



T.C.
FATİH SULTAN MEHMET VAKIF ÜNİVERSİTESİ

**BLM442E Computer System Security
Homework Report**

Nagkichan MOUSTAFA

Lecturer : Dr. Ömer KORÇAK

About

This report covers several different encryption algorithms as part of the BLM442E Computer System Security assignment. Tasks include creating RSA key pairs, symmetric keys, applying a SHA-256 hash, signing messages, and encrypting/decrypting an image file using AES in different modes. Details of each task are given below, along with code and results.

I created a project using HTML and JavaScript. The interface is built with HTML, while the cryptology functionality is implemented in JavaScript. The project features a simple design, consisting of buttons and text elements.

When the project is first run, the initial screen appears as shown below. The screen is minimalistic and user-friendly, allowing users to easily navigate and perform the desired operations.

Upon pressing the relevant buttons, the associated operations are executed. Below, I provide a detailed explanation of the functionalities and operations available in the project.

Cryptography Project

Generate RSA Keys

Generate Symmetric Keys

Encrypt Symmetric Keys

Decrypt Symmetric Keys

Enter a message for signing...

Sign Message

Verify Signature

No file chosen

Encrypt and decrypt the selected image using AES-CBC-128:

Encrypt Image (AES-CBC-128)

Encrypt and decrypt the selected image using AES-CBC-256:

Encrypt Image (AES-CBC-256)

Encrypt and decrypt the selected image using AES-CTR-256:

Encrypt Image (AES-CTR-256)

Decrypted Image:

Symmetric Keys

I created three symmetric keys using the **generateSymmetricKeys** function. When click on the button ‘Generate Symmetric Keys’ generates 3 keys. These keys are as follows:

1. **K1**: 128-bit AES-CBC
2. **K2**: 256-bit AES-CBC
3. **K3**: 256-bit AES-CTR

Generate Symmetric Keys

```
Generated Symmetric Keys:  
K1 (AES-CBC-128): 372a74df3f76666b0da0b0985089ac29  
K2 (AES-CBC-256): 08620373ed0eda60a7aff9f168c55dd1f54c33389cf7192db1307f19b5a8cfac  
K3 (AES-CTR-256): 000d10dc3d3f7868b8a557650d7dc6351625a236c5689b289fd1f863987b14ae
```

The ‘Encrypt Symmetric Keys’ button performs encryption. Then, to decrypt k1 and k2, I decrypt k1 and k2 by pressing the decrypt symmetric keys button. I performed encryption and decryption operations on these keys as follows:

1. **Encryption:**

Encrypt Symmetric Keys

```
Encrypted Symmetric Keys:  
Encrypted K1: 7fe31a04f4f4ee22e39e3cf4330f917fc17d4dbf272388333437fb427fa28f783d06543453010ea6b7ddd8bc  
b66109be5b2fa57563833c0738cac3189d124b07b233eed18fc1ebe00a47373e18feefda3c6994cf5655d00f56a3e9430ce8  
4b7f0fb7400517b4fac0c7e97cf8d1d3a0ed88aa8938b72e956a01d594f10ebd213cd22eee65d50ab5eb6181ceb94217845ba  
3de4c6ed65eb4006fcda45752a61d5c182f13d059503e74115d712b3466a111bc88b6d703afac75eff041de38ac303e8e2987b6  
6da480e5dd620e8533e5f746a17c698a190440835b68caa4b19e1f7a77624d450ab164c8bed3aa2e9e1321ccb245571b627d3f  
138b5cd9d1bf7357  
Encrypted K2: b64095eb2ead734d471434defabcaa22ff3d286d9723d35c6041d00f206db300b449bf9632a66dcfb7d0ef67  
cf93cef1e520d12903eca6a674fc54eb9d503f0e0816da993965a41f8026833b5e2bf0c9540461990334753d009b63d532bdb3  
7825bc5b198af1ee7a0fb5aa2ad16b03de0a7d66bad2d0f9cd42ac6228e5d8f65d0b77152bf4d063da3d34fb47cdd0ac21d79b  
b430139e54b184bbe61c83b91688d710fd089c79970b177a277f8db2d6858f3d8ce85003cfdde48e2a2ba0bc6aaa42adfd2a51  
f54e0a4b981ae8c12b265c2b66bbfa2c8b2da8509644dc9767679df0b8e4e30dcdb05567428f7fd209aefb95cd274ec0f18f96  
9b473a2d11cc614e
```

2. **Decryption:**

Decrypt Symmetric Keys

```
Decrypted Symmetric Keys:  
Decrypted K1: 372a74df3f76666b0da0b0985089ac29  
Decrypted K2: 08620373ed0eda60a7aff9f168c55dd1f54c33389cf7192db1307f19b5a8cfac
```

Message

First of all, I get the message. The text appearing at the top is my message. Then when I press the 'Sign Message' button, the following steps occur:

Sign Message

Message:
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Curabitur fringilla hendrerit nulla, eu ultrices nisi maximus et. Pellentesque finibus felis eget felis aliquam consequat. Pellentesque non bibendum lacus. Proin sit amet nisl vel eros dapibus euismod. Sed gravida, arcu a imperdiet hendrerit, nunc erat dapibus purus, ut pharetra quam enim in sapien. Nullam interdum sem eget placerat dignissim. Nunc laoreet ut nulla at mollis. Mauris vel lorem varius, auctor ipsum sit amet, tincidunt orci. Phasellus mi mauris, gravida sed eleifend rutrum, fermentum at quam.

In fringilla elementum velit vitae vehicula. Proin tincidunt sem ut orci dictum placerat at eu justo. Vestibulum accumsan, massa sed auctor tincidunt, nulla orci suscipit lectus, lacinia tristis odio mi nec erat. Nunc et accumsan ante. Nunc congue bibendum tincidunt. In consequat purus quis est ullamcorper consectetur. Aliquam in lobortis augue, egestas interdum erat. Proin dapibus augue ligula.

Pellentesque scelerisque eu ex in porta. Mauris ac interdum lorem. Sed eros nisl, consequat a nulla nec, ultricies sodales est. Maecenas sit amet bibendum sapien, non pulvinar sem. Phasellus malesuada auctor leo ut tincidunt. Morbi tincidunt suscipit lectus, et rhoncus dolor rhoncus et. Suspendisse accumsan libero a congue faucibus. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. In hac habitasse platea dictumst. Fusce varius pharetra maximus. Donec viverra urna sed scelerisque rhoncus. Proin convallis nibh eu sapien porttitor tristique.

Phasellus mollis euismod justo, eu semper odio aliquet quis. Duis gravida auctor congue. In massa metus, aliquam et pretium vel, congue id ex. Nullam viverra, dolor tempor consequat scelerisque, turpis duis porta ex, ut hendrerit augue mauris at odio. Mauris eu commodo leo, sed efficitur quam. Vivamus ac elit egestas metus tempor suscipit aliquet vitae elit. Duis at feugiat nunc, ut tempor mauris. Integer lacinia lectus ut tempor porta. Vestibulum vehicula, est at commodo tincidunt, odio eros fermentum magna, eu venenatis urna eros non ex. Proin posuere lectus sit amet diam porta scelerisque.

In fringilla elementum velit vitae vehicula. Proin tincidunt sem ut orci dictum placerat at eu justo. Vestibulum accumsan, massa sed auctor tincidunt, nulla orci suscipit lectus, lacinia tristis odio mi nec erat. Nunc et accumsan ante. Nunc congue bibendum tincidunt. In consequat purus quis est ullamcorper consectetur. Aliquam in lobortis augue, egestas interdum erat. Proin dapibus augue ligula.

Pellentesque scelerisque eu ex in porta. Mauris ac interdum lorem. Sed eros nisl, consequat a nulla nec, ultricies sodales est. Maecenas sit amet bibendum sapien, non pulvinar sem. Phasellus malesuada auctor leo ut tincidunt. Morbi tincidunt suscipit lectus, et rhoncus dolor rhoncus et. Suspendisse accumsan libero a congue faucibus. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. In hac habitasse platea dictumst. Fusce varius pharetra maximus. Donec viverra urna sed scelerisque rhoncus. Proin convallis nibh eu sapien porttitor tristique.

Phasellus mollis euismod justo, eu semper odio aliquet quis. Duis gravida auctor congue. In massa metus, aliquam et pretium vel, congue id ex. Nullam viverra, dolor tempor consequat scelerisque, turpis duis porta ex, ut hendrerit augue mauris at odio. Mauris eu commodo leo, sed efficitur quam. Vivamus ac elit egestas metus tempor suscipit aliquet vitae elit. Duis at feugiat nunc, ut tempor mauris. Integer lacinia lectus ut tempor porta. Vestibulum vehicula, est at commodo tincidunt, odio eros fermentum magna, eu venenatis urna eros non ex. Proin posuere lectus sit amet diam porta scelerisque.

SHA-256 Hash: 73c4ffff5a36c000208b89d4e2352089824bbafec5357b4e0997e3201164251fe

Digital Signature: 805221571db30dcba4dc8b89e13479dea5de888c017e6334638c1653efa34df4c2ffb24e653facef7938fa1f2311e68256854e47c84f8d83d146bf7ff3437f020df8882f89a35fc2bfe50a4050a3b795e3f2a0ab06d64a12e879f682437f7b81acbcc3a3da2863b3ec91cf98cbc277f33665a4581969de21d7d9fd046dbf5f61cb3fb6a5030d0914fc02bf7d4f755bb7b00669629f0941b1ea4b76af6ee25d49ef10ec0be0139935dc5dab2712569de6de8cc6e1a7049d7d2da086b893115f4d082ea4c51975b815bf3c78e2966d9a01d0288b70777d1c7e6f7d5dc97c2a642e9b6c6fec3ab76ceab9cb6205f9d646b71a3e29a6c79748a7d4a9572aac3ea

Verify Signature

Digital signature verification: true
If the digital signature is verified, it indicates that the message was indeed signed by the private key holder and the message has not been altered.

- 1) **Hash the Message:** The message is processed using the SHA256 hash algorithm to produce a message digest. This digest is a fixed-length representation of the message, ensuring its integrity.
- 2) **Create the Digital Signature:** The message digest is then encrypted using the private RSA key (K_A^-) to generate the digital signature. This digital signature is unique to the message and the private key used.

3) **Display the Results:** The original message, its SHA256 hash, and the digital signature are displayed on the screen.

- Hash the Message Again:** The original message is hashed again using the SHA256 algorithm to produce a new message digest.
- Decrypt the Signature:** The digital signature is decrypted using the public RSA key (KA^+), recovering the original message digest.
- Compare the Hashes:** The newly computed message digest is compared with the decrypted message digest. If they match, the signature is valid, confirming that the message was not altered and was indeed signed by the owner of the private key.

```
async function signMessage() {
  try {
    let message = document.getElementById('message').value;
    let encoder = new TextEncoder();
    let data = encoder.encode(message);

    messageHash = await window.crypto.subtle.digest("SHA-256", data);

    signature = await window.crypto.subtle.sign({ name: "RSA-PSS", saltLength: 32 }, rsaKeyPairPSS.privateKey, messageHash);

    const hashHex = bufferToHex(new Uint8Array(messageHash));
    const signatureHex = bufferToHex(new Uint8Array(signature));

    document.getElementById('signature-output').innerText = `Message:\n${message}\nSHA-256 Hash: ${hashHex}\nDigital Signature: ${signatureHex}`;

    saveToServer("hash_and_signature.txt", `Message:\n${message}\nSHA-256 Hash: ${hashHex}\nDigital Signature: ${signatureHex}`);
  } catch (error) {
    console.error("Error signing message:", error);
  }
}

async function verifySignature() {
  try {
    let message = document.getElementById('message').value;
    let encoder = new TextEncoder();
    let data = encoder.encode(message);
    let hash = await window.crypto.subtle.digest("SHA-256", data);

    let isValid = await window.crypto.subtle.verify({ name: "RSA-PSS", saltLength: 32 }, rsaKeyPairPSS.publicKey, signature, hash);
    document.getElementById('verification-output').innerText = `Digital signature verification: ${isValid}\nIf the digital signature is verified, it indicates that the message has not been tampered with.`;
  } catch (error) {
    console.error("Error verifying signature:", error);
  }
}
```

The digital signature, along with the message and its hash, is saved to a file named **hash_and_signature.txt**. This file is uploaded to the server using the **server.js** script. The relevant part of the **server.js** script handles the file upload and saves it to the server directory.

```
app.post('/save-hash-signature', (req, res) => {
  if (!req.files || Object.keys(req.files).length === 0) {
    return res.status(400).send('No files were uploaded.');
  }

  let hashSignatureFile = req.files.file;
  let savePath = path.join(__dirname, 'hash_and_signature.txt');

  hashSignatureFile.mv(savePath, (err) => {
    if (err) return res.status(500).send(err);
    res.send('Hash and signature file saved!');
  });
});
```

Image Encryption

I used a jpg of approximately 2MB in size to test. First, I select the photo from the files. I added the photo I used to the project file (image.jpg). Then I ran it separately for 3 different modes. I did both encrypt and decrypt in all of them. The photo above is the photo I chose from the files and uploaded. The decrypted photo under the heading 'Decrypted Image:'. Again, I use **server.js** and save it in the project file as **encrypted_image.bin** and **decrypted_image.jpg**

Let's examine the modes one by one:

1. AES (128 bit key) in CBC mode

The picture above is selected by user. When click on the button 'Encrypt Image (AES-CBC-128)' retuns the Image Size and Encryption Time.



Encrypt and decrypt the selected image using AES-CBC-128:

Encrypt Image (AES-CBC-128)

Encrypt and decrypt the selected image using AES-CBC-256:

Encrypt Image (AES-CBC-256)

Encrypt and decrypt the selected image using AES-CTR-256:

Encrypt Image (AES-CTR-256)

Image encrypted and decrypted successfully.
Original size: 2010740 bytes
Encrypted size: 2010740 bytes
Decrypted size: 2010740 bytes
Encryption time: 3.200000047683716 ms

Decrypted Image:



```
Image encrypted and decrypted successfully.  
Original size: 2010740 bytes  
Encrypted size: 2010740 bytes  
Decrypted size: 2010740 bytes  
Encryption time: 3.200000047683716 ms
```

Encryption time for CBC 128 bits key is ~ 3. 2000 ms.

And the picture below is the decrypted picture of the selected encrypted picture.

2. AES (256 bit key) in CBC mode.



Encrypt and decrypt the selected image using AES-CBC-128:

[Encrypt Image \(AES-CBC-128\)](#)

Encrypt and decrypt the selected image using AES-CBC-256:

[Encrypt Image \(AES-CBC-256\)](#)

Encrypt and decrypt the selected image using AES-CTR-256:

[Encrypt Image \(AES-CTR-256\)](#)

```
Image encrypted and decrypted successfully.  
Original size: 2010740 bytes  
Encrypted size: 2010752 bytes  
Decrypted size: 2010740 bytes  
Encryption time: 4.5 ms
```

Decrypted Image:



Image encrypted and decrypted successfully.
Original size: 2010740 bytes
Encrypted size: 2010752 bytes
Decrypted size: 2010740 bytes
Encryption time: 4.5 ms

Encryption time for CBC 256 bits key is 4.5 ms.

3. AES (256 bit key) in CTR mode.



Encrypt and decrypt the selected image using AES-CBC-128:

[Encrypt Image \(AES-CBC-128\)](#)

Encrypt and decrypt the selected image using AES-CBC-256:

[Encrypt Image \(AES-CBC-256\)](#)

Encrypt and decrypt the selected image using AES-CTR-256:

[Encrypt Image \(AES-CTR-256\)](#)

Image encrypted and decrypted successfully.
Original size: 2010740 bytes
Encrypted size: 2010752 bytes
Decrypted size: 2010740 bytes
Encryption time: 6.200000047683716 ms

Decrypted Image:



```
Image encrypted and decrypted successfully.  
Original size: 2010740 bytes  
Encrypted size: 2010752 bytes  
Decrypted size: 2010740 bytes  
Encryption time: 6.200000047683716 ms
```

Encryption time for CTR 256 bits key is ~6.200 ms.

Results for Image Encryption:

Mode (AES)	Bit	Time (ms)
CBC	128 bits	3.20000004768716
CBC	256 bits	4.5
CTR	256 bits	6.20000004768716