INSTITUT TEKNOLOGI DEL
MATERI PRAKTIKUM
Keamanan Perangkat Lunak
SEMESTER GASAL TAHUN AJAR 2024/2025

Session Date	**:**	30 Oktober 2024
Semester	**:**	V
Courses	**:**	Software Security / Keamanan Perangkat Lunak
Week/Session	**:**	05/02
Key Topics	**:**	**Public Key Cryptosystem, Fermat theorem, Euler theorem RSA, and Elgamal.**
Activity	**:**	Mengerjakan *review question*, dan *problem.*
Duration	**:**	170 menit
Delivery	**:**	Lembar jawaban tulis tangan.
Deadline of delivery	**:**	e-Course.
Place of delivery	**:**	e-Course.
Goal	**:**	Mahasiswa memahami konsep dasar dari *public key cryptography*

PENUGASAN:

Sebelum bekerja, setiap mahasiswa harus membaca instruksi di bawah ini.

Sangat disarankan bagi anda untuk:

1. ***Membaca soal-soal yang diberikan secara.***
2. ***Mencari sumber-sumber lain seperti buku, artikel, bahkan video untuk memperkaya wawasan dan meningkatkan pemahaman anda.***
3. ***Jika anda merasa ada hal yang belum dipahami, silakan untuk berkonsultasi pada TA.***
4. ***Dengan demikian diharapkan anda mampu mengikuti materi kuliah dan praktikum sebaik mungkin.***
5. ***Anda diharapkan membaca buku yang diberikan, untuk topik kali ini diambil dari Part Two : Asymmetric Chipers.***

Referensi :

- Stallings, W. (2019). *Cryptography and network security: principles and practice* (6th ed.). Hoboken, NJ: Pearson Education, Inc.

Selamat Belajar & Good Luck!

Review Questions

1. What are the principal elements of public-key cryptosystem?
2. What requirements must a public-key cryptosystems fulfill to be a secure algorithm?
3. What is one-way function?
4. Why does the usage of two prime number *p* and *q* in RSA satisfy Euler's theorem?
5. Find GCD(7378, 2006)
6. Using extended Euclidean, find the multiplicative inverse of 1234 mod 4321.

Problems

1. Find the answer for the following (Euler's Totient Function):
    a. $\emptyset(\mathrm{n})$

       n : your last 2 digits NIM. If NIM<10 then n+80.

       NIM 01 -> 81

       NIM 02 -> 82

       NIM 03 -> 83

       Etc.
    b. $\emptyset(220)$
2. Using Fermat's theorem, find 7^{301} mod 16.
3. Using Euler's theorem, find a number that is congruent with 3^{2000} mod 13.
4. Perform encryption and decryption using the RSA algorithm, for the following:
    a. p = 3; q = 11, e = 7; M = 5
    b. p = 5; q = 11, e = 3; M = 9
5. If you intercept the ciphertext C = 10 using the RSA algorithm, the public key *e* is 5, and *n* = 35, what is the plaintext *m*? (search for *d* first)
6. Users A and B use the Diffie-Hellman key exchange technique with a common prime *q* = 71 and a primitive root a = 7.
    a. If user A has private key *XA* = 5, what is A's public key *YA*?
    b. If user B has private key *XB* = 12, what is B's public key *YB*?
    c. What is the shared secret key ?
7. The example used by Sun-Tsu to illustrate the CRT was

   $$x \equiv 2\ (mod\ 3);\ x \equiv 3\ (mod\ 5);\ x \equiv 2\ (mod\ 7);$$

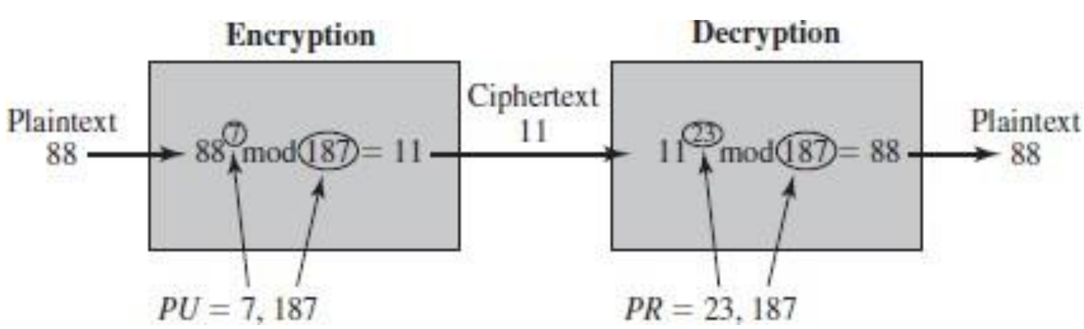
   Solve for x.
8. Given 2 as a primitive root of 29, construct a table of discrete logarithms, and use it to solve the following congruences.
    a. $x^7 \equiv 17\ (mod\ 29)$
    b. $17x^2 \equiv 10\ (mod\ 29)$
9. Consider an ElGamal scheme with a common prime q=71 and a primitive root α = 7
    a. If B has public key $Y_B = 3$ and A chose the random integer k=2, what is the ciphertext of M=30?
    b. If A now chooses a different value of so that the encoding of is C = (59, C_2), what is the integer C_2?

1. RSA Algorithm

Key Generation by Alice	
Select p, q	p and q both prime, $p \neq q$
Calculate $n = p \times q$	
Calcuate $\phi(n) = (p - 1)(q - 1)$	
Select integer e	$\gcd(\phi(n), e) = 1; 1 < e < \phi(n)$
Calculate d	$d \equiv e^{-1} \pmod{\phi(n)}$
Public key	$PU = \{e, n\}$
Private key	$PR = \{d, n\}$

Encryption by Bob with Alice's Public Key	
Plaintext:	$M < n$
Ciphertext:	$C = M^e \bmod n$

Decryption by Alice with Alice's Public Key	
Ciphertext:	C
Plaintext:	$M = C^d \bmod n$



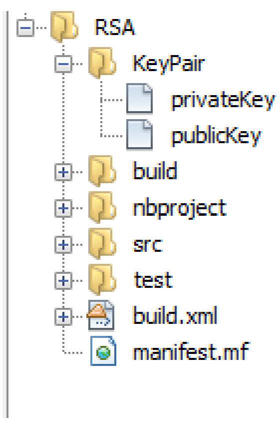
Code :

a. Generate a Public-Private Key Pair (Java)

```
package rsa;

import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.security.KeyPair;
import java.security.KeyPairGenerator;
import java.security.NoSuchAlgorithmException;
import java.security.NoSuchProviderException;
import java.security.PrivateKey;
import java.security.PublicKey;

public class GenerateKeys {

	private KeyPairGenerator keyGen;
	private KeyPair pair;
	private PrivateKey privateKey;
	private PublicKey publicKey;

	public GenerateKeys(int keylength) throws NoSuchAlgorithmException, 
NoSuchProviderException {
		this.keyGen = KeyPairGenerator.getInstance("RSA");
		this.keyGen.initialize(keylength);
	}

	public void createKeys() {
		this.pair = this.keyGen.generateKeyPair();
		this.privateKey = pair.getPrivate();
		this.publicKey = pair.getPublic();
	}

	public PrivateKey getPrivateKey() {
		return this.privateKey;
	}

	public PublicKey getPublicKey() {
		return this.publicKey;
	}

	public void writeToFile(String path, byte[] key) throws IOException {
		File f = new File(path);
		f.getParentFile().mkdirs();

		FileOutputStream fos = new FileOutputStream(f);
		fos.write(key);
		fos.flush();
```

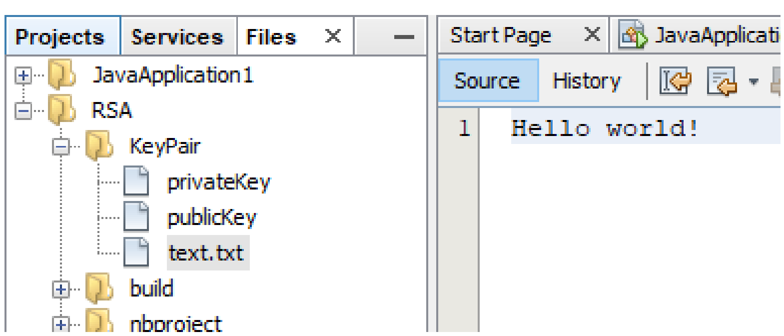
```
            fos.close();
      }

      public static void main(String[] args) {
            GenerateKeys gk;
            try {
                  gk = new GenerateKeys(1024);
                  gk.createKeys();
                  gk.writeToFile("KeyPair/publicKey",
gk.getPublicKey().getEncoded());
                  gk.writeToFile("KeyPair/privateKey",
gk.getPrivateKey().getEncoded());
            } catch (NoSuchAlgorithmException | NoSuchProviderException e)
{
                  System.err.println(e.getMessage());
            } catch (IOException e) {
                  System.err.println(e.getMessage());
            }

      }

}
```

File privateKey dan publicKey akan di-generate dalam folder KeyPair setelah code dijalankan.



b. Create text to encrypt

Tambahkan file text.txt denga nisi pesan yang akan di enkripsi.



c. Use the Key Pair to encrypt and decrypt data

```
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.UnsupportedEncodingException;
import java.nio.file.Files;
import java.security.GeneralSecurityException;
import java.security.InvalidKeyException;
import java.security.KeyFactory;
import java.security.NoSuchAlgorithmException;
import java.security.PrivateKey;
import java.security.PublicKey;
import java.security.spec.PKCS8EncodedKeySpec;
import java.security.spec.X509EncodedKeySpec;

import javax.crypto.BadPaddingException;
import javax.crypto.Cipher;
import javax.crypto.IllegalBlockSizeException;
import javax.crypto.NoSuchPaddingException;

import org.apache.commons.codec.binary.Base64;

public class AsymmetricCryptography {
	private Cipher cipher;

	public AsymmetricCryptography() throws NoSuchAlgorithmException,
NoSuchPaddingException {
		this.cipher = Cipher.getInstance("RSA");
	}

	//
https://docs.oracle.com/javase/8/docs/api/java/security/spec/PKCS8EncodedKeySpec.
html
	public PrivateKey getPrivate(String filename) throws Exception {
		byte[] keyBytes = Files.readAllBytes(new File(filename).toPath());
		PKCS8EncodedKeySpec spec = new PKCS8EncodedKeySpec(keyBytes);
		KeyFactory kf = KeyFactory.getInstance("RSA");
		return kf.generatePrivate(spec);
	}

	//
https://docs.oracle.com/javase/8/docs/api/java/security/spec/X509EncodedKeySpec.h
tml
	public PublicKey getPublic(String filename) throws Exception {
		byte[] keyBytes = Files.readAllBytes(new File(filename).toPath());
		X509EncodedKeySpec spec = new X509EncodedKeySpec(keyBytes);
		KeyFactory kf = KeyFactory.getInstance("RSA");
		return kf.generatePublic(spec);
	}

	public void encryptFile(byte[] input, File output, PrivateKey key)
		throws IOException, GeneralSecurityException {
```

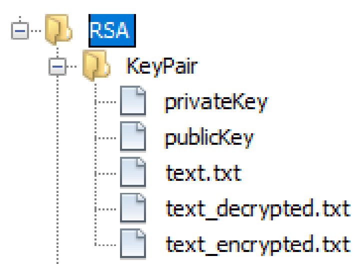
```
		this.cipher.init(Cipher.ENCRYPT_MODE, key);
		writeToFile(output, this.cipher.doFinal(input));
	}

	public void decryptFile(byte[] input, File output, PublicKey key)
		throws IOException, GeneralSecurityException {
		this.cipher.init(Cipher.DECRYPT_MODE, key);
		writeToFile(output, this.cipher.doFinal(input));
	}

	private void writeToFile(File output, byte[] toWrite)
			throws IllegalBlockSizeException, BadPaddingException,
IOException {
		FileOutputStream fos = new FileOutputStream(output);
		fos.write(toWrite);
		fos.flush();
		fos.close();
	}

	public String encryptText(String msg, PrivateKey key)
			throws NoSuchAlgorithmException, NoSuchPaddingException,
			UnsupportedEncodingException, IllegalBlockSizeException,
			BadPaddingException, InvalidKeyException {
		this.cipher.init(Cipher.ENCRYPT_MODE, key);
		return Base64.encodeBase64String(cipher.doFinal(msg.getBytes("UTF-
8")));
	}

	public String decryptText(String msg, PublicKey key)
			throws InvalidKeyException, UnsupportedEncodingException,
			IllegalBlockSizeException, BadPaddingException {
		this.cipher.init(Cipher.DECRYPT_MODE, key);
		return new String(cipher.doFinal(Base64.decodeBase64(msg)), "UTF-
8");
	}

	public byte[] getFileInBytes(File f) throws IOException {
		FileInputStream fis = new FileInputStream(f);
		byte[] fbytes = new byte[(int) f.length()];
		fis.read(fbytes);
		fis.close();
		return fbytes;
	}

	public static void main(String[] args) throws Exception {
		AsymmetricCryptography ac = new AsymmetricCryptography();
		PrivateKey privateKey = ac.getPrivate("KeyPair/privateKey");
		PublicKey publicKey = ac.getPublic("KeyPair/publicKey");

		String msg = "KEPAL is fun!";
		String encrypted_msg = ac.encryptText(msg, privateKey);
		String decrypted_msg = ac.decryptText(encrypted_msg, publicKey);
		System.out.println("Original Message: " + msg +
			"\nEncrypted Message: " + encrypted_msg
			+ "\nDecrypted Message: " + decrypted_msg);
```

```
            if (new File("KeyPair/text.txt").exists()) {
                  ac.encryptFile(ac.getFileInBytes(new 
File("KeyPair/text.txt")),
                        new File("KeyPair/text_encrypted.txt"),privateKey);
                  ac.decryptFile(ac.getFileInBytes(new 
File("KeyPair/text_encrypted.txt")),
                        new File("KeyPair/text_decrypted.txt"), publicKey);
            } else {
                  System.out.println("Create a file text.txt under folder 
KeyPair");
            }
      }
}
```

Output dari program adalah :

```
Original Message: KEPAL is fun!
Encrypted Message:
PPPt8W3HrZfyuaZQG1e64PeHsEqUdnkbwYwAoo3hfLOzkQLUhQSKgmHmoOS0ay+QRUqhzUWGTg
wEuT9306g0Z7WfdpPjxjuQ+Pd7USsBe6g1oKtUtsRTbRQwvlkLjMOltUNtp52iRJ4gP0buG01q
Zv7XH+BiAvkx31pjZ7bKI8w=
Decrypted Message: KEPAL is fun!
```

Dan ada 2 file hasil dari enkripsi file *text.txt* yang dibuat tadi.



2. Elgamal Crytosystem

Global Public Elements	
q	prime number
α	$\alpha < q$ and α a primitive root of q

Key Generation by Alice	
Select private X_A	$X_A < q - 1$
Calculate Y_A	$Y_A = \alpha^{X_A} \bmod q$
Public key	$\{q, \alpha, Y_A\}$
Private key	X_A

Encryption by Bob with Alice's Public Key	
Plaintext:	$M < q$
Select random integer k	$k < q$
Calculate K	$K = (Y_A)^k \bmod q$
Calculate C_1	$C_1 = \alpha^k \bmod q$
Calculate C_2	$C_2 = KM \bmod q$
Ciphertext:	(C_1, C_2)

Decryption by Alice with Alice's Private Key	
Ciphertext:	(C_1, C_2)
Calculate K	$K = (C_1)^{X_A} \bmod q$
Plaintext:	$M = (C_2 K^{-1}) \bmod q$

Proses dari Elgamal dari gambar diatas.

1. Bob menghasilkan integer acak *k*.

2. Bob menghasilkan sebuah one-time key *K* menggunakan komponen public key-nya Alice *YA*, *q*, dan *k*.

3. Bob mengenkripsi *k* menggunakan komponen a, menghasilkan *C*1. *C*1 menyediakan informasi yang cukup bagi Alice untuk mengembalikan *K*.

4. Bob mengenkripsi *plaintext M* menggunakan *K*.

5. Alice mengembalikan *K* dari *C*1 menggunakan *private key* miliknya.

6. Alice uses *K*-1 to recover the plaintext message from *C*2.

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package elgamal;


// referensi source code :
http://www.cs.ucf.edu/~dmarino/ucf/cis3362/progs/ElGamal.java


import java.math.*;
import java.util.*;
import java.security.*;
import java.io.*;

/**
 *
 * @author COMPUTER
 */
public class ElGamal {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {

        Scanner stdin = new Scanner(System.in);
        Random r = new Random();

        // Get user input for p.
        System.out.println("Enter the approximate value of the prime number for your
El Gamal key.");
        BigInteger p = getNextPrime(stdin.next());

        // Calculate a generator.
        BigInteger g = getGenerator(p, r);

        // We found a generator, so let's do the rest of it.
        if (g != null) {

            // Pick a secret a.
            BigInteger a = new BigInteger(p.bitCount() - 1, r);

            // Calculate the corresponding public b.
            BigInteger b = g.modPow(a, p);

            // Print out our public keys.
            System.out.println("Post p = " + p + " g = " + g + " b = " + b);
```

```
            // When we send a message, the sender picks a random k.
            BigInteger k = new BigInteger(p.bitCount() - 1, r);

            // Here, the sender starts calculating parts of the ciphertext that
            // don't involve the actual message.
            BigInteger c1 = g.modPow(k, p);
            BigInteger c2 = b.modPow(k, p);

            // Here we get the message from the user.
            System.out.println("Please enter your message. It should be in between 1
and " + p);
            BigInteger m = new BigInteger(stdin.next());

            // Now, we can calculate the rest of the second ciphertext.
            c2 = c2.multiply(m);
            c2 = c2.mod(p);

            // Print out the two ciphertexts.
            System.out.println("The corresponding cipher texts are c1 = " + c1 + "
c2 = " + c2);

            // First, determine the inverse of c1 raised to the a power mod p.
            BigInteger temp = c1.modPow(a, p);
            temp = temp.modInverse(p);

            // Print this out.
            System.out.println("Here is c1^ -a = " + temp);

            // Now, just multiply this by the second ciphertext
            BigInteger recover = temp.multiply(c2);
            recover = recover.mod(p);

            // And this will give us our original message back!
            System.out.println("The original message = " + recover);
        } // My sorry message!
        else {
            System.out.println("Sorry, a generator for your prime couldn't be
found.");
        }

    }

    // Incrementally tries each BigInteger starting at the value passed
    // in as a parameter until one of them is tests as being prime.
    public static BigInteger getNextPrime(String ans) {

        BigInteger one = new BigInteger("1");
        BigInteger test = new BigInteger(ans);
        while (!test.isProbablePrime(99)) {
            test = test.add(one);
        }
        return test;
    }
```

```
    // Precondition - p is prime and it's reasonably small, say, no more than
    //                      5,000,000. If it's larger, this method will be quite
    //                time-consuming.
    // Postcondition - if a generator for p can be found, then it is returned
    //                 if no generator is found after 1000 tries, null is
    //                 returned.
    public static BigInteger getGenerator(BigInteger p, Random r) {

        int numtries = 0;

        // Try finding a generator at random 100 times.
        while (numtries < 1000) {

            // Here's what we're trying as the generator this time.
            BigInteger rand = new BigInteger(p.bitCount() - 1, r);

            BigInteger exp = BigInteger.ONE;
            BigInteger next = rand.mod(p);

            // We exponentiate our generator until we get 1 mod p.
            while (!next.equals(BigInteger.ONE)) {
                next = (next.multiply(rand)).mod(p);
                exp = exp.add(BigInteger.ONE);
            }

            // If the first time we hit 1 is the exponent p-1, then we have
            // a generator.
            if (exp.equals(p.subtract(BigInteger.ONE))) {
                return rand;
            }
        }

        // None of the 1000 values we tried was a generator.
        return null;

    }

}
```

Deliverables:

Answers to review questions and problems, handwritten and scanned.

Create a .pdf document with the name **NIM_Tugas KEPAL_Week5.pdf**

Submission deadline:

Submit to Ecourse.del.ac.id

Sabtu, 30 September 2023. Pukul 17.00 WIB.