeks.org/workflow-of-oauth-2-0/*
- *https://frontegg.com/blog/oauth (used the same figure as reference but recreated with word shapes and icons).*
- *https://fusionauth.io/dev-tools/jwt-decoder*
- *https://www.wallarm.com/what/oauth-vs-jwt-detailed-comparison*