

Format

Homework was made using React JS to create a website for both practical homework assignments. You can switch between the Assignments using the *Navbar Menu*.

Instructions

[Node.js](#) is required to launch the homework in a browser. When Node.js is installed and homework folder is extracted:

1. Go to the homework directory path in a terminal of your choice
2. Install all the required dependencies with *"npm install"*
3. Run *"npm start"*
4. The browser should open automatically with the home page loaded, if it doesn't then it should be available at *"localhost:3000"* or with some other port that *"npm start"* offers.

Assignment 1

Task description

- **Input:** An arbitrary length string, a key (or keys), (for CFB/OFB decryption) a MAC value (if stored separately), a choice of chaining mode and a choice of whether to encrypt or decrypt.
- **Output:** An file with encryption/decryption of the string under the specified chaining mode with the specified key (for CFB/OFB encryption – a file with MAC code, if you chose to store the code separately), (for CFB/OFB decryption) the information (could be in any format – shown in GUI, printed on screen, stored in file etc.) whether the MAC value is correct.

Naturally, decrypting an encrypted string should give back the clear text. For this task, you may use block ciphers from an external library, but **you must implement the chaining (both for encryption and MAC computation) and the special processing of the last block(s) by yourself**. In other words, you may only use a function that takes one block as input, performs regular encryption, and returns the encrypted block. You don't have to implement error handling of any kind – you can assume that the format of the input is correct, and it is perfectly all-right to crash on invalid input!

File structure

Here is a list of important files for this assignment.

src/components/

This folder contains components used for both assignments

- *FileTransfer.jsx* – contains *FileUpload* & *FileDownload* components, **conversion to downloaded file format**

src/Assignment1/

- *Task.md* – Task description
- *Assignment1.jsx* – Contains just the Tab system to switch between Task description, CBC & CFB

src/Assignment1/components

- *CBCMode.jsx* – **CBC encryption & decryption input (iv, key & message)**
- *CFBMode.jsx* – **CFB encryption & decryption input (iv, key, mac & message)**
- *Cryption.jsx* – **Encryption/Decryption & MAC calculation**

Solution for CBC

Encryption

Input is a JSON object containing {iv, key, message} and converts it to Uint8Array buffers.

The algorithm takes a block of bits as input to encrypt until no there's not enough for a full block (message too short).

Padding for the last block is filled with bytes of missing block length, for example, if the last block of a message is only 9 characters, then the padding will be filled with the byte "7".

After padding the last block is encrypted and sent to output in hex format, which can then be copied back to message field to decrypt or downloaded to binary format.

Decryption

Input is a JSON object containing {iv, key, message} and converts it to Uint8Array buffers.

The algorithm decrypts all but the last block normally and the padding includes the length of the padding itself, so if the last bit is <16, the size for the padding, then there's padding, otherwise the message was already a full block length and didn't require any padding.

Formats


- Initialization vector – 16-bit hex filled with all 0s
- Key – 16-bit hex with random generation & file upload
- Input message – UTF-8 or hex, if encrypting or decrypting. Upload can be from binary file and the program automatically translates it into readable hex code
- Decrypted output – UTF-8 (or hex if double encrypted)
- Encrypted output – Hex in output field, but downloaded to binary format.


Cipher-Block Chaining Mode (AES-CBC)


Initialization vector

00000000000000000000000000000000

KEY


 227bec8b710b529aad8cafa7a0869d6








MESSAGE

Assignment 1 Cipher-Block Chaining Mode:
test Message







 ENCRYPT


 DECRYPT

ENCRYPTED

789da7d61c043c3ba8af971b7c53a4e81148c0ff037f96e893eb909
a4e42b3e8ad735e307588b2c855f901c3a4d63b8bc12fa6808998
a9ef0ea4e70bdefa251







Solution for CFB

Encryption and decryption uses the same functions as CBC mode, with the algorithm mode set to *AES-CFB*. MAC is calculated after encryption with the *macKey* & *message*. When decrypting, the MAC is calculated again and checked with already calculated MAC and returns *true* or *false* if it matches or not.

Formats


- IV – 16-bit hex with random generation & file upload
- Key – 16-bit hex with random generation & file upload
- macKey – 16-bit hex with random generation & file upload
- Input message – UTF-8 or hex, if encrypting or decrypting. Upload can be from binary file and the program automatically translates it into readable hex code
- Decrypted output – UTF-8 (or hex if double encrypted)
- Encrypted output – Hex in output field, but downloaded to binary format
- Mac – 16-bit hex, when encrypting, true or false when decrypting

Cipher Feedback Chaining Mode (AES-CFB)


IV

7898ed548b6e1908e03d6635528955b2

KEY


 9fc23844a653f1f527489d37da4080ef

MACKEY

 204315ba649a6f4a8d9da269d1246aa3

MESSAGE

Assignment 1 Cipher Feedback Chaining Mode:
test Message





ENCRYPT


DECRYPT

DECRYPTED

Assignment 1 Cipher Feedback Chaining Mode:
test Message

MAC

 c1e13d103cb39bdafa2da44fd7f78826

Assignment 2

Task description

Your task consists of the following 3 steps:

1. Write a program that creates a valid X.509 standard compliant root (i.e. self-signed) certificate on your name and issued by yourself (you can freely choose the values for other “meaningful fields”). The certificate should sign your public encryption key of **RSA** algorithm (desirably with **at least 2048 bit** key) and the certificate itself should be signed using **RSA** and **SHA-2** message digest algorithm.
2. Write a program that verifies your certificate (since it is a root certificate, it is sufficient to check whether certificate issuer and subject are the same and whether a digital signature of the issuer matches the subject’s public key).
3. Write a program that encrypts and decrypts a message of suitable length using the public RSA key from the certificate/private key file that you have created and one of RSA encryption/padding schemes specified in PKCS #1 standard.

File structure

Here is a list of important files for this assignment.

src/actions/

- *actions.jsx* – contains the redux actions for reducer storage

src/reducers/

- *reducers.jsx* – contains the certificate reducers, **Certificate creation & upload from file**

src/components/

This folder contains components used for both assignments

- *FileTransfer.jsx* – contains FileUpload & FileDownload components, **conversion to downloaded file format**

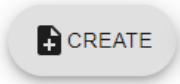
src/Assignment2/


- *Task.md* – Task description
- *Assignment2.jsx* – Contains just the Tab system to switch between Task description, Certification & Encryption/Decryption



src/Assignment2/components

- *Certification.jsx* – The outline of the certificate creation/verification processes
- *Create.jsx* – Contains dispatch functions to redux to create & upload certificates.
- *Cryption.jsx* – **Contains encryption & decryption functions**
- *Panel.jsx* – Contains the panels for PEM formatted output
- *Verify.jsx* – **Contains certificate verification**

Creating a certificate

To create a certificate go to the CERTIFICATION tab and click the  button or upload a

 certificate by clicking the button. After certificate creation, you can view all the PEM formatted outputs (certificate, private key, public key) and they're available for copying or downloading, or uploading each file (certificate, private key, public key) separately. If you import a certificate from a file, then you need to upload the private key yourself.




certificate


certificate

-----BEGIN CERTIFICATE-----
MIIDPjCCAIagAwIBATANBgkqhkiG9w0BAQsFADArMRIwEAYDVQQDEwlsb2Nh
bGhvc3QxFTATBgNVBAoTDE9sYWZlEVnbGFqczAeFw0yMDAxMTMwODA5NTJaFw0y
MTAxMTMwODA5NTJaMCsxEjAQBgNVBAMTCWxvY2FsaG9zdDEVMBMGA1UEChMMMT2xh
ZnMgRWdsYWpzMIIBJjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAmIXaZbCR
c7ez4VZXqfKF5Ax4A1zcSTNhS1RI7UI9wF5Nlva1AoGXjOtvwyQPeJNPspSqJf2+
Qdpo2YTErOI3ZR9ZVGmqrOJnu4yd2hGYowliBzwQ2RBAVXXHCyLxbexiDWB4Dw3o
BI5nOLVU9cVBEB9reA+Gop3QRJKmV7n3Hgr9XPZkoyloNK1/1iaRx98z+YHDdKuh
B6/lcTwjZOBIECF3KEad07IQSATURYwDpFNnMpsxrVxJL3zqDBuUpRzloyTTRjZo
+E51byxJ+kZQI28iWcFUSYO743Leri2TdGSMHxYhO4oQPceuiAvmdibzN4cGQdX8
nAlmqXnJpFh8iwlDAQABo20wazAMBgNVHRMERTADAAQH/MAsGA1UdDwQEAwIC9DA7
BgNVHSUENDAyBggrBgEFBQcDAQYIKwYBBQUHAwIGCCsGAQUFBwMDBggrBgEFBQcD
BAYIKwYBBQUHAwgewEQYJYIZIAyb4QgEBBAQDAgD3MA0GCSqGSIb3DQEBCwUAA4IB
AQB4+zfFZ5NB553Vkan8E+L/lj8xbWUgYdb0/PrPrDrQGpcVcjlbyFu+CxOR0Rx
ImYyHk4Q9PobmsVvMHTrggNHmbMh3PGv2CWHnEMHbs198ieqfDhGN2kkkOf31zFz
adNeUm0wYj5R6duGVhnguCdIG+S1j+Er7J1zPe6d+WKUQAIZx3W3SVxPMtdfgm3r
EwwY8vvgztIvAlvIo72OaSQqm3BdS1MAHJ6UDXhvlgsHMR42XLvF3W3HqC0RzrjM
OV0jypsW2RJUNINxrmX3Ik5IEkgjQ7dNTd2kOczJG1DiaKPezON+3KNI+7ug/eDZ
R5gd1t0AalBpY2qTYssIuw2O
-----END CERTIFICATE-----


COPY

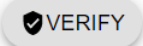


privateKey



publicKey






The code for certificate creation & upload is in *src/reducers/reducers.jsx*. The certificate uses RSA algorithm with 2048 bit keys. It's signed using generated the private key and SHA256 message digest algorithm. The certificate and it's private key are stored in a redux store and saves it even after a tab change, but not after a reload.

Verifying a certificate

Verification happens in the same tab as certificate creation – CERTIFICATION.

 VERIFY

To verify the certificate click on the button under all the panels and it should

 REFUTED

or

 VERIFIED

change to either depending on whether the certificate is verified or not. **The code for certificate verification is in `src/Assignment2/components/Verify.jsx`.**



Encrypting & Decrypting a message

Encryption & Decryption follows the same structure as Assignment 1. There's inputs that are available to edit and upload from files. You can also create and import a certificate from this tab. Output can be downloaded or copied to message field for Decryption. There's also an option to upload keys from inside the panels.

Formats


- Input message – UTF-8 or hex, if encrypting or decrypting. Upload can be from binary file and the program automatically translates it into readable hex code.
- Public key – must be PEM format
- Private key – must be PEM format
- Decrypted output – UTF-8 (or hex if double encrypted)
- Encrypted output – Hex in output field, but downloaded to binary format.

Encrypt & Decrypt

From Certificate:  CREATE  IMPORT

MESSAGE

Assignment 2: Crypton with Certificate public key:
test Message





publicKey

PUBLICKEY


```
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAgjEFuyMufXXiYphP204
oZ3EYdtES/G2kFd6X9Uz9RWB+yXz8X2X6PVaOW5k/+GEVSnXikle1QRzylIKxVx
EFRUbHUoYf8HGJRj7K8j7G3hbyAepCSxhOZ1SJEIL+wQ2p6HuXK9QJjwnd6b6S
dBqS9UNIS64PtosXyoAlkkG8b5Lws0A69ho0/Qy8sTHCoSsJavcWn6scDI0XHbSt
cfMGIAW0D6232pSvXK7DQkh9WyalXjHkGXuK2L3R70RV0Y0YC4gNpOxlwZIZF6w
K6QwdkFkru1zp3s8yE6Rt2pAnSESFwgtFHEWfVfJDhfhHnGqO5/KoSoS+XA6y5d
9QIDAQAB
-----END PUBLIC KEY-----
```



COPY





privateKey



 ENCRYPT  DECRYPT

DECRYPTED

Assignment 2: Crypton with Certificate public key:
test Message

