University of New Brunswick, Faculty of Computer Science

CS4355/6355: Topic 1 – Additional Note

1 Simple Substitution Ciphers

As Julius Caesar surveys the unfolding battle from his hilltop outpost, an exhausted and disheveled courier bursts into his presence and hands him a sheet of parchment containing gibberish:

j s j r d k f q q n s l g f h p g w j f p y m w t z l m n r r n s j s y q z h n z x

Within moments, Julius sends an order for a reserve unit of charioteers to speed around the left flank and exploit a momentary gap in the opponentŠs formation.
How did this string of seemingly random letters convey such important information?

Please use the simple substitution cipher: $ciphertext = plaintext + key \mod 26$ to recover the plaintext of the string and the used key, and explain why.

University of New Brunswick, Faculty of Computer Science

CS4355/6355: Topic 2 – Additional Note

1 A Simple Java Source Code for Calling AES Algorithm

```
import javax.crypto.Cipher;
import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;
import javax.crypto.spec.SecretKeySpec;
import java.security.NoSuchAlgorithmException;
import java.security.SecureRandom;
import javax.crypto.*;
import java.security.InvalidKeyException;
import java.io.UnsupportedEncodingException;

public class AES {
    /**
     * @Param [content, password]
     * @return byte[]
     * @Description:  AES Encryption One String
     **/
    public static byte[] encrypt(String content, String password) {
        try {
            // Create AES Key Generator
            KeyGenerator kgen = KeyGenerator.getInstance("AES");
            // User's pasword as the random number to generate the 128-bit key
            // SecureRandom generates the secure random sequence,
            // password.getBytes() is the seed.
            // Only if the seeds are the same, the sequences should be the same.
```

```
            // Note that, the security is based on the password.
            // It is not secure in reality.
            SecureRandom random = SecureRandom.getInstance("SHA1PRNG");
            random.setSeed(password.getBytes());
            kgen.init(128, random);
            // generate a secret key
            SecretKey secretKey = kgen.generateKey();
            // return the encoded secret key
            byte[] enCodeFormat = secretKey.getEncoded();
            // convert it to the AES secret key
            SecretKeySpec key = new SecretKeySpec(enCodeFormat, "AES");
            // generate the cipher instance
            Cipher cipher = Cipher.getInstance("AES");
            byte[] byteContent = content.getBytes("utf-8");
            // set the encryption mode
            cipher.init(Cipher.ENCRYPT_MODE, key);
            // runing the encryption.
            byte[] result = cipher.doFinal(byteContent);

            return result;

        } catch (NoSuchPaddingException e) {
            e.printStackTrace();
        } catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
        } catch (UnsupportedEncodingException e) {
            e.printStackTrace();
        } catch (InvalidKeyException e) {
            e.printStackTrace();
        } catch (IllegalBlockSizeException e) {
            e.printStackTrace();
        } catch (BadPaddingException e) {
            e.printStackTrace();
        }
        return null;
    }

    //Decryption
    public static byte[] decrypt(byte[] content, String password) {
        try {
            // Create AES Key Generator
            KeyGenerator kgen = KeyGenerator.getInstance("AES");

            SecureRandom random = SecureRandom.getInstance("SHA1PRNG");
            random.setSeed(password.getBytes());
            kgen.init(128, random);
            SecretKey secretKey = kgen.generateKey();
            byte[] enCodeFormat = secretKey.getEncoded();
            SecretKeySpec key = new SecretKeySpec(enCodeFormat, "AES");
```

```
            Cipher cipher = Cipher.getInstance("AES");
            // set the decryption mode
            cipher.init(Cipher.DECRYPT_MODE, key);
            byte[] result = cipher.doFinal(content);
            return result; // plaintext

        } catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
        } catch (NoSuchPaddingException e) {
            e.printStackTrace();
        } catch (InvalidKeyException e) {
            e.printStackTrace();
        } catch (IllegalBlockSizeException e) {
            e.printStackTrace();
        } catch (BadPaddingException e) {
            e.printStackTrace();
        }
        return null;
    }

    /**
     * @Param buf
     * @return String
     * @Description: Binary to HEX
     **/
    public static String parseByte2HexStr(byte buf[]) {
        StringBuffer sb = new StringBuffer();
        for (int i = 0; i < buf.length; i++) {
            String hex = Integer.toHexString(buf[i] & 0xFF);
            if (hex.length() == 1) {
                hex = '0' + hex;
            }
            sb.append(hex.toUpperCase());
        }
        return sb.toString();
    }

    /**
     * @Param String
     * @return byte[]
     * @Description: HEX to Binary
     **/
    public static byte[] parseHexStr2Byte(String hexStr) {
        if (hexStr.length() < 1)
            return null;
        byte[] result = new byte[hexStr.length()/2];
        for (int i = 0;i< hexStr.length()/2; i++) {
            int high = Integer.parseInt(hexStr.substring(i*2, i*2+1), 16);
            int low = Integer.parseInt(hexStr.substring(i*2+1, i*2+2), 16);
            result[i] = (byte) (high * 16 + low);
```

```
        }
        return result;
    }

    public static void main(String[] args) {
        String content = "Hello, I like CS6355/4355 :)";
        String password = "123";
        System.out.println("Plaintext:" + content);

        try{

            // encrypt
            byte[] encrypt = AES.encrypt(content, password);
            System.out.println("Ciphertext in Byte: ");
            System.out.println(new String(encrypt));

            // convert it to HEX
            String hexStrResult = parseByte2HexStr(encrypt);
            System.out.println("Ciphertext in HEX:");
            System.out.println(hexStrResult);

            //if HEX, convert it back to binary
            byte[] twoStrResult = parseHexStr2Byte(hexStrResult);
            encrypt = twoStrResult;

            // decrypt
            byte[] decrypt = AES.decrypt(encrypt, password);
            System.out.println("Recovered Plaintext:" + new String(decrypt));

        } catch (Exception e) {
            e.printStackTrace();
        }

    }
}

}
```

Running Result:

```
Plaintext:Hello, I like CS6355/4355 :)
Ciphertext in Byte:
?8???l??9WR1?A??Q
Ciphertext in HEX:
61EB08DF5E86EDFBEF7975980DA538C1AAF1B9836CAAB939575231ED41F8EB51
Recovered Plaintext:Hello, I like CS6355/4355 :)
```

2 A Java Source Code for Calling AES Algorithm to Encrypt A File

```
import java.io.BufferedInputStream;
import java.io.BufferedOutputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.UnsupportedEncodingException;
import java.security.InvalidKeyException;
import java.security.NoSuchAlgorithmException;
import java.security.SecureRandom;
import java.security.Security;
import java.util.Arrays;
import javax.crypto.Cipher;
import javax.crypto.CipherInputStream;
import javax.crypto.CipherOutputStream;
import javax.crypto.KeyGenerator;
import javax.crypto.NoSuchPaddingException;
import javax.crypto.SecretKey;
import javax.crypto.spec.SecretKeySpec;
/**
* @desc [file encryption]
 */
public class AESFileUtil {

    private static final String key = "password";

    /**
     * init AES Cipher
     * @param passsword
     * @param cipherMode
     * @return
     */
    public static Cipher initAESCipher(String passsword, int cipherMode) {
        Cipher cipher = null;
        try {
            SecretKey key = getKey(passsword);
            cipher = Cipher.getInstance("AES");
            cipher.init(cipherMode, key);
        } catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
            //To change body of catch statement use File | Settings | File Templates.
        } catch (NoSuchPaddingException e) {
            e.printStackTrace();
            //To change body of catch statement use File | Settings | File Templates.
        } catch (InvalidKeyException e) {
```

```
            e.printStackTrace();
            //To change body of catch statement use File | Settings | File Templates.
        }
        return cipher;
    }


    private static SecretKey getKey(String password) {
       // Create AES Key Generator

        try{
            KeyGenerator kgen = KeyGenerator.getInstance("AES");

            SecureRandom random = SecureRandom.getInstance("SHA1PRNG");
            random.setSeed(password.getBytes());
            kgen.init(128, random);
            SecretKey secretKey = kgen.generateKey();
            byte[] enCodeFormat = secretKey.getEncoded();
            SecretKeySpec key = new SecretKeySpec(enCodeFormat, "AES");
            return key;
        } catch (Exception e) {
            e.printStackTrace();}
        return null;
    }


    /**
     * AES encrypt
     * @param encryptPath
     * @param decryptPath
     * @param sKey
     * @return
     */
    public static boolean encryptFile(String encryptPath, String decryptPath, String sKey){
        File encryptFile = null;
        File decryptfile = null;
        CipherOutputStream cipherOutputStream = null;
        BufferedInputStream bufferedInputStream = null;
        try {
            encryptFile = new File(encryptPath);
            if(!encryptFile.exists()) {
                throw  new NullPointerException("Encrypt file is empty");
            }
            decryptfile = new File(decryptPath);
            if(decryptfile.exists()) {
                decryptfile.delete();
            }
            decryptfile.createNewFile();

            Cipher cipher = initAESCipher(sKey, Cipher.ENCRYPT_MODE);
```

```
            cipherOutputStream = new CipherOutputStream(new FileOutputStream(decryptfile), cipher);
            bufferedInputStream = new BufferedInputStream(new FileInputStream(encryptFile));

            byte[] buffer = new byte[1024];
            int bufferLength;

            while ((bufferLength = bufferedInputStream.read(buffer)) != -1) {
                cipherOutputStream.write(buffer, 0, bufferLength);
            }
            bufferedInputStream.close();
            cipherOutputStream.close();
            //   delFile(encryptPath);
        }  catch (IOException e) {
            delFile(decryptfile.getAbsolutePath());
            e.printStackTrace();
            //To change body of catch statement use File | Settings | File Templates.
            return false;
        }
        return true;
    }

    /**
     * AES decrypt
     * @param encryptPath
     * @param decryptPath
     * @param mKey
     * @return
     */

    public static boolean decryptFile(String encryptPath, String decryptPath, String mKey){
        File encryptFile = null;
        File decryptFile = null;
        BufferedOutputStream outputStream = null;
        CipherInputStream inputStream = null;
        try {
            encryptFile = new File(encryptPath);
            if(!encryptFile.exists()) {
                throw new NullPointerException("Decrypt file is empty");
            }
            decryptFile = new File(decryptPath);
            if(decryptFile.exists()) {
                decryptFile.delete();
            }
            decryptFile.createNewFile();

            Cipher cipher = initAESCipher(mKey, Cipher.DECRYPT_MODE);

            outputStream = new BufferedOutputStream(new FileOutputStream(decryptFile));
            inputStream = new CipherInputStream(new FileInputStream(encryptFile), cipher);
```

```
            int bufferLength;
            byte[] buffer = new byte[1024];

            while ((bufferLength = inputStream.read(buffer)) != -1) {
                outputStream.write(buffer, 0, bufferLength);
            }
            inputStream.close();
            outputStream.close();
            //  delFile(encryptPath);
        } catch (IOException e) {
            delFile(decryptFile.getAbsolutePath());
            e.printStackTrace();
            //To change body of catch statement use File | Settings | File Templates.
            return false;
        }
        return true;
    }


    /**
     * delete File
     * @param pathFile
     * @return
     */
    public static boolean delFile(String pathFile) {
        boolean flag = false;
        if(pathFile == null && pathFile.length() <= 0) {
            throw new NullPointerException("cannot be empty file!");
        }else {
            File file = new File(pathFile);
            // delete the path with file name and not empty
            if (file.isFile() && file.exists()) {
                file.delete();
                flag = true;
            }
        }
        return flag;
    }


    public static void main(String[] args) {
        boolean flag = AESFileUtil.encryptFile
                ("C:/text/rxlu.jpg", "C:/text/encrypted.jpg", key);
        System.out.println(flag);
        flag = AESFileUtil.decryptFile
                ( "C:/text/encrypted.jpg","C:/text/recovered.jpg", key);
        System.out.println(flag);

        flag = AESFileUtil.encryptFile
                ("C:/text/abc.txt", "C:/text/encrypted.txt", key);
```

```
        System.out.println(flag);
        flag = AESFileUtil.decryptFile
                ( "C:/text/encrypted.txt","C:/text/recovered.txt", key);
        System.out.println(flag);
    }


}
```

University of New Brunswick, Faculty of Computer Science

CS4355/6355: Topic 2 – Additional Note

1 DES and AES Problems

1. Let K be a 56-bit DES key, and let M be a 64-bit plaintext, given the ciphertext

$$C = DES(K, M) \tag{1.1}$$

how to recover the key K and the plaintext M?

Solutions.

- Case 1: If M is meaningless, e.g., password, secret key, we cannot verify whether a key is correct or not.
- Case 2: If M is meaningful,
  - For each key $k \in \{0,1\}^{56}$ do
  - $M = DES^{-1}(k, C)$
  - if M is meaningful, return $k||M$

2. Let K be a 56-bit DES key, let L be a 64-bit string, and let M be a 64-bit plaintext, check the following two algorithms derived from DES are secure or not.

$$case\ 1: \quad C = DES(K, L \oplus M) \tag{1.2}$$

$$case\ 2: \quad C = L \oplus DES(K, M) \tag{1.3}$$

For each algorithm, three pairs of plaintext-ciphertext $(M_1, C_1), (M_2, C_2), (M_3, C_3)$ are available for your cryptanalysis.

Solutions.

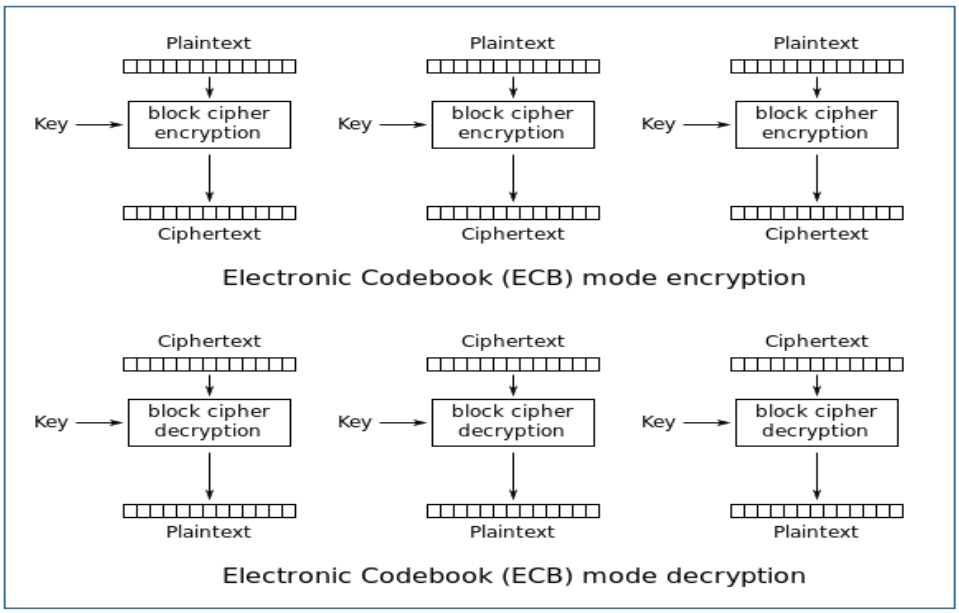
- Case 1
  - For each key $k \in \{0,1\}^{56}$ do
  - $L_1 = DES^{-1}(k, C_1) \oplus M_1$, $L_2 = DES^{-1}(k, C_2) \oplus M_2$, and $L_3 = DES^{-1}(k, C_3) \oplus M_3$
  - if $L_1 = L_2 = L_3$, return $k||L_1$
- Case 2
  - For each key $k \in \{0,1\}^{56}$ do
  - $L_1 = DES(k, M_1) \oplus C_1$, $L_2 = DES(k, M_2) \oplus C_2$, and $L_3 = DES(k, M_3) \oplus C_3$
  - if $L_1 = L_2 = L_3$, return $k||L_1$

3. Assume AES is a secure PRF (Pseudorandom Function), define a function $F(K, M) = AES(M, K)$. Is $F(K, M)$ is a secure PRF?

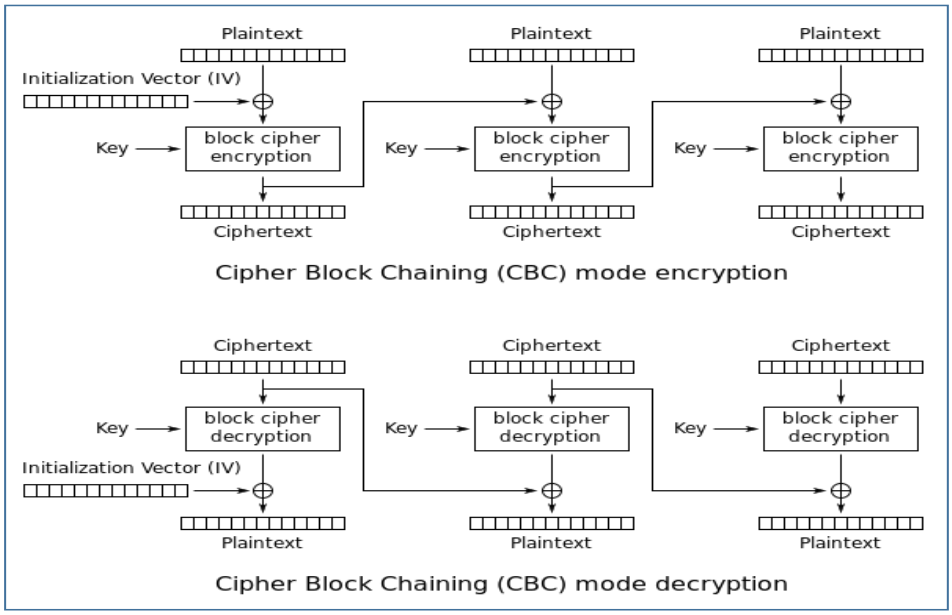
   Solution. Once we are given a pair of plaintext-ciphertext (M, C), we can easily recover the key K as $AES^{-1}(M, C) = K$. Thus, $F(K, M)$ is not a secure PRF.

2 Block Cipher Modes

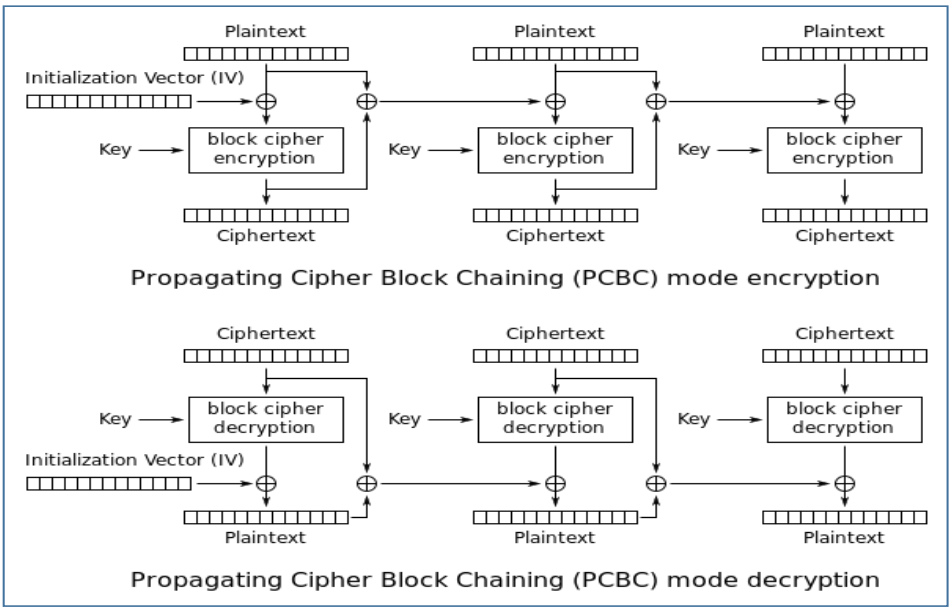
- Electronic Codebook (ECB)



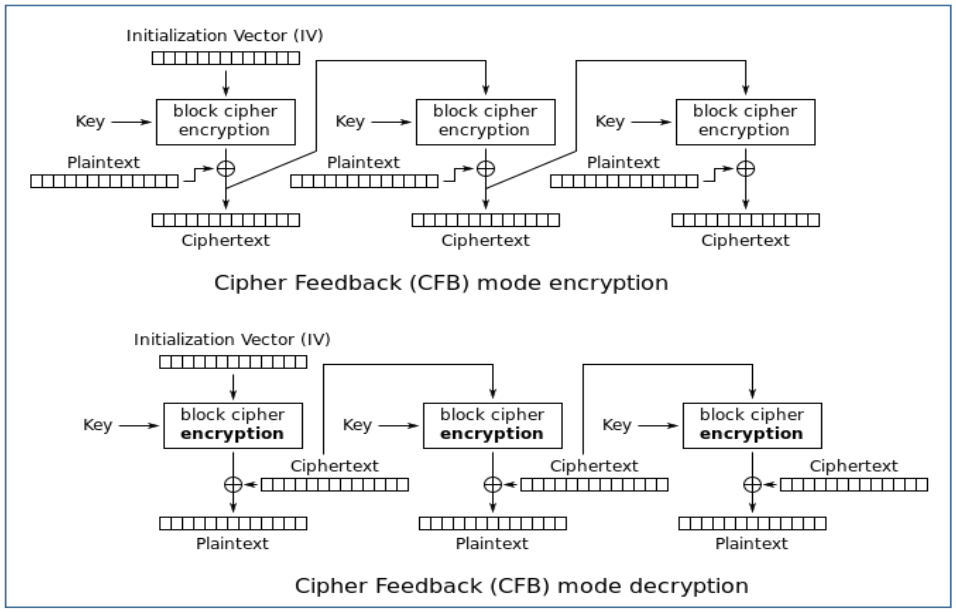
- Cipher Block Chaining (CBC)



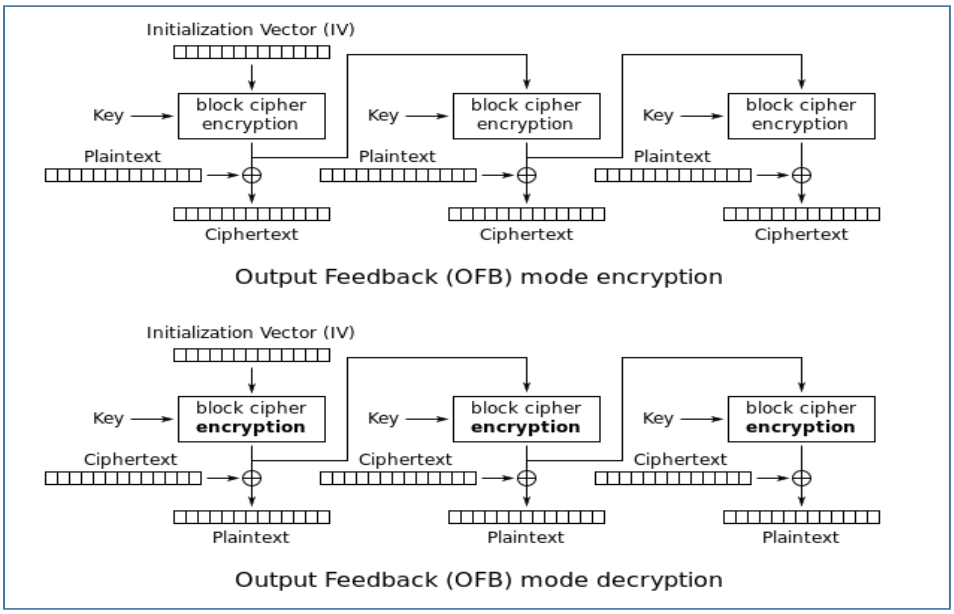
- Propagating Cipher Block Chaining (PCBC)



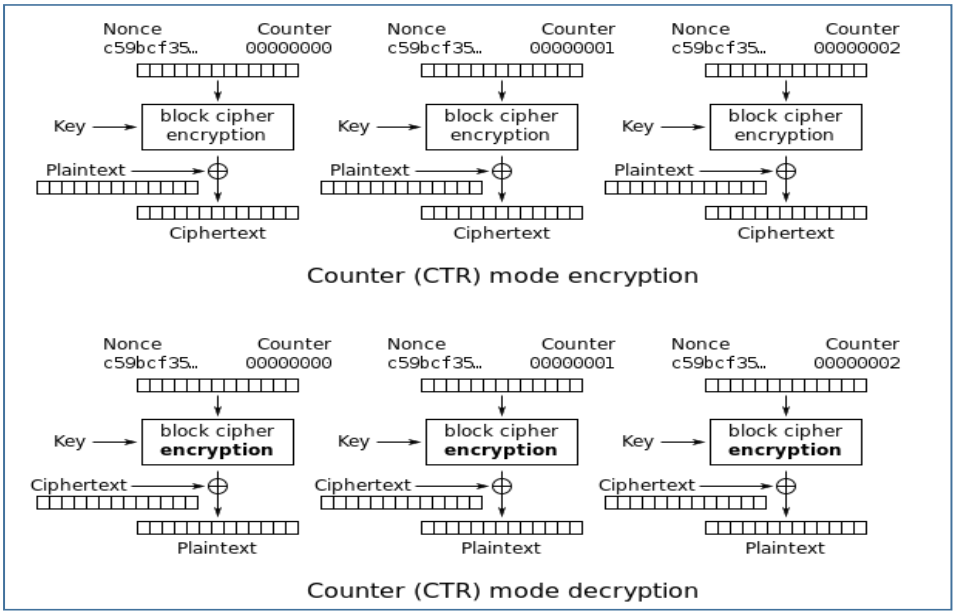
- Cipher Feedback (CFB)



- Output Feedback (OFB)


Initialization Vector (IV)
Key
block cipher encryption
Plaintext
Ciphertext
Output Feedback (OFB) mode encryption
Initialization Vector (IV)
Key
block cipher encryption
Ciphertext
Plaintext
Output Feedback (OFB) mode decryption

- Counter (CTR)


Nonce
c59bcf35...
Counter
00000000
00000001
00000002
Key
block cipher encryption
Plaintext
Ciphertext
Counter (CTR) mode encryption
Counter (CTR) mode decryption

University of New Brunswick, Faculty of Computer Science

CS4355/6355: Topic 2 – Additional Note

1 Encrypt and MAC

Using different keys for encryption and MAC schemes will be more secure.
Example: ssh (Secure Shell) protocol

```
0. k_E = Gen_E(len)
   k_M = Gen_M(len)
1. A: c = Enc(k_E; m)
      t = MAC(k_M; m)
2. A -> B: c, t
3. B: m' = Dec(k_E; c)
      t' = MAC(K_M; m')
      if t = t'
        then output m'
        else abort
```

2 Encrypt then MAC

Provably most secure in three options; do not have to decrypt if MAC fails; resist DoS
Example: IPsec (Internet Protocol Security)

```
1. A: c = Enc(k_E; m)
      t = MAC(k_M; c)
2. A -> B: c, t
3. B: t' = MAC(k_M; c)
      if t = t'
        then output Dec(k_E; c)
        else abort
```

3 MAC then Encrypt

Provably next more secure
Example: SSL (Secure Sockets Layer)

```
1. A: t = MAC(k_M; m)
      c = Enc(k_E; m,t)
2. A -> B: c
3. B: m',t' = Dec(k_E; c)
      if t' = MAC(k_M; m')
        then output m'
        else abort
```

4 AES-GCM (Galois Counter Mode)

- **Applications:** GCM mode is used in the IEEE 802.1AE (MACsec) Ethernet security, IEEE 802.11ad (also known as WiGig), ANSI (INCITS) Fibre Channel Security Protocols (FC-SP), IEEE P1619.1 tape storage, IETF IPsec standards, SSH, TLS 1.2 and TLS 1.3. AES-GCM is included in the NSA Suite B Cryptography and its latest replacement in 2018 Commercial National Security Algorithm (CNSA) suite. GCM mode is used in the SoftEther VPN server and client, as well as OpenVPN since version 2.4.
- GCM provides assurance of the confidentiality of data using a variation of the Counter mode of operation for encryption. GCM provides assurance of the authenticity of the confidential data (up to about 64 gigabytes per invocation) using a universal hash function that is defined over a binary Galois (i.e., finite) field. GCM can also provide authentication assurance for additional data (of practically unlimited length per invocation) that is not encrypted.

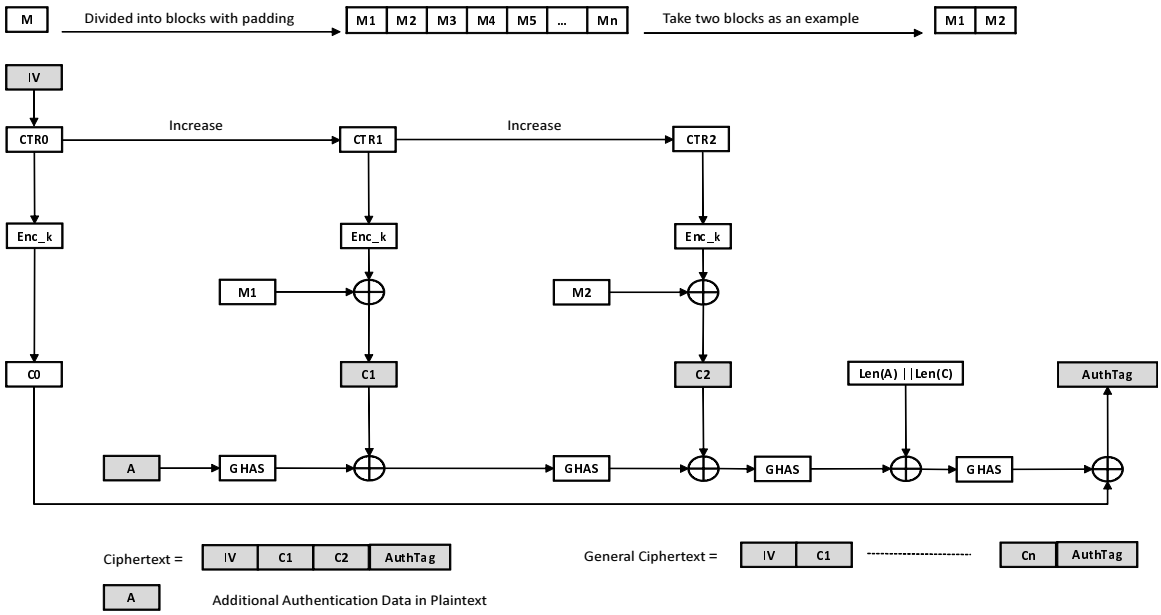


Figure 4.1: AES-GCM

- GCM provides stronger authentication assurance than a (non-cryptographic) checksum or error detecting code; in particular, GCM can detect both 1) accidental modifications of the data and 2) intentional, unauthorized modifications.
- The plaintext (denoted as M in figure) and the AAD (Additional Authentication Data denoted as A in figure) are the two categories of data that GCM protects. GCM protects the authenticity of the plaintext and the AAD; GCM also protects the confidentiality of the plaintext, while the AAD is left in the clear. For example, within a network protocol, the AAD might include addresses, ports, sequence numbers, protocol version numbers, and other fields that indicate how the plaintext should be treated. The IV is essentially a nonce, i.e, a value that is unique within the specified context, which determines an invocation of the authenticated encryption function on the input data to be protected.

University of New Brunswick, Faculty of Computer Science

CS4355/6355: Topic 2 – Additional Note

1 Group Problems

Check whether the following sets can form group under the given operation?

- Case 1: the set of real numbers R, for the operation $a \circ b = 2(a + b)$

  Answer: Cannot. Because for the given operation $a \circ b = 2(a + b)$, there is no identity. Suppose x is the identity, we have $x \circ 0 = 2(x + 0) = 2x = 0$, thus $x = 0$. However, for 1, $1 \circ 0 = 2(1 + 0) = 2 \neq 1$, which is contradictory to $x = 0$.

- Case 2: $G = \{1, -1\}$, for the ordinary multiplication operation.

  Answer: Can.

×	1	-1
1	1	-1
-1	-1	1

- Case 3: Non-Zero Real Number Set R^*, for operation $a \circ b = 2ab$.

  Answer: Can. It is easy to see Associativity is satisfied; 1/2 is the identity of R^*, for any $a \in R^*$, $\frac{1}{4a}$ is its inverse.

- Case 4: Let $G = \{(a, b) | a, b$ are real numbers and $a \neq 0\}$, for the operation $(a, b) \circ (c, d) = (ac, ad + b)$.

  Answer: Can. Check the followings:

  – G is a non-empty set

– Closure: for any (a, b), (c, d) in G, where $a \neq 0, c \neq 0$, we have $(ac, ad + b)$ are still real numbers and $ac \neq 0$, thus $(a, b) \circ (c, d) = (ac, ad + b)$ is still in G.

– Associativity: (e, f) in G, we have

$$[(a, b) \circ (c, d)] \circ (e, f) = (ac, ad + b) \circ (e, f) = (ace, acf + ad + b)$$

$$(a, b) \circ [(c, d) \circ (e, f)] = (a, b) \circ (ce, cf + d) = (ace, acf + ad + b)$$

– Existence of Identity: $(1, 0)$ in G, and $(1, 0) \circ (a, b) = (a, b)$, i.e., $(1, 0)$ is the left identity. (it is easy to see $(1, 0)$ is the right identity)

– Existence of Inverse: (a, b) in G, we have $(1/a, -b/a)$ in G and $(1/a, -b/a) \circ (a, b) = (1, 0)$ (it is easy to see $(1/a, -b/a)$ is the right identity)

– Therefore, it is a group. But it is not a commutative group, for example

$$(3, 6) = (1, 2) \circ (3, 4) \neq (3, 4) \circ (1, 2) = (3, 10)$$

University of New Brunswick, Faculty of Computer Science

CS4355/6355: Topic 3 – Additional Note

1 Group Problems

1. Let G be a group. Please prove G is an abelian group if and only if for any elements $a, b \in G$, the condition $(ab)^2 = a^2b^2$ is true.

   Proof. (1) If G is an abelian group, then for any elements $a, b \in G$, we have $(ab)^2 = (ab)(ab) = a^2b^2$.

   (2) For any elements $a, b \in G$, we have $(ab)^2 = a^2b^2$, that is, $abab = aabb$. Both sides left-multiplies a^{-1}, right-multiplies b^{-1}, we have

   $$a^{-1}ababb^{-1} = a^{-1}aabbb^{-1} \Rightarrow ebae = eabe \Rightarrow ba = ab$$

   As a result, it is an abelian group.

2. Let G be a group, and a, b, c are any three elements in G. Please prove the equation $xaxba = xbc$ has *one and only one* solution in G.

   Proof.

   $$xaxba = xbc \Rightarrow x^{-1}xaxba = x^{-1}xbc \Rightarrow axba = bc \Rightarrow a^{-1}axba = a^{-1}bc \Rightarrow xba = a^{-1}bc$$

   $$\Rightarrow xbaa^{-1} = a^{-1}bca^{-1} \Rightarrow xb = a^{-1}bca^{-1} \Rightarrow xbb^{-1} = a^{-1}bca^{-1}b^{-1}$$

   $$\Rightarrow x = a^{-1}bca^{-1}b^{-1}$$

   Therefore, it is easy to see $x = a^{-1}bca^{-1}b^{-1}$ is one solution for the equation $xaxba = xbc$.

Assume y is another solution of the equation

$$xaxba = xbc \tag{1.1}$$

i.e., we have

$$yayba = ybc \tag{1.2}$$

From Eq.(1.1), we have

$$axbac^{-1}b^{-1} = e$$

From Eq.(1.2), we have

$$aybac^{-1}b^{-1} = e$$

As a result, $x = y$. That is, the equation $xaxba = xbc$ has *one and only one* solution in G.

3. Let G be a group, please prove the elements within each case have the same order.
   - Case 1: a and a^{-1}.

     Proof. Assume $a^n = e$, we have

     $$(a^{-1})^n = a^{-n} = (a^n)^{-1} = e^{-1} = e$$

     that is, $(a^{-1})^n = e$. On the other hand, assume $(a^{-1})^n = e$, we have

     $$a^n(a^{-1})^n = a^n a^{-n} = e.$$

     we have $a^n = e$. Therefore, $|a| = |a^{-1}|$. (Note $|a|$ denotes the order of a.)

     Proof 2. We always have $aa^{-1} = e$, we have $aaa^{-1}a^{-1} = aea^{-1} = e$. Continue it, we have

     $$a^n(a^{-1})^n = e$$

     if $a^n = e$, we have $(a^{-1})^n = e$, and vice verse.
   - Case 2: a and cac^{-1} for any $c \in G$.

     Proof. Assume $a^n = e$, we have $ca^nc^{-1} = cec^{-1} = e$. Then,

     $$\underbrace{cac^{-1}cac^{-1}\cdots cac^{-1}}_{n} = ca^nc^{-1} = e$$

     We have $(cac^{-1})^n = e$.

     On the other hand, if $(cac^{-1})^n = e$, we have

     $$e = (cac^{-1})^n = \underbrace{cac^{-1}cac^{-1}\cdots cac^{-1}}_{n} = ca^nc^{-1}$$

     Then, $c^{-1}ca^nc^{-1}c = c^{-1}ec \Rightarrow a^n = e$. Therefore, $|a| = |cac^{-1}|$.

- Case 3: ab and ba.

  Proof. Assume $(ab)^n = e$, that is,

  $$(ab)^n = \underbrace{(ab)(ab)\cdots(ab)}_{n} = e \Rightarrow a^{-1}\underbrace{(ab)(ab)\cdots(ab)}_{n}b^{-1} = a^{-1}eb^{-1}$$

  $$\Rightarrow \underbrace{(ba)(ba)\cdots(ba)}_{n-1} = a^{-1}eb^{-1} \Rightarrow \underbrace{(ba)(ba)\cdots(ba)(ba)}_{n} = a^{-1}eb^{-1}(ba) = e$$

  Therefore, $(ba)^n = e$, and vice versa. Therefore, $|ab| = |ba|$.

  Proof 2. Use the result of Case 2. Because

  $$ab = a(ba)a^{-1}$$

  from the result of Case 2, we have $|ab| = |ba|$.

- Case 4: abc, bca, cab.

  Proof. Use the result of Case 2. Because

  $$bca = a^{-1}(abc)a, \quad cab = c(abc)c^{-1}$$

  from the result of Case 2, we have $|abc| = |bca| = |cab|$.

4. Let G be a group, and an element $a \in G$ has the order n. Please prove $a^s = a^t \Leftrightarrow n|(s-t)$.

   Proof. Because $a^s = a^t$, we have

   $$a^s = a^t \Rightarrow a^s(a^{-t}) = a^t a^{-t} = e \Rightarrow a^{s-t} = e$$

   Therefore, $n|(s-t)$.

   On the other side, $n|(s-t) \Rightarrow (s-t) = n \cdot k$ for some k. Then, $a^{s-t} = a^{n\cdot k} = e$.

   $$a^{s-t} = e \Rightarrow a^{s-t}a^t = ea^t \Rightarrow a^s = a^t$$

2 Ring Problem

1. Let R be a ring with identity (denoted as 1). Prove R is also a ring with the identity under the operations $a \oplus b = a + b - 1$, $a \circ b = a + b - ab$.

   Proof.

   Under $\oplus$, R is a group. We easily check it is non-empty, closure, identity $=1$, and a's inverse is $2 - a$.

   Regarding Associativity,

   $$(a \oplus b) \oplus c = a \oplus (b \oplus c) = a + b + c - 2$$

Under $\circ$, Associativity

$$(a \circ b) \circ c = a \circ (b \circ c) = a + b + c - ab - ac - bc + abc$$

Also,

$$a \circ (b \oplus c) = (a \circ b) \oplus (a \circ c) = 2a + b + c - 1 - ab - ac$$

Similarly,

$$(a \oplus b) \circ c = (a \circ c) \oplus (b \circ c)$$

As a result, R for the operations $(\oplus, \circ)$ is a ring.

University of New Brunswick, Faculty of Computer Science

CS4355/6355: Topic 3 – Additional Note

1 Finite Field

A **field** is a non-empty set F with two binary operators which are usually denoted by $+$ and $*$, that satisfy the usual arithmetic properties:

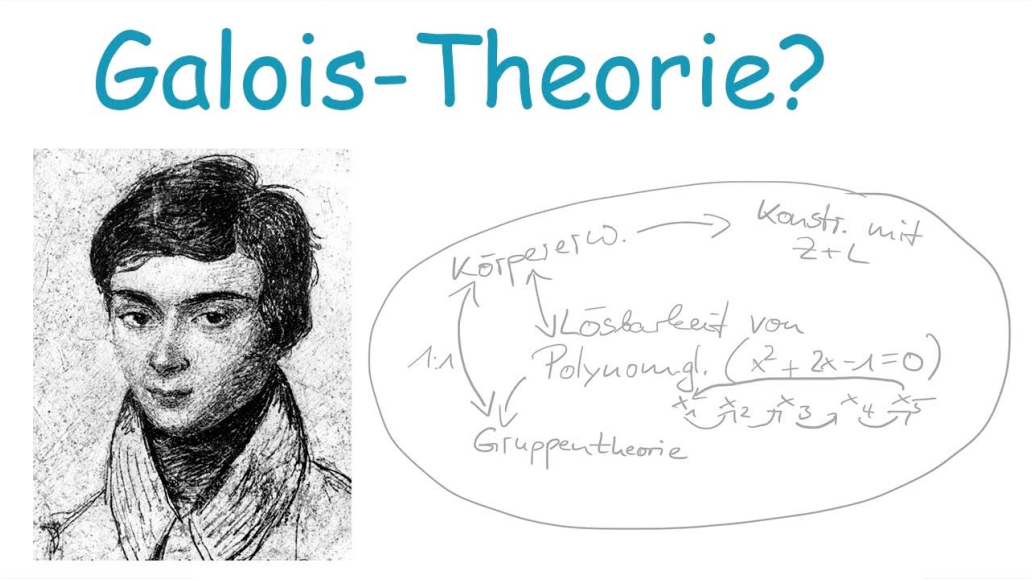
- $(F, +)$ is an Abelian group with (additive) identity denoted by 0.
- $(F/\{0\}, *)$ is an Abelian group with (multiplicative) identity denoted by 1.
- The distributive law holds: $(a + b) * c = a * c + b * c$ for all $a, b, c \in F$.

If the set F is finite, then the field is said to be a **finite field**. The **order** of a finite field is the number of elements in the finite field. By definition, $(Z, +, *)$ does not form a field because $(Z/\{0\}, *)$ is not a multiplicative group. $(Z_n, +, *)$ in general is not a finite field. For example, $Z_8/\{0\} = \{1, 2, 3, 4, 5, 6, 7\}$ along with modulo 8 multiplication does not form a group. However, when n is a prime number, things become different. For example $Z_5/\{0\} = \{1, 2, 3, 4\}$ along with modulo 5 multiplication forms the Abelian group Z_5^*. Therefore, $(Z_5, +, *)$ is a finite field.

2 Galois' Theorem and Polynomial Arithmetic

Sometimes, a finite field is also called a **Galois Field**. It is so named in honour of Évariste Galois, a French mathematician. Galois is the first one who established the following fundamental theorem on the existence of finite fields:

An order-n finite field exists if and only if $n = p^m$ for some prime p (p is called the characteristic of this finite field) and some positive integer m.



In fact, an order-n finite field is unique. All finite fields of the same order are structurally identical. We usually use $GF(p^m)$ to represent the finite field of order p^m. As we have shown above, addition and multiplication modulo a prime number p form a finite field. The order of the field is p^1. However, modulo arithmetic on its own will not let us to construct a finite field with order of p^m for $m > 1$. For example, $2^3 = 8$, and we've already know $(Z_8, +, *)$ is not a field. One way to construct a finite field with $m > 1$ is using the polynomial basis. The field is constructed as a set of p^m polynomials along with two polynomial operations.

Here a polynomial $f(x)$ is a mathematical expression in the form $a_n x^n + a_{n-1}x^{n-1} + ... + a_0$. The highest exponent of x is the **degree** of the polynomial. For example, the degree of $x^5 + 3x^3 + 4$ is 5. In a polynomial, $a_n, a_{n-1}, \cdots, a_0$ are called coefficients. If in a polynomial, the coefficients $a_n, a_{n-1}, \cdots, a_1$ are all 0, or in other words, the polynomial is in the form of a_0, we call this polynomial a constant. We can add, subtract polynomials by combine the terms in the polynomials with the same powers. For example:

- Polynomial addition: $(x^5 + 3x^3 + 4) + (6x^6 + 4x^3) = 6x^6 + x^5 + 7x^3 + 4$

$$\begin{array}{r} x^5+3x^3+4 \\ +\ 6x^6+\quad +4x^3 \\ \hline 6x^6+x^5+7x^3+4 \end{array}$$

- Polynomial subtraction: $(x^5+3x^3+4)-(6x^6+4x^3) = -6x^6+x^5-x^3+4$

$$\begin{array}{r} x^5+3x^3+4 \\ -\quad 6x^6+\quad +4x^3 \\ \hline -6x^6+x^5-1x^3+4 \end{array}$$

- Polynomial multiplication: $(x^5+3x^3+4)*(6x^6+4x^3) = 6x^{11}+18x^9+4x^8+36x^6+16x^3$

$$\begin{array}{r} x^5+3x^3+\ 4 \\ \times \qquad\qquad 6x^6+\ 4x^3 \\ \hline 4x^8+12x^6+\ 16x^3 \\ 6x^{11}+18x^9 \qquad +24x^6 \\ \hline 6x^{11}+18x^9+4x^8+36x^6+16x^3 \end{array}$$

- Polynomial division: $(6x^{11}+18x^9+4x^8+36x^6+16x^3) \div (x^5+3x^3+4) = 6x^6+4x^3$

$$\begin{array}{r} \quad 6x^6+\quad 4x^3 \\ x^5+3x^3+\ 4\ \overline{)\ 6x^{11}+18x^9+4x^8+36x^6+16x^3} \\ \quad 6x^{11}+\ 18x^9+\quad 24x^6 \\ \hline \qquad\qquad 4x^8+12x^6+16x^3 \\ \qquad\qquad 4x^8+12x^6+16x^3 \\ \hline \qquad\qquad\qquad 0 \end{array}$$

- Polynomial division with remainder: $(3x^6+7x^4+4x^3+5) \div (x^4+3x^3+4) = 3x^2-9x+34$ with remainder $-98x^3 - 12x^2 + 36x - 131$

$$
\begin{array}{r}
3x^2 - 9x + 34 \\
x^4 + 3x^3 + 4 \overline{)\,3x^6 + \quad 7x^4 + 4x^3 \quad + 5} \\
\underline{3x^6 + 9x^5 \quad + 12x^2} \\
-9x^5 + 7x^4 + 4x^3 - 12x^2 \quad + 5 \\
\underline{-9x^5 - 27x^4 \quad - 36x} \\
34x^4 + 4x^3 - 12x^2 + 36x + 5 \\
\underline{34x^4 + 102x^3 + \quad 136} \\
-98x^3 - 12x^2 + 36x - 131
\end{array}
$$

If a polynomial is divisible only by itself and constants, then we call this polynomial an **irreducible polynomial**. We will see later that irreducible polynomials have properties similar to prime numbers.

If the coefficients are taken from a field F, then we say it is a polynomial over F. With polynomials over field $GF(p)$, you can add and multiply polynomials just like you have always done but the coefficients need to be reduced modulo p. For example, compare the above results with polynomials over $GF(11)$:

- $(x^5 + 3x^3 + 4) + (6x^6 + 4x^3) = 6x^6 + x^5 + 7x^3 + 4$
- $(x^5 + 3x^3 + 4) - (6x^6 + 4x^3) = -6x^6 + x^5 - x^3 + 4 = 5x^6 + x^5 + 10x^3 + 4$
- $(x^5+3x^3+4)*(6x^6+4x^3) = 6x^{11}+18x^9+4x^8+36x^6+16x^3 = 6x^{11}+7x^9+4x^8+3x^6+5x^3$
- $(3x^6 + 7x^4 + 4x^3 + 5) \div (x^4 + 3x^3 + 4) = 3x^2 - 9x + 34 = 3x^2 + 2x + 1$ with remainder $-98x^3 - 12x^2 + 26x - 131 = x^3 + 10x^2 + 4x + 1$

Similar to integers, you can do modular arithmetic with polynomials over a field. Now the operands and modulus are polynomials. Let $f(x) = a_n x^n + a_{n-1}x^{n-1} + ... + a_0$ and $g(x) = b_m x^m + b_{m-1}x^{m-1} + ... + b_0$ be two polynomials over a field F, then there is a unique polynomial $r(x)$ of degree smaller than m and another unique polynomial $h(x)$, both over F, such that $f(x) = h(x) * g(x) + r(x)$. The polynomial $r(x)$ is called the remainder of $f(x)$ modulo $g(x)$. For polynomials $a(x)$, $b(x)$ and $g(x)$ which are over the same field, we say $a(x)$ is congruent to $b(x)$ modulo $g(x)$ written $a(x) \equiv b(x) \bmod g(x)$, if $g(x)$ divides $a(x) - b(x)$. For example (all polynomials are over $GF(3)$):

- $2x^2 \equiv 2 \bmod (x^2 - 1)$
- $x^4 \equiv 1 \bmod (x^2 - 1)$
- $x^3 \equiv x \bmod (x^2 - 1)$
- $2x^2 + x^4 \equiv 0 \bmod (x^2 - 1)$
- $2x^2 * x^4 \equiv 2 \bmod (x^2 - 1)$
- $2x^2 + x^3 \equiv x + 2 \bmod (x^2 - 1)$

Always remember there are two moduli involved: a polynomial modulus and an integer modulus. You need to reduce the result from the polynomial operations by modulo the polynomial modulus and then reduce the coefficients modulo the integer modulus. Take one of the above examples: $2x^2 + x^4 = x^4 + 2x^2$, you reduce this result by dividing by $x^2 - 1$:

$$
\begin{array}{r}
x^2+3 \\
x^2-1 \overline{)\ x^4+2x^2} \\
\underline{x^4-x^2} \\
3x^2 \\
\underline{3x^2-3} \\
3
\end{array}
$$

The remainder 3 is then reduced modulo 3: $3 \equiv 0 \bmod 3$. So the final result is $2x^2 + x^4 \equiv 0 \bmod (x^2 - 1)$.

3 Finite Fields $GF(p^m)$

If the modulus $g(x)$ is an **irreducible polynomial** of degree m over $GF(p)$, then the finite field $GF(p^m)$ can be constructed by the set of polynomials over $GF(p)$ whose degree is at most $m - 1$, where addition and multiplication are done modulo $g(x)$.

For example, the finite field $GF(3^2)$ can be constructed as the set of polynomials whose degrees are at most 1, with addition and multiplication done modulo the irreducible polynomial $x^2 + 1$ (you can also choose another modulus, as long as it is irreducible and has degree 2).

- Addition modulo $x^2 + 1$

+	**0**	**1**	**2**	**x**	**x+1**	**x+2**	**2x**	**2x+1**	**2x+2**
0	0	1	2	x	x+1	x+2	2x	2x+1	2x+2
1	1	2	0	x+1	x+2	x	2x+1	2x+2	2x
2	2	0	1	x+2	x	x+1	2x+2	2x	2x+1
x	x	x+1	x+2	2x	2x+1	2x+2	0	1	2
x+1	x+1	x+2	x	2x+1	2x+2	2x	1	2	0
x+2	x+2	x	x+1	2x+2	2x	2x+1	2	0	1
2x	2x	2x+1	2x+2	0	1	2	x	x+1	x+2
2x+1	2x+1	2x+2	2x	1	2	0	x	x+2	x+2
2x+2	2x+2	2x	2x+1	2	0	1	x+2	x	x+1

- Multiplication modulo $x^2 + 1$

+	**0**	**1**	**2**	**x**	**x+1**	**x+2**	**2x**	**2x+1**	**2x+2**
0	0	0	0	0	0	0	0	0	0
1	0	1	2	x	x+1	x+2	2x	2x+1	2x+2
2	0	2	1	2x	2x+2	2x+1	x	x+2	x+1
x	0	x	2x	2	x+2	2x+2	1	x+1	2x+1
x+1	0	x+1	2x+2	x+2	2x	1	2x+1	2	x
x+2	0	x+2	2x+1	2x+2	1	x	x+1	2x	2
2x	0	2x	x	1	2x+1	x+1	2	2x+2	x+2
2x+1	0	2x+1	x+2	x+1	2	2x	2x+2	x	1
2x+2	0	2x+2	x+1	2x+1	x	2	x+2	1	2x

4 Finite Fields $GF(2^m)$

Finite fields of order 2^m are called binary fields or characteristic-two finite fields. They are of special interest because they are particularly efficient for implementation in hardware, or on a binary computer.

The elements of $GF(2^m)$ are binary polynomials, i.e. polynomials whose coefficients are either 0 or 1. There are 2^m such polynomials in the field and the degree of each polynomial

is no more than $m-1$. Therefore the elements can be represented as m-bit strings. Each bit in the bit string corresponding to the coefficient in the polynomial at the same position. For example, $GF(2^3)$ contains 8 elements $\{0, 1, x, x+1, x^2, x^2+1, x^2+x, x^2+x+1\}$. $x+1$ is actually $0x^2+1x+1$, so it can be represented as a bit string 011. Similarly, $x^2+x = 1x^2+1x+0$, so it can be represented as 110.

In modulo 2 arithmetics, $1+1 \equiv 0 \bmod 2$, $1+0 \equiv 1 \bmod 2$ and $0+0 \equiv 0 \bmod 2$, which coincide with bit-XOR, i.e. $1 \oplus 1 = 0$, $1 \oplus 0 = 1$, $0 \oplus 0 = 0$. Therefore for binary polynomials, addition is simply bit-by-bit XOR. Also, in modulo 2 arithmetics, $-1 \equiv 1 \bmod 2$, so the result of subtraction of elements is the same as addition. For example:

- $(x^2+x+1)+(x+1) = x^2+2x+2$, since $2 \equiv 0 \bmod 2$, the final result is x^2. It can also be computed as $111 \oplus 011 = 100$. 100 is the bit string representation of x^2.
- $(x^2+x+1)-(x+1) = x^2$

Multiplication of binary polynomials can be implemented as simple bit-shift and XOR. For example:

- $(x^2+x+1)*(x^2+1) = x^4+x^3+2x^2+x+1$. The final result is x^4+x^3+x+1 after reduction modulo 2. It can also be computed as $111 * 101 = 11100 \oplus 111 = 11011$, which is exactly the bit string representation of x^4+x^3+x+1.

$$
\begin{array}{r}
111 \\
\times\ 101 \\
\hline
111 \\
0000 \\
11100 \\
\hline
11011
\end{array}
$$

In $GF(2^m)$, when the degree of the result is more than $m-1$, it needs to be reduced modulo a irreducible polynomial. This can be implemented as bit-shift and XOR. For example, x^3+x+1 is an irreducible polynomial and $x^4+x^3+x+1 \equiv x^2+x \bmod (x^3+x+1)$. The bit-string representation of x^4+x^3+x+1 is 11011, and the bit-string representation of x^3+x+1 is 1011. The degree of 11011 is 4 and the degree of the irreducible polynomial is 3, so the reduction starts by shifting the irreducible polynomial 1011 one bit left, you get 10110, then $11011 \oplus 10110 = 1101$. The degree of 1101 is 3 which is still greater than

$m-1=2$, so you need another XOR. But you don't need to shift the irreducible polynomial this time. $1101 \oplus 1011 = 0110$, which is the bit-string representation of x^2+x.

- $GF(2^3)$ – Addition – Polynomial Arithmetic Modulo x^3+x+1

	+	000	001	010	011	100	101	110	111
		0	1	2	3	4	5	6	7
000	0	0	1	2	3	4	5	6	7
001	1	1	0	3	2	5	4	7	6
010	2	2	3	0	1	6	7	4	5
011	3	3	2	1	0	7	6	5	4
100	4	4	5	6	7	0	1	2	3
101	5	5	4	7	6	1	0	3	2
110	6	6	7	4	5	2	3	0	1
111	7	7	6	5	4	3	2	1	0

- $GF(2^3)$ – Multiplication – Polynomial Arithmetic Modulo x^3+x+1

	x	000	001	010	011	100	101	110	111
		0	1	2	3	4	5	6	7
000	0	0	0	0	0	0	0	0	0
001	1	0	1	2	3	4	5	6	7
010	2	0	2	4	6	3	1	7	5
011	3	0	3	6	5	7	4	1	2
100	4	0	4	3	7	6	2	5	1
101	5	0	5	1	4	2	7	3	6
110	6	0	6	7	1	5	3	2	4
111	7	0	7	5	2	1	6	4	3

- $GF(2^3)$ – Addition, Multiplication – Polynomial Arithmetic Modulo x^3+x+1

	+	000 0	001 1	010 x	011 $x+1$	100 x^2	101 x^2+1	110 x^2+x	111 x^2+x+1
000	0	0	1	x	$x+1$	x^2	x^2+1	x^2+x	x^2+x+1
001	1	1	0	$x+1$	x	x^2+1	x^2	x^2+x+1	x^2+x
010	x	x	$x+1$	0	1	x^2+x	x^2+x+1	x^2	x^2+1
011	$x+1$	$x+1$	x	1	0	x^2+x+1	x^2+x	x^2+1	x^2
100	x^2	x^2	x^2+1	x^2+x	x^2+x+1	0	1	x	$x+1$
101	x^2+1	x^2+1	x^2	x^2+x+1	x^2+x	1	0	$x+1$	x
110	x^2+x	x^2+x	x^2+x+1	x^2	x^2+1	x	$x+1$	0	1
111	x^2+x+1	x^2+x+1	x^2+x	x^2+1	x^2	$x+1$	x	1	0

	×	000 0	001 1	010 x	011 $x+1$	100 x^2	101 x^2+1	110 x^2+x	111 x^2+x+1
000	0	0	0	0	0	0	0	0	0
001	1	0	1	x	$x+1$	x^2	x^2+1	x^2+x	x^2+x+1
010	x	0	x	x^2	x^2+x	$x+1$	1	x^2+x+1	x^2+1
011	$x+1$	0	$x+1$	x^2+x	x^2+1	x^2+x+1	x^2	1	x
100	x^2	0	x^2	$x+1$	x^2+x+1	x^2+x	x	x^2+1	1
101	x^2+1	0	x^2+1	1	x^2	x	x^2+x+1	$x+1$	x^2+x
110	x^2+x	0	x^2+x	x^2+x+1	1	x^2+1	$x+1$	x	x^2
111	x^2+x+1	0	x^2+x+1	x^2+1	x	1	x^2+x	x^2	$x+1$

University of New Brunswick, Faculty of Computer Science

CS4355/6355: Topic 3 – Additional Note

1 Number Theory Problems

1. Let n be an integer than 1. Prove that 2^n is the sum of two odd consecutive integers.

   Proof. For the problem, the relation $2^n = (2k-1) + (2k+1)$ implies $k = 2^{n-2}$ and we obtain $2^n = (2^{n-1} - 1) + (2^{n-1} + 1)$.

   □

2. Let n be an integer than 1. Prove that 3^n is the sum of three consecutive integers.

   Proof. For this problem, the relation $3^n = (s-1) + s + (s+1)$ implies $s = 3^{n-1}$ and we obtain the representation $3^n = (3^{n-1} - 1) + 3^{n-1} + (3^{n-1} + 1)$.

   □

3. Prove that if x, y, z are integers such that $x^2 + y^2 = z^2$, then $xyz \equiv 0 \bmod 30$.

   Proof.

   - First, all three of x, y, z cannot be odd, since odd + odd = even. So xyz is even, i.e., $2|(xyz)$.
   - Second, $1^2 \equiv 2^2 \equiv 1 \bmod 3$, all perfect squares are 0 or 1 mod3. However, $x^2 + y^2 \equiv z^2 \bmod 3$ is not solved by making each of x^2, y^2, z^2 be 1 mod 3. Thus, one is 0 mod 3, and so xyz is divisible by 3, i.e., $3|(xyz)$

- Third, we have $1^2 \equiv 4^2 \equiv 1 \bmod 5$, and $2^2 \equiv 3^2 \equiv -1 \bmod 5$. So $x^2 + y^2 = z^2 \bmod 5$ can look like:

$$left\ side = \begin{cases} case1: & 1+1 = 2 \bmod 5 \\ case2: & 1+(-1) = 0 \bmod 5 \\ case3: & (-1)+1 = 0 \bmod 5 \\ case4: & (-1)+(-1) = -2 = 3 \bmod 5 \end{cases}$$

$$right\ side = \begin{cases} case1: & 1 \bmod 5 \\ case2: & -1 \bmod 5 \end{cases}$$

If none of x, y, z is 0 mod 5, the left side is NOT equal to the right side. Therefore, one of x, y, z is 0 mod 5, and xyz is divisible by 5, i.e., $5|(xyz)$.

Finally, because $2|(xyz)$, $3|(xyz)$, and $5|(xyz)$, we have $2 \cdot 3 \cdot 5|(xyz)$, i.e., $xyz \equiv 0 \bmod 30$.

□

University of New Brunswick, Faculty of Computer Science

CS4355/6355: Topic 3 – Additional Note

1 Number Theory Problems

1. If $p|10a - b$, $p|10c - d$, then $p|ad - bc$.

   Proof. From $p|10a - b$, we know $p \cdot k_1 = 10a - b$ for some k_1. Then,

   $$p \cdot k_1 \cdot c = (10a - b) \cdot c$$

   Let $k_2 = k_1 \cdot c$, we have $p \cdot k_2 = (10a - b) \cdot c = 10ac - bc$.
   Similarly, we have $p \cdot k_4 = (10c - d) \cdot a = 10ca - ad$ for some k_4. Then,

   $$p \cdot k_2 - p \cdot k_4 = 10ac - bc - 10ac + ad \Rightarrow p(k_2 - k_4) = ad - bc$$

   Therefore,

   $$p|ad - bc$$

   □

2. If n is odd, then $3|2^n + 1$

   Proof. Since $2 + 1 \equiv 0 \bmod 3$, we have $2 \equiv -1 \bmod 3$. Then,

   $$2^n \equiv (-1)^n \bmod 3$$

   Because n is odd, we have

   $$2^n - (-1)^n \equiv 0 \bmod 3 \Rightarrow 2^n + 1 \equiv 0 \bmod 3 \Rightarrow 3|2^n + 1$$

   □

3. $k = 0, 1, 2, \cdots$, for $n \in \mathbb{Z}$, we have $2n+1 | 1^{2k+1} + 2^{2k+1} + \cdots + (2n-1)^{2k+1} + 2n^{2k+1}$.

   Proof. For each $i = 1, 2, \cdots, n$, we have

   $$i + (2n+1) - i \equiv 2n + 1 \equiv 0 \bmod 2n+1$$

   $$i \equiv -((2n+1) - i) \bmod 2n+1 \Rightarrow i^{2k+1} \equiv [-((2n+1) - i)]^{2k+1} \bmod 2n+1$$

   Since $2k+1$ is odd, we have

   $$i^{2k+1} + ((2n+1) - i)^{2k+1} \equiv 0 \bmod 2n+1$$

   $$\sum_{i=1}^{n} [i^{2k+1} + ((2n+1) - i)^{2k+1}] \equiv 0 \bmod 2n+1$$

   Therefore,

   $$2n+1 | 1^{2k+1} + 2^{2k+1} + \cdots + (2n-1)^{2k+1} + 2n^{2k+1}$$

   □

4. If $m - p | mn + pq$, then $m - p | mq + np$

   Proof. Since

   $$(m-p) | (m-p)(n-q) \Rightarrow (m-p) | mn + pq - (mq + np)$$

   Because $m - p | mn + pq$, we have $m - p | mq + np$. □

5. If $x \equiv 1 \bmod m^k$, then $x^m \equiv 1 \bmod m^{k+1}$.

   Proof. Since $x \equiv 1 \bmod m^k$, we have $x = 1 + k \cdot m^k = 1 + (k \cdot m^{k-1}) \cdot m$, thus

   $$m^k | x - 1, \qquad x \equiv 1 \bmod m$$

   From $x \equiv 1 \bmod m$, we have $x^i \equiv 1^i \bmod m$ for $i = 0, 1, \cdots m-1$. Then,

   $$\sum_{i=0}^{m-1} x^i \equiv \sum_{i=0}^{m-1} 1^i \bmod m$$

   $$1 + x + x^2 + \cdots + x^{m-1} \equiv m \bmod m \Rightarrow 1 + x + x^2 + \cdots + x^{m-1} \equiv 0 \bmod m$$

   Then,

   $$m | (1 + x + x^2 + \cdots + x^{m-1})$$

   Finally, we have

   $$m^k \cdot m | (x-1)(1 + x + x^2 + \cdots + x^{m-1}) \Rightarrow m^{k+1} | x^m - 1 \Rightarrow x^m \equiv 1 \bmod m^{k+1}.$$

   □

University of New Brunswick, Faculty of Computer Science

CS4355/6355: Topic 3 – Additional Note

1 Number Theory Problems

1. Let n be a positive integer. Prove that $3^{2^n}+1$ is divisible by 2, but not by 4.

   Proof. **Method 1.** Clearly, 3^{2^n} is odd and $3^{2^n}+1$ is even. Note that $3^{2^n} = (3^2)^{2^{n-1}} = 9^{2^{n-1}} = (8+1)^{2^{n-1}}$. Recall the **Binomial theorem**

   $$(x+y)^m = x^m + \binom{m}{1}x^{m-1}y + \binom{m}{2}x^{m-2}y^2 + \cdots + \binom{m}{m-1}xy^{m-1} + y^m$$

   Setting $x = 8, y = 1$, and $m = 2^{n-1}$ in the above equation, we see that each summand besides the last (that is, $y^m = 1$) is a multiple of 8 (which is a multiple of 4). Hence the remainder of 3^{2^n} on dividing by 4 is equal to 1, and the remainder of $3^{2^n}+1$ on dividing by 4 is equal to 2. □

   Proof. **Method 2.** We have $3+1 \equiv 0 \bmod 4$, that is, $3 \equiv -1 \bmod 4$. Then, $3^{2^n} \equiv (-1)^{2^n} \bmod 4$, we have $3^{2^n} \equiv 1 \bmod 4$. As a result, $3^{2^n}+1 \equiv 2 \bmod 4$, the proof is completed.

   □

2. Let p be a prime number. Then $x^2 \equiv 1 \bmod p$ if and only if $x \equiv \pm 1 \bmod p$.

   Proof. If $x \equiv \pm 1 \bmod p$, we have $x^2 \equiv 1 \bmod p$. Conversely, if $x^2 \equiv 1 \bmod p$, then p divides $x^2 - 1 = (x-1)(x+1)$, and so p must divide $x-1$ or $x+1$. □

3. If p is prime, then $(p-1)! \equiv -1 \bmod p$.

Proof. If $p = 2$, $(p-1)! \equiv -1 \bmod p$ is true, since $1! \equiv -1 \bmod 2$.

If $p = 3$, $(p-1)! \equiv -1 \bmod p$ is also true, since $2! \equiv -1 \bmod 3$.

If p is prime ≥ 5, we know $(Z_p^*, *)$ is a group, where $Z_p^* = \{1, 2, 3, \cdots, p-1\}$ has total $p-1$ elements. Based on the Group theory, each element $a \in Z_p^*$ has its inverse $a^{-1} \in Z_p^*$ such that $a * a^{-1} \equiv 1 \bmod p$. Based on the Question 2 above, we know $a = a^{-1}$ if and only $a = 1$ or $a = p-1$. Therefore, we can partition the $p-3$ numbers in the set $\{2, 3, \cdots, p-2\}$ into $(p-3)/2$ pairs of integers $\{a_i, a_i^{-1}\}$ such that $a_i * {a_i}^{-1} \equiv 1 \bmod p$ for $i = 1, 2, \cdots, (p-3)/2$. Then,

$$(p-1)! \equiv 1 \cdot 2 \cdot 3 \cdots (p-2)(p-1) \equiv (p-1) \prod_{i=1}^{(p-3)/2} a_i a_i^{-1} \equiv p-1 \equiv -1 \bmod p.$$

□

4. Let $p \geq 7$ be a prime. Prove that the number

$$\underbrace{11\cdots1}_{p-1 \quad 1's}$$

is divisible by p.

Proof. We have

$$\underbrace{11\cdots1}_{p-1 \quad 1's} = \frac{10^{p-1}-1}{9}$$

and the conclusion follows from Fermat's Little Theorem. (Note also that $\gcd(10, p) = 1$.) □

University of New Brunswick, Faculty of Computer Science

CS4355/6355: Topic 4 – Additional Note

1 The cycling attack

The cycling attack was one of the first attacks on RSA [1]. As the name of this attack suggests, the way this attack works is by repeatedly encrypting the ciphertext. When an attacker gets $c \equiv m^e \bmod n$, he will encrypt the ciphertext with the public key and this will lead him to, eventually getting an encryption which will be the original ciphertext. That is, after l encryptions, he will have

$$c^{e^l} \equiv c \equiv m^e \bmod n$$

so he will know that the previous encryption is the original plaintext, that is,

$$c^{e^{l-1}} \equiv m \bmod n$$

The value l is called the recovery exponent for the plaintext m. Suppose a plaintext m is encrypted with the public key (e, n), the recovery exponent of m divides $\phi(\phi(n))$. Because $e \in Z^*_{\phi(n)}$, we have $e^{\phi(\phi(n))} \equiv 1 \bmod n$. If $ord(e) = l$, we have $l|\phi(\phi(n))$. We need to choose e with a larger l.

2 Pollard's ρ algorithm

Pollard's ρ algorithm, described by Pollard in 1975 [2], is to find a small factor p of a given integer N. The simplified version of this algorithm is described as follows.
Algorithm: Pollard's ρ Algorithm: Given a composite $N = pq$:

1. set $a = 2, b = 2$.
2. Define the modular polynomial $f(x) = (x^2 + c) \bmod N$, with $c \neq 0, -2$
3. For $i = 1, 2, \cdots$ do:
   a) Compute $a = f(a), b = f(f(b))$.
   b) Compute $d = (a - b, N)$.
   c) If $1 < d < N$, then return d with success.
   d) If $d = N$, then terminate the algorithm with failure.

The function f is used to create two pseudo random sequences on $\mathbb{Z}_N$. The reason for this is that, picking randomly two numbers $x, y \in \mathbb{Z}_N$, there is a probability of 0.5 that after $1.777\sqrt{p}$ tries, one will be congruent modulo p. If they are $a \neq b$, then $(a - b, N)$ yields a factor of N [3]. Concretely,

$$a, b \in \mathbb{Z}_N \to a' = a \bmod p, b' = b \bmod p \to a', b' \in \mathbb{Z}_p^*$$

After $1.777\sqrt{p}$ tries, we may have $a' = b' \bmod p$, which also shows $a = b \bmod p$. Then, we have $p|(a - b)$, and $\gcd(a - b, N) = p$.
The runtime of the algorithm is $O(\sqrt{p})$, where p is N's smallest prime factor [4]. This means that against an RSA modulus N with balanced primes the runtime of the algorithm is $O(N^{1/4})$, making its an inefficient method.

Example.
Let $N = 8051$ and $f(x) = (x^2 + 1) \bmod 8051$, then, from the initial values $a = 2, b = 2$, we have

- when $i = 1$, $a = 5$, $b = 26$, then $\gcd(|x - y|, 8051) = 1$
- when $i = 2$, $a = 26$, $b = 7474$, then $\gcd(|x - y|, 8051) = 1$
- when $i = 3$, $a = 677$, $b = 871$, then $\gcd(|x - y|, 8051) = 97$
- when $i = 4$, $a = 7474$, $b = 1481$, then $\gcd(|x - y|, 8051) = 1$

3 Pollard's $p - 1$ algorithm

Let $n = pq$, where p, q are large primes. If $q|(p - 1)$, where q is also a large prime, then p is a strong prime. Otherwise, p is strong, and n can be factored by Pollard's $p - 1$ algorithm. For example, $p - 1 = 2p_1p_2p_3 \cdot p_k$ only includes small prime factors, where $p_0 = 2$. If all p_i, $i = 1, 2, \cdots, k$, $p_i < B$, where B is an integer, we will know that

$$p_0p_1p_2p_3 \cdot p_k|B! \Rightarrow (p - 1)|B! \Rightarrow B! = (p - 1) \cdot \alpha$$

Algorithm: Pollard's ρ Algorithm: Given a composite $n = pq$:

1. set $a = 2$.

2. For $i = 1, 2, \cdots, B$ do:

   a) Compute $a \equiv a^i \bmod n$.

   $d = gcd(a - 1, n)$;

   if $(1 < d < n)$

   * return $p = d$;

   else

   * return failure.

After running the algorithm, we know $a \equiv 2^{B!} \bmod n$. That is, $a = 2^{B!} + kn = 2^{B!} + kpq = 2^{B!} + k'p$, where $k' = kq$. Then,

$$a \equiv 2^{B!} \bmod p$$

Based on the Fermat's Little Theorem, we know

$$2^{p-1} \equiv 1 \bmod p \Rightarrow (2^{p-1})^{\alpha} \equiv 1^{\alpha} \bmod p \Rightarrow 2^{B!} \equiv 1 \bmod p \Rightarrow a \equiv 1 \bmod p$$

Then,

$$p|(a-1) \Rightarrow a - 1 = p \cdot k'' \Rightarrow \gcd(a - 1, n) = p$$

Example. Suppose $n = 15770708441$. If we set $B = 180$, then from the above algorithm, we can find that $a = 1162221425$ and d is computed to be 135979. In fact, the complete factorization of n into primes is

$$15770708441 = 135979 \times 115979$$

In this example, the factorization is successful because $135979 - 1 = 135978$ has only "small" prime factors:

$$135978 = 2 \times 3 \times 131 \times 173$$

Therefore, by taking $B \geq 173$, it will be the case that $135978|B!$, as desired.

References

[1] G. J. Simmons and M. J. Norris, Preliminary Comments on the MIT Public-key Cryptosystem, Cryptologia (1977).

[2] J. M. Pollard, A Monte Carlo Method for Factorization, BIT Numerical Mathematics (1975).

[3] H. Riesel, Prime Numbers and Computer Methods for Factorization, Birkhauser, 1994.

[4] Abdullah Darwish, Imad Khaled Salah, and Saleh Oqeili, Mathematical At- tacks on RSA Cryptosystem, Journal of Computer Science (2006).

University of New Brunswick, Faculty of Computer Science

CS4355/6355: Topic 5 – Additional Note

1 Hastad's Broadcast Attack on RSA

After you know the CRT (Chinese Remainder Theorem), you can understand the Hastad's Broadcast Attack on RSA.

Hastad's Broadcast Attack is also called as Common Plaintext Attack due to the fact that it needs the same plaintext be encrypted more than once. In the original attack, we need k messages, $k \geq e$, where e is a common public exponent used to encoded the k messages.

Theorem 1. Suppose a plaintext m is encrypted k times with the public keys $< e, N_1 >$, $< e, N_2 >, \cdots, < e, N_k >$, where $k \geq e$ and the $N_1, N_2, \cdots, N_k$ are pairwise co-prime. Let $N_0 = \min\{N_1, N_2, \cdots, N_k\}$ and $N = \prod_{i=1}^{k} N_i$. If the plaintext m satisfies $m < N_0$, then, for each $i = 1, 2, \cdots, k$, we have $c_i \cong m^e \bmod N_i$, and we can compute the plaintext m in time polynomial in $\log(N)$.

Proof. Given that the (N_i's are co-prime, we can apply the CRT (Chinese Remainder Theorem) to compute $C \cong m^e \bmod N$). As $m < N_0$, we have that $m^e < N_1 N_2 \cdots N_k = N$ and so $C = m^e$. Therefore, all we need to do is to compute the e-th root of C over the integers to find out m. All computations can be done in time polynomial in $\log(N)$.

2 Primitive Root

Prove if n has a primitive root, then it has exactly $\phi(\phi(n))$ of them.

Let $a \in Z_n^*$ be a primitive root. We consider the following two parts:
First, if a^k is primitive, then for any m we have some t such that

$$a^m = (a^k)^t = a^{kt}$$

Thus,

$$tk \equiv m \bmod \phi(n).$$

If $\gcd(k, \phi(n)) = c$, we have

$$m = t(k/c)c + x(\phi(n)/c)c.$$

Therefore, $c|m$. If we let $m = 1$, we get that $c = 1$. Then,

$$1 = tk + x\phi(n)$$

Thus, k and $\phi(n)$ are co-prime.
Second, on the other hand, if k and $\phi(n)$ are co-prime, we have some t such that

$$tk \equiv 1 \bmod \phi(n)$$

so for any m, we have

$$mtk \equiv m \bmod \phi(n)$$

Thus,

$$(a^k)^{mt} = a^m$$

Here, a^k is a primitive root.
For the set $S = \{k | \gcd(k, \phi(n)) = 1, k < \phi(n)\}$, based on the Euler Totient Function, we know the size of S is $\phi(\phi(n))$.
If $n = p$ a prime, then the size $\phi(p - 1)$.

University of New Brunswick, Faculty of Computer Science

CS4355/6355: Topic 5 – Additional Note

1 Discrete Logarithm Problem

How Hard Is the Discrete Logarithm Problem?

Shanks's Babystep-Giantstep Algorithm. Let G be a group and let $g \in G$ be an element of order $N \geq 2$. The following algorithm solves the discrete logarithm problem $g^x = h$ in $O(\sqrt{N} \cdot \log N)$ steps using $O(\sqrt{N})$ storage.

1. Let $n = 1 + \lfloor \sqrt{N} \rfloor$; in particular $n > \sqrt{N}$.
2. Create two lists,
$$List - 1: \quad e, g, g^2, g^3, \cdots, g^n$$
$$List - 2: \quad h, h \cdot g^{-n}, h \cdot g^{-2n}, h \cdot g^{-3n}, \cdots, h \cdot g^{-n^2}$$
3. Find a match between the two lists, say $g^i = h \cdot g^{-jn}$.
4. Then, $x = i + jn$ is a solution to $g^x = h$.

Proof. We begin with a couple of observations. First, when creating List 2, we start by computing the quantity $u = g^n$ and then compile List 2 by computing $h, h \cdot u, h \cdot u^2, h \cdot u^3, \cdots, h \cdot u^n$. Thus, creating the two lists takes approximately $2n$ multiplications.
Second, assuming that a match exists, we can find a match in a small multiple of $n \log n$ steps using standard sorting and searching algorithms, so step (3) takes $O(n \log n)$ steps.

Hence the total running time for the algorithm is $O(n \log n) = O(\sqrt{N} \log N)$. Fort the last step, we use the fact that $n \approx \sqrt{N}$, so

$$n \log n \approx \sqrt{N} \log \sqrt{N} = \frac{1}{2} \sqrt{N} \log N.$$

Third, the lists in Step (2) have length n, so require $\sqrt{N}$ storage.
In order to prove that the algorithm works, we must show that Lists 1 and 2 always have a match. To see this, let x be the unknown solution to $g^x = h$ and write x as

$$x = n \cdot q + r \quad with \quad 0 \leq r < n$$

We know that $1 \leq x < N$, so

$$q = \frac{x - r}{n} < \frac{N}{n} < n. \quad since \quad n > \sqrt{N}$$

Hence we can rewrite the equation $g^x = h$ as

$$g^r = h \cdot g^{-qn}$$

with $0 \leq r < n$ and $0 \leq q < n$.

Thus g^r is in List 1 and $h \cdot g^{-qn}$ is in List 2, which shows that Lists 1 and 2 have a common element.

University of New Brunswick, Faculty of Computer Science

CS4355/6355: Topic 5 – Additional Note

1 Some Examples of Elliptic Curve Group

Elliptic Curve Addition Algorithm

Let

$$E : Y^2 = X^3 + AX + B$$

be an elliptic curve and let P_1 and P_2 be points on E.

1. If $P_1 = \mathcal{O}$, then $P_1 + P_2 = P_2$.
2. Otherwise, if $P_2 = \mathcal{O}$, then $P_1 + P_2 = P_1$.
3. Otherwise, write $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$.
4. If $x_1 = x_2$, and $y_1 = -y_2$, then $P_1 + P_2 = \mathcal{O}$.
5. Otherwise, define λ by

$$\lambda = \begin{cases} \dfrac{y_2 - y_1}{x_2 - x_1}, & \text{if } P_1 \neq P_2; \\ \dfrac{3x_1^2 + A}{2y_1}, & \text{if } P_1 = P_2. \end{cases}$$

and let

$$x_3 = \lambda^2 - x_1 - x_2$$
$$y_3 = \lambda(x_1 - x_3) - y_1$$

Then,

$$P_1 + P_2 = (x_3, y_3)$$

Elliptic Curve over Finite Fields

Definition. Let $p \geq 3$ be a prime, An elliptic curve over $\mathbb{F}_p$ is an equation of the form

$$E : Y^2 = X^3 + AX + B$$

with $A, B \in \mathbb{F}_p$ satisfying $4A^3 + 27B^2 \neq 0$.
The set of points on E with coordinates in $\mathbb{F}_p$ is the set

$$E(\mathbb{F}_p) = \{(x, y) : x, y \in \mathbb{F}_p \text{ satisfy } y^2 = x^3 + Ax + B\} \cup \{\mathcal{O}\}$$

Example

Elliptic Curve $E : Y^2 = X^3 + 3X + 8$ over $\mathbb{F}_{13}$

- $P = (9, 7)$, and $Q = (1, 8)$ in $E(\mathbb{F}_{13})$
  - $$\lambda = \frac{y_2 - y_1}{x_2 - x_1} = \frac{8-7}{1-9} = \frac{1}{-8} = \frac{1}{5} = 8$$
  - $$v = y_1 - \lambda x_1 = 7 - 8 \cdot 9 = -65 = 0$$
  - $$x_3 = \lambda^2 - x_1 - x_2 = 64 - 9 - 1 = 54 = 2$$
  - $$y_3 = \lambda(x_1 - x_3)y_1 = -\lambda x_3 - y_1 + \lambda x_1 = -\lambda x_3 - v = -(\lambda x_3 + v) = -8 \cdot 2 = -16 = 10.$$
- Similarly, we can use the addition algorithm to add $P = (9, 7)$ to itself.
  - $$\lambda = \frac{3x_1^2 + A}{2y_1} = \frac{3 \cdot 9^2 + 3}{2 \cdot 7} = \frac{246}{14} = 12$$
  - $$v = y_1 - \lambda x_1 = 7 - 12 \cdot 9 = 3$$
  - Then
    $$x_3 = \lambda^2 - x_1 - x_2 = (12)^2 - 9 - 9 = 9$$
    $$y_3 = -(\lambda x_3 + v) = -(12 \cdot 9 + 3) = 6,$$
    so
    $$P + P = (9, 7) + (9, 7) = (9, 6) \quad in \quad E(\mathbb{F}_{13})$$

	$\mathcal{O}$	$(1,5)$	$(1,8)$	$(2,3)$	$(2,10)$	$(9,6)$	$(9,7)$	$(12,2)$	$(12,11)$
$\mathcal{O}$	$\mathcal{O}$	$(1,5)$	$(1,8)$	$(2,3)$	$(2,10)$	$(9,6)$	$(9,7)$	$(12,2)$	$(12,11)$
$(1,5)$	$(1,5)$	$(2,10)$	$\mathcal{O}$	$(1,8)$	$(9,7)$	$(2,3)$	$(12,2)$	$(12,11)$	$(9,6)$
$(1,8)$	$(1,8)$	$\mathcal{O}$	$(2,3)$	$(9,6)$	$(1,5)$	$(12,11)$	$(2,10)$	$(9,7)$	$(12,2)$
$(2,3)$	$(2,3)$	$(1,8)$	$(9,6)$	$(12,11)$	$\mathcal{O}$	$(12,2)$	$(1,5)$	$(2,10)$	$(9,7)$
$(2,10)$	$(2,10)$	$(9,7)$	$(1,5)$	$\mathcal{O}$	$(12,2)$	$(1,8)$	$(12,11)$	$(9,6)$	$(2,3)$
$(9,6)$	$(9,6)$	$(2,3)$	$(12,11)$	$(12,2)$	$(1,8)$	$(9,7)$	$\mathcal{O}$	$(1,5)$	$(2,10)$
$(9,7)$	$(9,7)$	$(12,2)$	$(2,10)$	$(1,5)$	$(12,11)$	$\mathcal{O}$	$(9,6)$	$(2,3)$	$(1,8)$
$(12,2)$	$(12,2)$	$(12,11)$	$(9,7)$	$(2,10)$	$(9,6)$	$(1,5)$	$(2,3)$	$(1,8)$	$\mathcal{O}$
$(12,11)$	$(12,11)$	$(9,6)$	$(12,2)$	$(9,7)$	$(2,3)$	$(2,10)$	$(1,8)$	$\mathcal{O}$	$(1,5)$

Addition table for $E : Y^2 = X^3 + 3X + 8$ over $\mathbb{F}_{13}$

2 Quadratic Residue

Definition. Suppose p is an odd prime and a is an integer. a is defined to be a **quadratic residue** modulo p if $a \not\equiv 0 \bmod p$ and the congruence $y^2 \equiv a \bmod p$ has a solution $y \in Z_p$. a is defined to be a **quadratic non-residue** modulo p if $a \not\equiv 0 \bmod p$ and a is not a quadratic residue modulo p.

Example. In Z_{11}, we have that $1^2 = 1, 2^2 = 4, 3^2 = 9, 4^2 = 5, 5^2 = 3, 6^2 = 3, 7^2 = 5, 8^2 = 9, 9^2 = 4$, and $10^2 = 1$. Therefore, the quadratic residues modulo 11 are 1, 3, 4, 5, and 9, and quadratic non-residues modulo 11 are 2, 6, 7, 8, and 10.
Suppose that p is an odd prime and a is a quadratic residue modulo p. Then, there exists $y \in Z_p^*$ such that $y^2 \equiv a \bmod p$. Clearly, $(-y)^2 \equiv a \bmod p$, and $y \not\equiv -y \bmod p$.
Now consider the quadratic congruence $x^2 - a \equiv 0 \bmod p$. Because $y^2 \equiv a \bmod p$, we have $(x-y)(x+y) \equiv 0 \bmod p$, that is, $p|x-y$ or $p|x+y$. In other words, $x \equiv \pm y \bmod p$. Therefore, we can conclude that there are exactly two solutions (modulo p) to the congruence $x^2 - a \equiv 0 \bmod p$. Moreover, these two solutions are negative of each other modulo p.

3 Euler's Criterion

Euler's Criterion. Let p be an odd prime. Then a is a quadratic residue modulo p if and only if $a^{(p-1)/2} \equiv 1 \bmod p$.

Proof. First, suppose $a \equiv y^2 \bmod p$. Then, based on the Fermat's Little Theorem, we know $y^{p-1} \equiv 1 \bmod p$ for any $y \in Z_p^*$. Therefore, we have

$$a^{(p-1)/2} \equiv (y^2)^{(p-1)/2} \bmod p \equiv y^{(p-1)} \bmod p \equiv 1 \bmod p$$

Conversely, suppose $a^{(p-1)/2} \equiv 1 \bmod p$. Let b be a primitive root modulo p. Then, $a \equiv b^i \bmod p$ for some positive integer i. Then, we have

$$a^{(p-1)/2} \equiv (b^i)^{(p-1)/2} \bmod p \equiv b^{i(p-1)/2} \bmod p$$

Since b has order $p-1$, it must be the case that $p-1$ divides $i(p-1)/2$. Hence, i is even, and then the square roots of a are $\pm b^{i/2} \bmod p$.