Experiment No. 5

Title: Vlab on Message Authentication Codes

Batch: A3 **Roll No.: 16010421059** **Experiment No.:05**

Title: Illustrate and implement message authentication code.

Resources needed: Windows/Linux OS

Theory:

In cryptography, a cipher block chaining message authentication code (CBC-MAC) is a technique for constructing a message authentication code (MAC) from a block cipher. The message is encrypted with some block cipher algorithm in cipher block chaining (CBC) mode to create a chain of blocks such that each block depends on the proper encryption of the previous block. This interdependence ensures that a change to any of the plaintext bits will cause the final encrypted block to change in a way that cannot be predicted or counteracted without knowing the key to the block cipher.

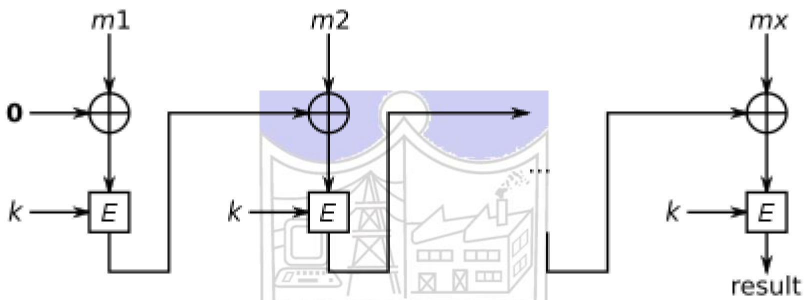


Figure 1 - CBC-MAC construction

To calculate the CBC-MAC of message m, one encrypts m in CBC mode with zero initialization vector(IV) and keeps the last block. The figure 1 sketches the computation of the CBC-MAC of message comprising blocks m1, m2,…mx using a secret key k and a block cipher E.

Activity :

1) Perform the Vlab on MAC - https://cse29-iiith.vlabs.ac.in/exp/message-authentication-codes/index.html
2) Implement the similar vlab simulation with a simple block cipher in CBC mode with following details-
   - Plain text message M = user's choice (string type)
   - Block Size = user's choice (must be < (length of M_2)/2)
   - Key k = user's choice (length of key is same as block size)
   - IV = user's choice (length of IV is same as block size)
   - E = XOR function

INLAB:

<!DOCTYPE html>

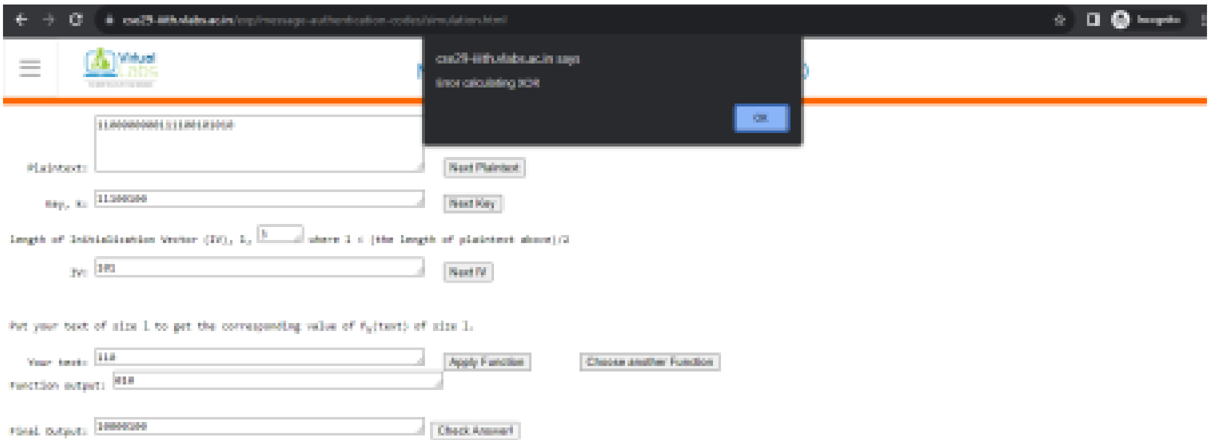
<html lang="en">

```
<head>

 <meta charset="UTF-8">

 <meta name="viewport" content="width=device-width, initial-

scale=1.0">  <title>CBC-MAC Verification</title>

</head>

<body>

 <h1>CBC-MAC Verification</h1>

 <form>

 <label for="plain">Plain Text (Binary):</label>

 <input type="text" name="plain" pattern="[0-1]+" placeholder="Enter binary plain
text"  required oninput="calculateCBCMAC()"><br><br>

 <label for="key">Key (Binary):</label>
 <input type="text" name="key" pattern="[0-1]+" placeholder="Enter binary key"
required  oninput="calculateCBCMAC()"><br><br>

 <label for="vec">Length of Initialization Vector (IV) &lt; (Length of plain text/2):</label>

 <input type="text" name="vec" placeholder="Enter IV length"
required  oninput="calculateCBCMAC()"><br><br>

 <label for="IV">IV (Binary):</label>

 <input type="text" name="IV" pattern="[0-1]+" placeholder="Enter binary IV"
required  oninput="calculateCBCMAC()"><br><br>

 <label for="text">Function Output (Binary):</label>

 <input type="text" name="output" placeholder="Function output will be displayed
here"  readonly><br><br>

 <label for="text">Final Output (Binary):</label>

 <input type="text" name="final" placeholder="Final output will be displayed
here"  required><br><br>

 <input type="submit" value="Submit">

 </form>

 <script>

 function calculateCBCMAC() {
```

```
 const ivLength = parseInt(document.querySelector('input[name="vec"]').value,
10);  const ivInput = document.querySelector('input[name="IV"]').value;  const
plainInput = document.querySelector('input[name="plain"]').value;  const key =
'11100100'; // Replace with your key

 if (ivLength <= 0 || ivLength > plainInput.length / 2) {
 alert('Invalid IV length.');
 return;
 }

 const numBlocks = Math.ceil(plainInput.length / ivLength);
 let encryptedResult = '';

 for (let blockIndex = 0; blockIndex < numBlocks; blockIndex++)
{  const start = blockIndex * ivLength;
 const end = (blockIndex + 1) * ivLength;
 const blockIV = ivInput.substring(start, end);
 const blockPlain = plainInput.substring(start, end);

 if (blockIV.length !== blockPlain.length) {
 alert('IV and Plain Text blocks must have the same
length.');  return;
 }

 let xorResult = '';

 for (let i = 0; i < blockIV.length; i++) {
 xorResult += blockIV[i] === blockPlain[i] ? '0' : '1';
```

```
}

for (let i = 0; i < xorResult.length; i++) {
encryptedResult += xorResult[i] === key[i] ? '0' : '1';  }

}

document.querySelector('input[name="output"]').value =
encryptedResult;  }

</script>
</body>

</html>
```

VLAB ERROR:



Questions:

1) Compare MAC and cryptographic Hash functions.

MAC (Message Authentication Code) and cryptographic hash functions serve different purposes in cryptography, although they share some similarities. Here's a comparison of the two:

1. Purpose:

- MAC (Message Authentication Code): MACs are used to ensure the integrity and authenticity of a message. They provide a way to verify that the message hasn't been tampered with and that it was indeed generated by the expected sender. MACs are typically used for verifying the authenticity of messages in network communications or for protecting the integrity of data.

- Cryptographic Hash Function: Cryptographic hash functions are primarily used for data integrity, data structure verification, and data storage. They take an input (message) and produce a fixed-size output (hash) that is a unique representation of the input data. Hash functions are used in a wide range of applications, including password hashing, digital signatures, and data deduplication.

2. Keyed vs. Unkeyed:

- MAC: MACs require a secret key shared between the sender and receiver. They use both the message and the key to produce the MAC. This makes MACs suitable for authentication and integrity verification in scenarios where both parties possess the same key.

- Cryptographic Hash Function: Hash functions are typically unkeyed and produce a fixed-size hash of the input data. They do not require a secret key and can be used publicly. Their primary purpose is to ensure the integrity of the data, but they do not provide authenticity or protect against unauthorized modifications unless used in conjunction with a digital signature.

3. Output Size:

- MAC: The output size of a MAC is often fixed and can be chosen to be larger for stronger security. It's typically designed to be more resilient against collision attacks.

- Cryptographic Hash Function: Hash functions have a fixed output size, and while they provide a unique representation of input data, they can suffer from collision attacks where two different inputs produce the same hash value. This is why they are not suitable for authentication without additional measures.

4. Security Properties:

- MAC: MACs are designed to provide both data integrity and authenticity when used with a shared secret key. They are resistant to attacks by adversaries who do not know the key.

- Cryptographic Hash Function: Hash functions are resistant to preimage attacks (finding an input for a given hash) and second preimage attacks (finding a different input that hashes to the same value). However, they are not designed for authenticity and can't protect against a determined attacker who can manipulate the data and hash.

In summary, MACs are designed for message authentication and integrity verification, requiring a shared secret key. Cryptographic hash functions, on the other hand, are primarily used for data integrity and data structure verification and do not require a key. While they share some mathematical properties, their purposes and use cases are different.

Outcomes:

CO 2: Illustrate different cryptographic algorithms for security.

Conclusion:

Implemented MAC successfully.

Grade: AA / AB / BB / BC / CC / CD /DD

Signature of faculty in-charge with date

References:

Books/ Journals/ Websites:

- William Stallings, "Cryptography and Network Security" by Pearson Education 4th Edition 2014.