DAFTAR PUSTAKA

Aumasson, J.-P. (2018). *Serious Cryptography A Practical Introduction to Modern Encryption.* San Francisco: No Starch Press.

Badan Standar Nasional Pendidikan (BSNP) Dan Badan Penelitian Dan Pengembangan, Kementerian Pendidikan Dan Kebudayaan (Balitbang Kemdikbud). (2018). *Buku Saku Ujian Nasional 2019.* Jakarta: Badan Standar Pendidikan Nasional.

Badan Standar Nasional Pendidikan. (2019). *Prosedur Operasional Standar Penyelenggaraan Ujian Nasional Tahun Pelajaran 2018/2019.* Jakarta: Badan Standar Nasional Pendidikan.

Bellare, M., Canetti, R., & Krawczyk, H. (1997). *HMAC: Keyed-Hashing for Message Authentication.* Internet Engineering Task Force, Request for Comments (RFC) 2104.

Deffie, W., & Hellman, M. E. (1976). New Directions in Cryptography . *IEEE Transactions On Infomation Theory*, 1.

Houtven, L. V. (2013). *Crypto 101*. Github.

Information Technology Laboratory. (2008). *The Keyed-Hash Message Authentication Code (HMAC).* Information Technology Laboratory, National Institute of Standards and Technology.

Kementrian Pendidikan dan Kebudayaan. (2020, 06 01). *Ujian Nasional Berbasis Komputer Pusat Penilaian Pendidikan Kementerian Pendidikan Dan Kebudayaan UNBK 2020*. Diambil kembali dari Ujian Nasional Berbasis Komputer Pusat Penilaian Pendidikan Kementerian Pendidikan Dan Kebudayaan UNBK 2020: https://unbk.kemdikbud.go.id/#tentang

Kerchoffs, A. (1883). La Cryptographie Militaire. *Journal des sciences militaires*.

moeljo, S. (2010). *Teori & Aplikasi Kriptografi.* SPK IT Consulting.

A. J., Katz, J., Oorschot, P. C., & Vanstone, S. A. (1997). *Handbook of pplied Cryptography.* CRC Press.

Menteri Pendidikan dan Kebudayaan Republik Indonesia. (2018). *Peraturan Menteri Pendidikan Dan Kebudayaan Republik Indonesia No. 4 Tahun 2018.* Jakarta: Kementrian Pendidikan.

Merkle, R. C. (1979). *Secrecy, Authentication, and Public Key Systems.* Stanford University PhD Dissertation.

Munir, R. (2019). *Kriprografi.* Bandung: Informatika Bandung.

NIST, N. I. (2004). *Secure Hash Standard.* U.S. Department of Commerce.

Paar, C., & Pelzl, J. (2010). *Understanding Cryptography — ATextbook for Students and Practitioners.* New York: Springer Science & Business Media.

Rivest, R., Shamir, A., & Adleman, L. (1977). A Method for Obtaining Digital Signatures and Public- Key Cryptosystems . *Communications of the ACM*.

Schneier, B. (1996). *Applied Cryptography 2nd.* John Wiley & Sons.

Zhang, J., & Jin, X. (2012). Encryption System Design Based On DES And SHA-1. *2012 11th International Symposium on Distributed Computing and Applications to Business, Engineering & Science*.

LAMPIRAN

Lampiran 1. RSABigInteger.java

```
package skripsi.deo;

import java.math.BigInteger;
import java.util.Scanner;

public class RSABigInteger {

    private static final Scanner sc = new Scanner (System.in);

    static BigInteger isPrime (BigInteger number) {

        if (number.compareTo (BigInteger.ONE) < 1)
            return BigInteger.ZERO;

        if (number.compareTo (new BigInteger ("3")) < 1)
            return BigInteger.ONE;

        if (number.mod (new BigInteger ("2")) == BigInteger.ZERO
|| number.mod (new BigInteger ("3")).equals (BigInteger.ZERO)) {
            return BigInteger.ZERO;
        }

        for (BigInteger i = BigInteger.valueOf (5) ; i.multiply
(i).compareTo (number) < 1 ; i = i.add (new BigInteger ("6"))) {
            if (number.mod (i).compareTo (BigInteger.ZERO) == 0 ||
number.mod (i.add (new BigInteger ("2"))).compareTo
(BigInteger.ZERO) == 0) {
                return BigInteger.ZERO;
            }
        }

        return BigInteger.ONE;
    }

    static BigInteger gcd (BigInteger firstNumber, BigInteger
secondNumber) {
        BigInteger r, temp;

        if (firstNumber.compareTo (secondNumber) < 0) {
            temp = firstNumber;
            firstNumber = secondNumber;
            secondNumber = temp;
        }

        while (!secondNumber.equals (BigInteger.ZERO)) {
            r = firstNumber.mod (secondNumber);
            firstNumber = secondNumber;
            secondNumber = r;
        }

        return firstNumber;
```

```
    static BigInteger countPrivateKey (BigInteger e, BigInteger
totient) {
        BigInteger privateKey = new BigInteger ("1"), h, d;

        while (true) {
            d = privateKey.multiply (e);
            h = d.mod (totient);

            if (h.compareTo (BigInteger.ONE) == 0) {
                return privateKey;
            } else {
                privateKey = privateKey.add (BigInteger.ONE);
            }
        }
    }

    static BigInteger[] keyGeneration () {
        BigInteger testPrime, n, d = BigInteger.ZERO, totient;

        System.out.print ("Input p: ");
        BigInteger p = sc.nextBigInteger ();

        testPrime = isPrime (p);

        if (testPrime.compareTo (BigInteger.ZERO) == 0) {
            System.out.println ("Not Prime");
            System.exit (0);
        }

        System.out.print ("Input q: ");
        BigInteger q = sc.nextBigInteger ();

        testPrime = isPrime (q);

        if (testPrime.compareTo (BigInteger.ZERO) == 0) {
            System.out.println ("Not Prime");
            System.exit (0);
        }

        n = p.multiply (q);
        totient = p.subtract (BigInteger.ONE).multiply (q.subtract
(BigInteger.ONE));

        System.out.print ("Public Key (e) = ");
        BigInteger e = sc.nextBigInteger ();

        testPrime = gcd (e, totient);

        if (testPrime.compareTo (BigInteger.ONE) == 0) {
            d = countPrivateKey (e, totient);
        } else {
            System.out.println ("Not relatively prime");
            System.exit (0);
```

```
        }

        return new BigInteger[] {n, e, d};
    }

    final String encryption (String message, BigInteger e,
BigInteger n) {
        long messageLength = message.length ();

        StringBuilder ciphertext = new StringBuilder ();

        long m;
        BigInteger j, result;

        for (long i = 0 ; i < messageLength ; i++) {
            m = message.charAt ((int) i);
            result = BigInteger.ONE;

            for (j = BigInteger.ZERO; j.compareTo (e) < 0 ; j =
j.add (BigInteger.ONE)) {
                result = result.multiply (BigInteger.valueOf (m));
                result = result.mod (n);
            }

            String epp = result.toString (16);

            int nLeght = String.valueOf (n).length ();
            String nol = "0";

            if (epp.length () < nLeght) {
                epp = nol.repeat (nLeght - epp.length ()) + epp;
            }

            ciphertext.append (epp);
        }

        return ciphertext.toString ();
    }

    final String decryption (String encryptMessage, BigInteger d,
BigInteger n) {
        long c;
        BigInteger hasil, j;

        StringBuilder plaintext = new StringBuilder ();

        int nLeght = String.valueOf (n).length ();

        for (int i = 0 ; i < encryptMessage.length () ; i +=
        {
            hasil = BigInteger.ONE;
            String str = encryptMessage.substring (i, i + nLeght);
            c = Integer.parseInt (str, 16);
```

```
            for (j = BigInteger.ZERO; j.compareTo (d) < 0 ; j =
j.add (BigInteger.ONE)) {
                hasil = hasil.multiply (BigInteger.valueOf (c));
                hasil = hasil.mod (n);
            }

            plaintext.append ((char) Integer.parseInt
(String.valueOf (hasil)));
        }

        return plaintext.toString ();
    }
}
```

Lampiran 2. HMAC.java

```
package skripsi.deo;

public class HMAC {
    SHA1 sha1 = new SHA1 ();
    private static final int B = 64;
    private static final byte[] IPAD = new byte[B];
    private static final byte[] OPAD = new byte[B];

    public HMAC () {
        for (int i = 0 ; i < B ; i++) {
            IPAD[i] = 0x36;
            OPAD[i] = 0x5c;
        }
    }

    byte[] hashMessage (String arr) {
        String hash_string_pertama = sha1.sha1 (arr);
        return stringToByte (hash_string_pertama);
    }

    byte[] stringToByte (String arr) {
        byte[] val = new byte[arr.length () / 2];

        for (int i = 0 ; i < val.length ; i++) {
            int index = i * 2;
            int j = Integer.parseInt (arr.substring (index, index
+ 2), 16);
            val[i] = (byte) j;
        }

        return val;
    }

    String processMessage (String message, String key) {
        byte[] k = (key).getBytes ();
        byte[] messageInByte = (message).getBytes ();
        byte[] k0 = new byte[B];

        if (k.length == B) {
            k0 = k;
        } else {
            if (k.length > B) {
                byte[] k_append = hashMessage (key);
                System.arraycopy (k_append, 0, k0, 0,
k_append.length);
                int i = k0.length;
                while (i < B) {
                    k0[i] = (byte) 0x00;
                    i++;
                }
            } else {
                System.arraycopy (k, 0, k0, 0, k.length);
```

```
                for (int i = k.length ; i < B ; i++) {
                    k0[i] = (byte) 0x00;
                }
            }
        }

        /** K0 ⊕ ipad */
        byte[] innerKey = new byte[B];
        for (int i = 0 ; i < B ; i++) {
            innerKey[i] = (byte) (k0[i] ^ IPAD[i]);
        }

        /** K0 ⊕ ipad || Message */
        byte[] firstHashWithMessage = new byte[innerKey.length +
messageInByte.length];
        System.arraycopy (innerKey, 0, firstHashWithMessage, 0,
innerKey.length);
        System.arraycopy (messageInByte, 0, firstHashWithMessage,
innerKey.length, messageInByte.length);

        /** Hash(K0 ⊕ ipad || Message) */
        String stringHash = sha1.sha1_byte (firstHashWithMessage);
        byte[] firstHash = stringToByte (stringHash);

        /** K0 + opad */
        byte[] outerKey = new byte[B];
        for (int i = 0 ; i < B ; i++) {
            outerKey[i] = (byte) (k0[i] ^ OPAD[i]);
        }

        /** (K0 ⊕ opad) || Hash(K0 ⊕ ipad || Message ) */
        byte[] secondHashWithMessage = new byte[outerKey.length +
firstHash.length];
        System.arraycopy (outerKey, 0, secondHashWithMessage, 0,
outerKey.length);
        System.arraycopy (firstHash, 0, secondHashWithMessage,
outerKey.length, firstHash.length);

        /** Hash((K0 ⊕ opad) || Hash(K0 ⊕ ipad || Message )) */
        return sha1.sha1_byte (secondHashWithMessage);
    }
}
```

Lampiran 3. SHA-1.java

```
package skripsi.deo;

public class SHA1 {
    static int temp;
    static int A, B, C, D, E, F;

    /**
     * SHA-1 untuk memproses String
     */
    public String sha1 (String msg) {
        Hash hash = new Hash ();
        byte[] dataBuffer = (msg).getBytes ();
        return hash.reset (dataBuffer);
    }

    /**
     * SHA-1 untuk memproses Byte
     */
    public String sha1_byte (byte[] msg) {
        Hash hash = new Hash ();
        return hash.reset (msg);
    }

    public class Hash {
        public String reset (byte[] data) {
            byte[] paddedMessage = padMessage (data);

            int[] H = {0x67452301, 0xEFCDAB89, 0x98BADCFE,
0x10325476, 0xC3D2E1F0};
            int[] K = {0x5A827999, 0x6ED9EBA1, 0x8F1BBCDC,
0xCA62C1D6};

            if (paddedMessage.length % 64 != 0) {
                System.out.println ("Invalid padded data
length.");
                System.exit (0);
            }

            int numbersBlocks = paddedMessage.length / 64;
            byte[] W = new byte[64];

            for (int i = 0 ; i < numbersBlocks ; i++) {
                System.arraycopy (paddedMessage, 64 * i, W, 0,
64);
                processMessagebyBlock (W, H, K);
            }

            return intArrayToHexString (H);
        }

        private byte[] padMessage (byte[] message) {
            int messageLength = message.length;
```

```
            int tailLength = messageLength % 64;
            int appendLength;

            if ((64 - tailLength >= 9)) {
                appendLength = 64 - tailLength;
            } else {
                appendLength = 128 - tailLength;
            }

            byte[] append = new byte[appendLength];
            append[0] = (byte) 0x80;
            long lengthInBits = messageLength * 8;

            for (int i = 0 ; i < 8 ; i++) {
                append[append.length - 1 - i] = (byte)
((lengthInBits >> (8 * i)) & 0x00000000000000FF);
            }

            byte[] paddedMessage = new byte[messageLength +
appendLength];

            System.arraycopy (message, 0, paddedMessage, 0,
messageLength);
            System.arraycopy (append, 0, paddedMessage,
messageLength, append.length);

            return paddedMessage;
        }

        private void processMessagebyBlock (byte[] w, int[] H,
int[] K) {
            int[] W = new int[80];

            for (int outer = 0 ; outer < 16 ; outer++) {
                int temp;
                for (int inner = 0 ; inner < 4 ; inner++) {
                    temp = (w[outer * 4 + inner] & 0x000000FF) <<
(24 - inner * 8);
                    W[outer] = W[outer] | temp;
                }
            }

            for (int j = 16 ; j < 80 ; j++) {
                W[j] = circularShift (W[j - 3] ^ W[j - 8] ^ W[j -
14] ^ W[j - 16], 1);
            }

            A = H[0];
            B = H[1];
            C = H[2];
            D = H[3];
            E = H[4];

            /** Round 1 */
```

```
            for (int j = 0 ; j < 20 ; j++) {
                F = (B & C) | ((~B) & D);
                temp = circularShift (A, 5) + F + E + K[0] + W[j];
                E = D;
                D = C;
                C = circularShift (B, 30);
                B = A;
                A = temp;
            }

            /** Round 2 */
            for (int j = 20 ; j < 40 ; j++) {
                F = B ^ C ^ D;
                temp = circularShift (A, 5) + F + E + K[1] + W[j];
                E = D;
                D = C;
                C = circularShift (B, 30);
                B = A;
                A = temp;
            }

            /** Round 3 */
            for (int j = 40 ; j < 60 ; j++) {
                F = (B & C) | (B & D) | (C & D);
                temp = circularShift (A, 5) + F + E + K[2] + W[j];
                E = D;
                D = C;
                C = circularShift (B, 30);
                B = A;
                A = temp;
            }

            /** Round 4 */
            for (int j = 60 ; j < 80 ; j++) {
                F = B ^ C ^ D;
                temp = circularShift (A, 5) + F + E + K[3] + W[j];
                E = D;
                D = C;
                C = circularShift (B, 30);
                B = A;
                A = temp;
            }

            H[0] += A;
            H[1] += B;
            H[2] += C;
            H[3] += D;
            H[4] += E;
        }

        al int circularShift (int value, int bits) {
         return (value << bits) | (value >>> (32 - bits));
```

```
    private String intArrayToHexString (int[] data) {
        StringBuilder output = new StringBuilder ();
        String tempStr;
        int tempInt;

        for (int datum : data) {
            tempInt = datum;
            tempStr = Integer.toHexString (tempInt);

            if (tempStr.length () == 1) {
                tempStr = "0000000" + tempStr;
            } else if (tempStr.length () == 2) {
                tempStr = "000000" + tempStr;
            } else if (tempStr.length () == 3) {
                tempStr = "00000" + tempStr;
            } else if (tempStr.length () == 4) {
                tempStr = "0000" + tempStr;
            } else if (tempStr.length () == 5) {
                tempStr = "000" + tempStr;
            } else if (tempStr.length () == 6) {
                tempStr = "00" + tempStr;
            } else if (tempStr.length () == 7) {
                tempStr = "0" + tempStr;
            }

            output.append (tempStr);
        }
        return output.toString ();
    }
}
```

Lampiran 4. Main.java

```
package skripsi.deo;

import java.math.BigInteger;
import java.time.Duration;
import java.time.Instant;
import java.util.Random;
import java.util.Scanner;

public class Main {
    private static final Scanner sc = new Scanner (System.in);
    private static final RSABigInteger rsa = new RSABigInteger ();
    private static final HMAC hmac = new HMAC ();
    private static BigInteger e, d, n;

    private static void keyGeneration () {
        System.out.println ("HMAC Key = " + HMACKey ());
        BigInteger[] key = rsa.keyGeneration ();
        n = key[0];
        e = key[1];
        d = key[2];

        System.out.println ("Publick Key = " + e + "," + n);
        System.out.println ("Private Key = " + d + "," + n);
    }

    private static String HMACKey () {
        Random random = new Random ();

        String alphabet =
"0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ,.;
:'{}-_=+!@#$%^&*()<>`~|", HMACKeyRandom = "";
        for (int i = 0 ; i < 64 ; i++) {
            HMACKeyRandom += alphabet.charAt (random.nextInt
(alphabet.length ()));
        }

        return HMACKeyRandom;
    }

    private static String senderSide () {
        System.out.print ("Insert answer: ");
        String word = sc.next ();

        System.out.print ("Input e = ");
        BigInteger e = sc.nextBigInteger ();
        System.out.print ("Input n = ");
        BigInteger n = sc.nextBigInteger ();

        String encryptMessage = rsa.encryption (word, e, n);

        System.out.print ("Insert HMAC key: ");
        String key = sc.next ();
```

```
            String hmacMessage = hmac.processMessage (word, key);

            return (encryptMessage + hmacMessage);
        }

        private static String receiverSide () {
            System.out.print ("Insert Message: ");
            String encryptMessage = sc.next ();

            System.out.print ("Input d = ");
            BigInteger d = sc.nextBigInteger ();
            System.out.print ("Input n = ");
            BigInteger n = sc.nextBigInteger ();

            System.out.print ("Insert HMAC key: ");
            String key = sc.next ();

            String splitHMAC = splitHMACMessage (encryptMessage);
            String splitRSA = splitEncryptMessage (encryptMessage);

            String decryptMessage = rsa.decryption (splitRSA, d, n);
            String hmacMessage = hmac.processMessage (decryptMessage,
key);

            if (hmacMessage.equals (splitHMAC)) {
                System.out.println ("Message has been verified");
                return decryptMessage;
            } else {
                System.out.println ("Message not verified");
                System.exit (1);
                return null;
            }
        }

        private static String splitEncryptMessage (String
encryptMessage) {
            String result = "";
            int lenght = encryptMessage.length ();

            while (lenght <= 40) {
                System.out.println ("Wrong input");
                System.exit (1);
            }

            for (int i = 0 ; i < (lenght - 40) ; i++) {
                result += encryptMessage.charAt (i);
            }

            return result;


        vate static String splitHMACMessage (String encryptMessage)
```

```
            String result = "";
            int lenght = encryptMessage.length ();

            while (lenght <= 40) {
                System.out.println ("Wrong input");
                System.exit (1);
            }

            for (int i = (lenght - 40) ; i < lenght ; i++) {
                result += encryptMessage.charAt (i);
            }

            return result;
        }

        public static void main (String[] args) {
            String sendMessage;

            while (true) {
                System.out.println ("1. Key Generation\n" + "2.
Encryption & HMAC\n" + "3. Decryption & Verification\n" + "4.
Exit");

                int choose = sc.nextInt ();

                switch (choose) {
                    case 1:
                        keyGeneration ();
                        break;

                    case 2:
                        sendMessage = senderSide ();
                        System.out.println (sendMessage);
                        break;

                    case 3:
                        System.out.println (receiverSide ());
                        break;

                    case 4:
                        System.exit (1);

                    default:
                        System.out.println ("Wrong input");
                }
            }
        }
    }
```

Lampiran 5. Visualisasi Contoh Operasi Enkripsi dan Penghitungan Nilai HMAC serta Operasi Dekripsi dan Verifikasi